DESIGN METHODOLOGY ADDRESSING STATIC/RECONFIGURABLE PARTITIONING FOR OPTIMIZING SOFTWARE DEFINED RADIO (SDR) IMPLEMENTATION THROUGH FPGA DYNAMIC PARTIAL RECONFIGURATION AND RAPID PROTOTYPING TOOLS

RAÚL TORREGO ARTOLA

Supervisors:
Iñaki Val
Eñaut Muxika

A thesis submitted for the degree of
Doctor by Mondragon Unibertsitatea

Department of Electronics and Computer Science
Mondragon Unibertsitatea
November 2012

STATEMENT OF ORIGINALITY

I hereby declare that the research recorded in this thesis and the thesis itself were developed entirely by myself at the Signal Theory and Communications Area, Department of Electronics and Computer Science, at Mondragon Unibertsitatea.

Raúl Torrego Artola

November 2012

ACKNOWLEDGEMENTS

I would like to thank to all those who have contributed in any way to make this thesis possible. Thank you!

ABSTRACT

The characteristics people request for communication devices become more and more demanding every day. And not only in those aspects dealing with communication speed, but also in such different characteristics as different communication standards compatibility, battery life, device size or price. Moreover, when this communication need is addressed by the industrial world, new characteristics such as reliability, robustness or time-to-market appear. In this context, Software Defined Radios (SDR) and evolutions such as Cognitive Radios or Intelligent Radios seem to be the technological answer that will satisfy all these requirements in a short and mid-term.

Consequently, this PhD dissertation deals with the implementation of this type of communication system. Taking into account that there is no limitation neither in the implementation architecture nor in the target device, a novel framework for SDR implementation is proposed. This framework is made up of FPGAs, using dynamic partial reconfiguration, as target device and rapid prototyping tools as designing tool. Despite the benefits that this framework generates, there are also certain drawbacks that need to be analyzed and minimized to the extent possible. On this purpose, a SDR design methodology has been designed and tested. This methodology addresses the static/reconfigurable partitioning of the SDRs in order to optimize their implementation in the aforementioned framework.

In order to verify the feasibility of both the design framework and the design methodology, several implementations have been carried out making use of them. A multi-standard modulator implementing WiFi, WiMAX and UMTS, a small-form-factor cognitive video transmission system and the implementation of several data coding functions over R3TOS, a hardware operating system developed by the University of Edinburgh, are these implementations.

CONTENTS LIST

LIST OF ILLUSTRATIONS

LIST OF TABLES

GLOSSARY AND ABBREVIATIONS

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
Baud	Symbol per second
BUFG	Global Clock Buffer
BUFR	Regional Clock Buffer
CCTV	Closed-Circuit Television
CLB	Configurable Logic Block
CPI	Cyclic Prefix Inserter
DAC	Digital-to-Analog Converter
DCM	Digital Clock Manager
DSL	Domain Specific Language
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
EDA	Electronic Design Automation
ESL	Electronic System Level
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
HDL	Hardware Description Language
HWS	Hardware Semaphore (R3TOS's task synchronization resource)
I2CI	ICAP-based Inter-task Communication Infrastructure
ICAP	Internal Configuration Access Port
ICT	Information and Communication Technology

IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
JTAG	Joint Test Action Group
LAPU	Live-At-Power-Up
LUT	Look-Up Table
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
LVTTL	Low Voltage Transistor Transistor Logic
MAC unit	Multiply-Accumulate unit
MBDK	Model-Based Design Kit
MIG	Memory Interface Generator
N	The total number of different configurations of a certain design. That is, the total number of waveforms present in the SDR
n	Each of the configurations/waveforms of a design. *n* = 1, 2, 3, ... N
NCO	Numerically Controlled Oscillator
OFDM	Orthogonal Frequency Division Multiplexing
OQPSK	Offset Quadrature Phase Shift Keying
OTP	One-Time Programmable
P	The total number of different partitions obtained for a certain design
p	Each of the partitions of a design. *p* = 1, 2, 3, ... P
PC	Personal Computer
PCAP	Parallel Configuration Access Port
PCI	Peripheral Component Interconnect
PLL	Phase-Locked Loop
PRA	Partially Reconfigurable Area
PRBS	Pseudo Random Binary Sequence
PRM	Partially Reconfigurable Module
PROM	Programmable Read Only Memory
PSD	Power Spectral Density
R3TOS	Reliable Reconfigurable and Real-time Operating System

RF	Radio Frequency
SDR	Software Defined Radio
SerDes	Serializer-Deserializer
SEU	Single Event Upset
SoC	System on Chip
SOFDMA	Scalable Orthogonal Frequency-Division Multiple Access
UMTS	Universal Mobile Telecommunications System
UPaRC	Ultra-fast Power-aware Reconfiguration Controller
Waveform	Each of the different radio protocols, communication standards or functions in charge of the communication present in a Software Defined Radio
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Acces
ZBTRAM	Zero Bus Turnaround Random Access Memory

Chapter 1

Introduction

1. Introduction

1. MOTIVATION

We belong to the information society where the trend is towards people being connected, one way or another and in an uninterrupted way, both with other people and to the Internet. The necessity to share and transmit data grows exponentially, so the features users demand from communication devices become more and more exigent every day. These requirements go beyond the simple fact of a bigger bandwidth and concern other factors such as size, battery life, compatibility with various communication standards, update capability and, of course, cost. This communication eagerness not only affects consumer environments but also industrial ones. Up to date, wired communication systems were the favourite ones in this type of environments due to the robustness and reliability they offer when dealing with harsh environments. However, a new tendency towards replacing these wired communication systems by wireless ones is emerging. These wireless communication systems have to achieve, at least, the same performance the wired systems reach, therefore they have to fulfil the aforementioned reliability and robustness. For both, the consumer and industrial environments, the technology that nowadays seems to meet the requirements stated above is the Software Defined Radios (SDR).

Software Defined Radios are digital, wireless, communication systems with reconfigurable nature. By definition, there is no restriction in the type of parameters or characteristics that can be reconfigured. Therefore, this type of radio supports the change of multiple features; from the update of the simplest parameter (i.e. the gain of an amplifier) to a complete communication standard change. This reconfiguration capability is the one that makes them ideal for accomplishing the aforementioned requirements. Furthermore, the implementation of these radios is not restricted to any technology; hence, the developers can choose the implementation technology that better suits their expertise, time-to-market or fabrication costs. The first objective of this thesis focuses on proposing an implementation framework for Software Defined Radios. This framework is formed by the dynamic partial reconfiguration of the FPGAs and rapid prototyping tools.

In the field of high speed data processing, FPGAs are becoming the alternative to microprocessors and classical signal processing architectures. A mixture of factors such as their high processing capacity due to their hardware oriented programming, their reconfigurable nature and new characteristics like dynamic partial reconfiguration, together with the decrease in the price that FPGAs have had in the last years, contribute to this spread. It is precisely this last characteristic, the dynamic partial reconfiguration, which makes the FPGAs an optimum candidate for the implementation of SDRs. Dynamic partial reconfiguration is defined as the possibility of changing the configuration of part of the FPGA while the rest of it remains working. Consequently the link of SDRs with this characteristic is clear.

On the other hand, rapid prototyping tools are programs developed in order to simplify and ease the use of traditional compilers and code synthesizers. These tools complement the compilers with new

features such as graphical programming, functional simulations in early design steps, resource estimations or automatic code generation. The joint of all these characteristics leads definitely to an important saving in the overall design time. SDRs being complex communication systems, the design, development and test of signal processing algorithms are a very important parts of the design flow. Therefore, the link between SDR implementation and the use of rapid prototyping tools is evident again.

Unfortunately, the use of this SDR development environment composed by FPGA dynamic partial reconfiguration and rapid prototyping tools, besides multiple benefits, also introduces several inconveniences. These inconveniences have to be evaluated and solved as far as possible so that they do not have a significant influence in the final implementation. Among them, the reconfiguration time in which the area to be partially reconfigured is not functional and the fact that the code generated by the rapid prototyping tools is not as optimal as hand-made code would be, are two of the more relevant. In order to minimize the impact of these inconveniences, and looking for the achievement of high performance designs, the use of design methodologies is a widely used solution. As will be presented in the next chapter, there are multiple design methodologies present in the literature that look for the optimization of the designs following several approaches and focusing on diverse system factors. Among them, and at the best of the author's knowledge, none of them consider the partitioning of which parts of the design should be implemented statically and which in a reconfigurable way in order to obtain the highest performance. This thesis proposes a design methodology that addresses the static/reconfigurable partitioning for optimizing SDR implementation through FPGA rapid prototyping tools and dynamic partial reconfiguration and investigates all the implications that such methodology and design environment generate.

2. OBJECTIVES

There are two main objectives that this thesis pursues. On the one hand the presentation and validation of a design framework for the implementation of Software Defined Radios composed of FPGA dynamic partial reconfiguration and rapid prototyping tools. Afterwards, and with the aim of optimising the implementations carried out over this design framework, the second objective considers the development of a design methodology. This design methodology evaluates the characteristics of the SDR to be implemented and chooses the optimum static/reconfigurable partitioning.

3. CONTRIBUTION

The major contributions of this thesis are split into seven key aspects.

1. A novel design framework for the implementation of Software Defined Radios has been proposed and tested. This design framework is made up of FPGA dynamic partial reconfiguration and rapid prototyping tools. The drawbacks and design implications of this framework have been analyzed and evaluated, coming to the conclusion that, despite the

multiple benefits that it offers, a design methodology is needed in order to optimize the obtained implementations.

2. A complete design methodology that addresses static/reconfigurable partitioning for optimizing the SDR implementation through the aforementioned design framework has been designed. The methodology selects which parts of the SDR are to be implemented statically and which in a reconfigurable way in order to minimize system's cost.

3. A cost function has been developed in order to quantitatively evaluate a certain partition's optimality. The function evaluates those design factors that have been determined as most important in the performance of a design. Namely: design size, reconfiguration time and maximum applicable clock frequency.

4. A multi-standard modulator implementing three of the most used communication standards nowadays (WiFi, WiMAX and UMT) has been designed and implemented using the proposed design framework and design methodology. This implementation serves as a proof-of-concept and as a validation of the first and second contributions.

5. A small-form-factor cognitive video transmission system has been implemented using the proposed design framework. This system is able to change its Intermediate Frequency (IF) if the transmission channel is occupied, hence achieving a secure communication. The frequency change is carried out via dynamic partial reconfiguration. It provides a tangible demonstration of the feasibility of the proposed framework with a fully functional application.

6. The implementation of several data coding functions used in SDRs has been carried out over R3TOS, a Reliable, Reconfigurable and Real-Time hardware Operating System developed by the University of Edinburgh [Iturbe'10]. The design of the functions has been carried out using rapid prototyping tools and applying some of the guidelines proposed by the design methodology. Therefore the possibility of using this methodology in other reconfigurable architectures is demonstrated.

7. A novel task context saving and restoration procedure has been designed for its use with R3TOS. This feature, similar to context saving and restoration performed in software processors, requires special care when dealing with FPGA dynamic partial reconfiguration as some issues related with Flip-Flop initialization appear.

4. THESIS STRUCTURE

This thesis is structured into eight chapters. This first chapter has provided the reader with a general introduction to the thesis' topic and has presented the motivation, objectives and main contributions of this research work.

Chapter 2 contains descriptions of the background and existing literature. It provides an overview to all the aspects that take part in this research work. Taking the Software Defined Radios as starting

and key point, the chapter summarizes the different works and investigations reported in the literature around the SDRs itself, the FPGAs and their dynamic partial reconfiguration, the rapid prototyping tools and the diverse design methodologies that try to optimize this type of implementations.

Chapter 3 presents the design methodology that has been developed. Firstly the degree of freedom over which the methodology will act is decided and explained. Subsequently, the several steps that make up the design flow of the methodology are exposed. Finally, the basics of an automation procedure for the methodology that aims to ease the design flow and hence reduce the design time is presented.

Chapters 4 and 5 thoroughly describe the two key steps within the design methodology, i.e. the establishment of common parts and the evaluation of system's performance via a cost function. Chapter 4 takes care of the first step. It initially presents the original the "Common functions / common operators" technique that aims to search for and find all the commonalities between several different standards. This technique is later adapted for its use on the design framework presented by this thesis. Subsequently "Function parameterization" is presented. This procedure is in charge of making the similar parts of functions in the design really compatible. In turn Chapter 5 deals with the cost function that quantitatively evaluates the optimality of each of the generated design partitions. This chapter also presents other auxiliary functions that make an estimation of those design factors that are not available in early design steps.

Chapters 6 and 7 provide the reader with a set of implementations that serve as proof-of-concept of the feasibility of both the proposed design framework and design methodology. Chapter 6, on the one hand, present the development of a multi-standard modulator that implements three of the most used communication standards nowadays. Namely: WiFi, WiMAX and UMTS. This multi-standard modulator, designed following all the steps within the design methodology serves as a validation for it. On the other hand, Chapter 7 presents two implementations that, although do not completely follow all the guidelines proposed by the design methodology, show the viability of the design framework via real functional applications. The two presented implementations are: a small-form-factor cognitive video transmission system, and the implementation of several data coding functions used in SDRs over R3TOS, a Reliable, Reconfigurable and Real-Time hardware Operating System developed by the University of Edinburgh. In regards to this last implementation, this chapter also presents a novel task context saving and restoration procedure has been designed for its use with R3TOS.

Finally, concluding remarks and future works that can extend the results presented in this thesis are drawn in Chapter 8. Annex 1 compiles certain information that, even though is necessary to properly complete this thesis, do not have place within the normal dissertation.

Chapter 2

State of the Art

2. State of the Art

1. INTRODUCTION

This chapter covers a compilation of the state of the diverse technologies that take part in the topics that this thesis investigates. Taking the Software Defined Radios as starting and key point, the chapter summarizes the different works and investigations reported in the literature around the SDRs itself, the FPGAs and their dynamic partial reconfiguration, the rapid prototyping tools and the diverse design methodologies that try to optimize this type of implementations. On this basis, a novel design framework for SDR implementation and a design methodology that addresses static/reconfigurable partitioning are presented. Precisely the chapter concludes with the introduction to this methodology's objectives and with the initial hypothesis that motivates its development.

2. SOFTWARE DEFINED RADIOS

The term “Software Radio” was used for the first time by Joseph Mitola [Mitola'92, Mitola'95] in 1992 for referring to reconfigurable or reprogrammable radios. In radios a single piece of hardware could have different functionalities in different times with just the introduction of software configuration changes. The origins of these "Software Radios" are the digital radios: communication systems in which some parts that had been traditionally implemented with analog hardware devices (mixers, filters, amplifiers... etc.) are carried out in a digital way. This digital implementation can refer both to software systems such as processors or DSPs or to 'pure' digital hardware devices like FPGAs, ASICs and so on. Taking advantage of the reconfigurability that software applications and digital hardware devices have, its logical evolution provides these radios with the ability of changing their characteristics, leading to Software Defined Radios, commonly known as SDR.

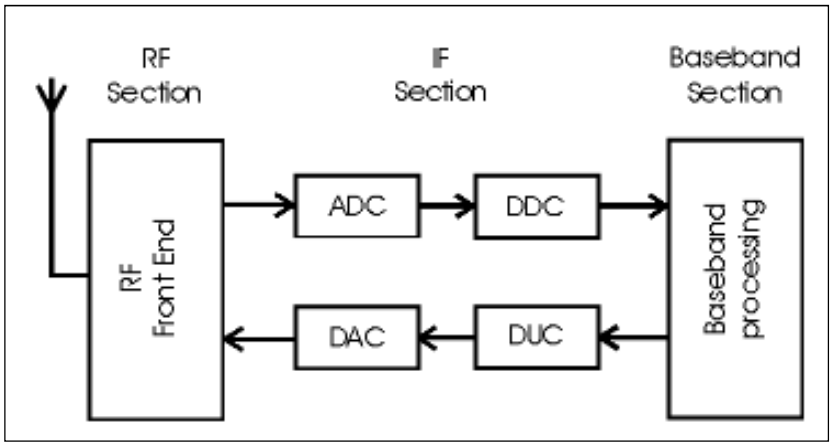


Figure 1: Basic scheme of a SDR

In Figure 1 the basic scheme of a SDR can be observed. Ideally the analog-to-digital (ADC) and digital-to-analog (DAC) converters should be directly connected to the antenna so that the whole

processing could be digitally accomplished. Unfortunately, there is a limitation with the maximum frequency at which these converters can work. Taking into account that the radio frequency (RF) used in this kind of system is usually placed above the Gigahertz, the converters able to properly work at this frequency are scarce, expensive or just under development. Therefore, the typical structure that SDRs have nowadays includes an analog RF front-end that reduces this RF frequency. This reduction can be made to an intermediate frequency (IF), like in the figure, or directly to baseband, what is called zero-IF or direct conversion systems.

Once in the digital domain, the functionality changes, or the parameter updates mentioned in the definition of what a SDR is, are not limited in any way. Therefore, they span from small granularity changes in which just some parameters such as gains or filter coefficients are updated, to big reconfigurations where a whole communication standard (usually known as 'waveforms') is replaced by another (i.e. from WiFi to WiMAX). Taking advantage of these possibilities, the use of Software Defined Radios solves many of the requirements that users demand to communication systems and that have been expounded in the introduction. The hardware reuse that is achieved due to the reconfiguration capacity that SDRs have, leads directly to a reduction in the size of the devices needed to make up the implementation. This size reduction entails a reduction in both the power consumption of the device and, usually, in its price, what in turn has influence in the battery life and in the final price of the communication system. On the other hand, the compatibility with various communication standards and the update capability are also fulfilled by SDRs and their reconfigurability.

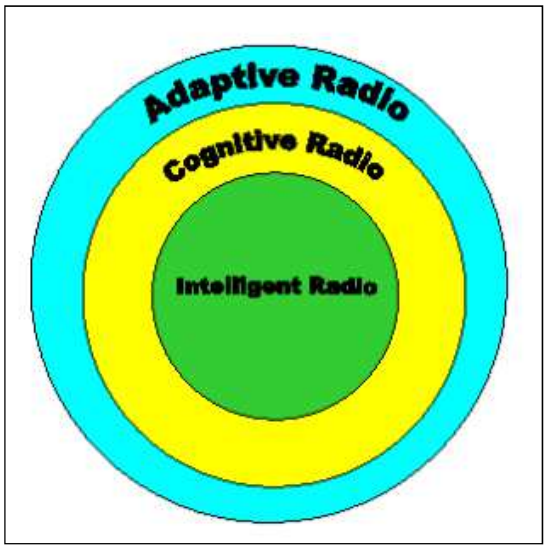


Figure 2: Evolution of Software Defined Radios

Going a bit further, Software Defined Radios are the base for other novel architectures for communication systems like Adaptive Radios, Cognitive Radios or Intelligent Radios [WIINN-FORUM'12]. Figure 2 shows the evolution of these communication technologies. Adaptive Radios monitor its own performance and are able to change, using the SDR technology, some of its characteristics to improve it without user's intervention. Cognitive Radios behave in a similar way but,

apart from monitoring its own performance, they are aware of the characteristics of the environment they are working in and change their parameters accordingly. One of the most typical characteristics measured by Cognitive Radios is the availability of the RF spectrum. This way Cognitive Radios can detect which wireless channels are available and change their transmission frequency to them in order to improve the quality of the communication link. This process is also known as dynamic spectrum management. Finally, Intelligent Radios are an evolution of Cognitive Radios that add them the ability to learn. That is, Intelligent Radios are aware of their environment and also of their previous actuations. Definitively, all these evolutions of SDRs add them certain intelligence that looks for the improvement of the success probability of the data transmissions. Going back to the requirements explained in the introduction, these types of radios fit perfectly with the robustness and reliability demanded by industrial communications.

2.1. Architectures for Software Defined Radios

The implementation of Software Defined Radios is not limited in any way by the definition. Therefore there are plenty of systems, architectures or frameworks that can hold this kind of communication systems. A short survey of them is presented below:

Bearing in mind the above presented SDR evolution from the original "digital radios" to the current concepts, one of first and still most common implementation architecture is the DSP. By way of example Nicollet and Demeure [Nicollet'03] present a an innovative software architecture for modem and audio applications embedded in Digital Signal Processors. For his part, Schoenes et al. [Schoenes'03] have developed a new DSP architecture for baseband processing in multistandard software-defined radios. The designed single-instruction multiple-data (SIMD) architecture provides best support for vector-oriented algorithms commonly used in many signal processing applications. In turn, Boch et al. [Boch'06] analyze the digital and reconfigurable implementation of selected signal processing algorithms using a commercial DSP.

FPGAs are other of the favourite devices for the implementation of Software Defined Radios, both in a stand-alone mode or together with other devices. Mecwan and Gajjar [Mecwan'11] propose the design of a transmitter and receiver on a reconfigurable platform like a FPGA, so that the modulation scheme can be dynamically adapted depending on the noise in the communication medium. Saha and Sinha [Saha'09], in turn, present a FPGA based architecture of a novel reconfigurable radio processor for SDRs. The proposed architecture acts essentially as a reconfigurable accelerator for radio functions and works in conjunction with a main CPU (i.e. a DSP or a GPP). Harnessing the opportunities that the FPGA market offers, Kuo et al. [Kuo'12] present uSDR, a compact, inexpensive, and battery-powered SDR platform built on a single-chip from Actel. This chip is made up of a flash-based FPGA fabric and ARM Cortex-M3 processor.

There are also in the literature several custom-made devices for the implementation of SDRs. RICA [Khawam'08, Zong'08], presented by Khawam et al. and used by Zong and Arslan in the development of SDRs, is a reconfigurable instruction cell array. Designing the silicon fabric in a similar way to reconfigurable arrays but with a closer equivalence to software, the same high

performance as coarse-grain FPGA architectures is achieved and the same flexibility, low cost, and programmability as DSPs is maintained. Similarly SODA, presented by Lin et al. [Lin'07] is described as a high-performance DSP architecture for SDR. It is a multiprocessor architecture which consists of multiple processing elements, a scalar control processor, and global scratch-pad memory, all connected through a shared bus.

2.2. Novel design framework for the implementation of Software Defined Radios

Analyzing a common, current definition of what an SDR is, e.g.: "*Communication system where a single piece of hardware has different functionalities in different times*" [Rappaport'01] we can get an idea of which could be an interesting SDR implementation framework. On the one hand, "Different functionalities in different times" is part of the definition of what FPGA dynamic partial reconfiguration is. Therefore, the use of dynamic partial reconfiguration seems a promising opportunity. On the other hand, the design of a "Communication system" has to face with tasks such as algorithm design, coding and debugging, realization of performance tests, optimization of the final code and so on. Rapid prototyping tools are the perfect software tools for doing all these tasks in an efficient way so their use in the design of SDRs seems feasible.

Ultimately, this thesis proposes at first instance a design framework for the implementation of Software Defined Radios based on FPGA dynamic partial reconfiguration and rapid prototyping tools with the intention of benefiting from the advantages these two technologies offer. Figure 3 shows a first outline of the sought out integration.

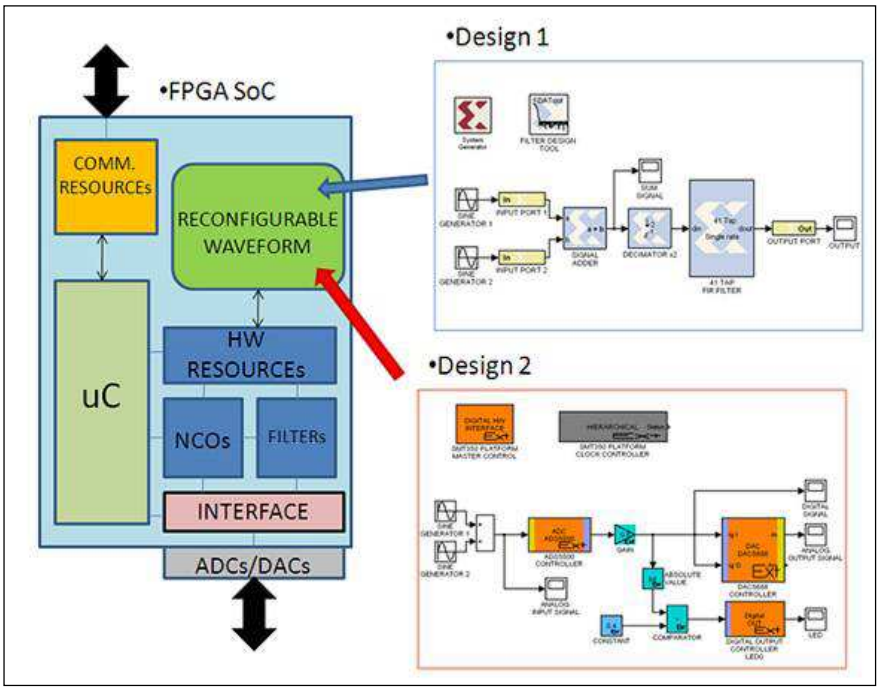


Figure 3: Outline of the integration of FPGA dynamic partial reconfiguration and rapid prototyping tolos in the design of Software Defined Radios

The subsequent sections will discuss the characteristics, advantages and disadvantages that these technologies have.

3. FPGAS AND DYNAMIC PARTIAL RECONFIGURATION

Field Programmable Gate Arrays (FPGAs) are nowadays the reprogrammable logic devices par excellence. They can implement any type of digital circuit, from the most simple logic gate to the most complex System on Chip (SoC) made up of several processors, peripherals and glue logic in a single device [Astarloa'05]. This is possible due to the particular structure they have (Figure 4) made up of an interconnection matrix that communicates thousands of Configurable Logic Blocks (named CLB in Xilinx's FPGAs). These blocks constitute the heart of the FPGA and are typically formed by a combinational programmable circuit in the form of a Look-up Table (LUT), a bistable component (latch or Flip-Flop) and additional logic for internal signal routing. Besides, CLBs usually complete with memory cells that store the configuration of the aforementioned components. However, this feature is dependent on the configuration technology of the FPGAs as will be analyzed later on. Additionally, FPGAs are provided with dedicated Input-Output Blocks (IOBs) as well as diverse extra resources directly implemented in silicon and connected to the interconnection matrix. These resources such as multipliers, RAM memory blocks, DSPs, communication controllers and even microcontrollers like PowerPC or ARM [XILINX'12e] give the FPGAs the ability of implementing high performance functions without the inefficient consumption of the programmable logic. The distribution and amount of these resources change between the different FPGA families or manufacturers with the aim of covering the different application necessities. It is precisely the inclusion of these new possibilities in the FPGAs together with their gradual price reduction one of the reasons for their increasing popularity.

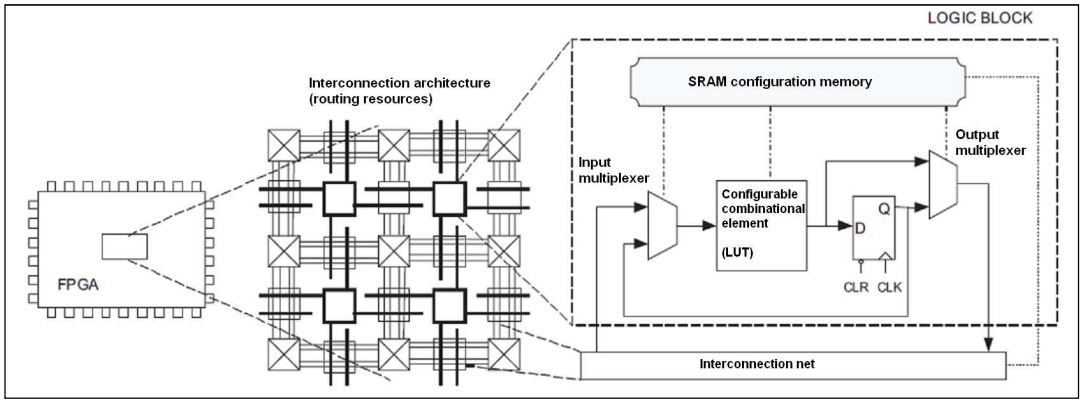


Figure 4: Generic diagram of a SRAM FPGA

Moreover, due to their own nature and abovementioned structure, FPGAs allow the implementation of multiple functions or systems in parallel, and, therefore, the achievement of very high data processing speeds. As can be seen in Figure 5 [PENTEK'10], this high speed processing capability, together with the flexibility FPGAs offer have made them find their own place in the Software Defined Radio application task space.

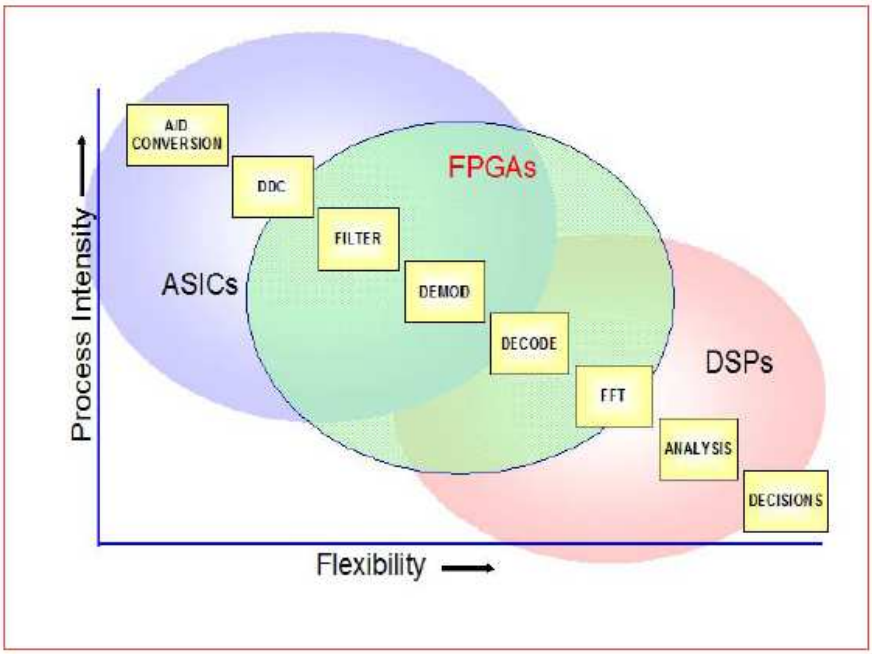


Figure 5: FPGAs in the SDR application task space

3.1. Configuration technologies

The classification of the different types of FPGAs present in the market can be done attending to several factors such as internal structure, granularity (size of the smallest functional unit that synthesis and routing tools can address and, therefore, a measurement of the flexibility they offer), power consumption or number and distribution of the aforementioned silicon-implemented resources. However, from the point of view that this thesis faces the use of FPGAs in the implementation of SDRs, the most interesting classification focuses on the different configuration technologies of the FPGAs [Cofer'05, Wolf'04]. The possibility of dynamically reconfiguring the FPGA directly depends on these configuration technologies.

3.1.1. Anti-fuse FPGAs

Anti-fuse FPGAs are One-Time Programmable (OTP) devices. That is, once they are programmed they maintain their internal configuration indefinitely and cannot be reprogrammed. This is due to the use of the Anti-fuse technology for the carrying out of the interconnections needed for the configuration of the FPGA. This technology (called anti-fuse because it works in the inverse way a fuse does) instead of melting the connections that are not used generates new permanent connections when programming (Figure 6). This way, taking into account that the programming of the FPGA creates new physical connections, it is not possible to reprogram it, and therefore, they do not support dynamic partial reconfiguration either. On the positive side, this type of FPGAs does not need the transmission of a bitstream on start-up, moreover, this configuration bitstream does not exist, and therefore anti-fuse FPGAs are extremely secure when dealing with intellectual property protection. Similarly, this lack of configuration bitstream at start-up enables these FPGAs to be Live-

At-Power-Up (LAPU), that is, the devices are operational as soon as the system voltage has reached its minimum level. Regarding FPGA robustness and immunity against soft errors, anti-fuse FPGAs also achieve a good qualification. The use of physical permanent interconnections once the FPGA is programmed, makes them highly immune against effects like Single Event Upsets (SEU) [Carmichael'99, XILINX'10].

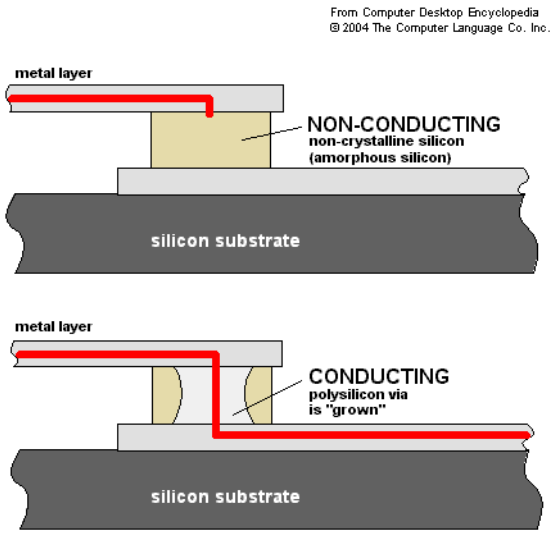


Figure 6: Detail of an Anti-Fuse connection

3.1.2. FLASH FPGAs

This type of FPGAs replaces the anti-fuse connections by programmable FLASH-based switches (Figure 7) [Wang'02] . These switches adopt FLASH technology's characteristics and therefore they are non-volatile but reprogrammable. This way FLASH FPGAs maintain the aforementioned benefits around security and robustness and add them the possibility of being reprogrammed. Unfortunately, although this configuration technology could support partial reconfiguration, the manufacturers do not offer this possibility in their design tools. Consequently, this type of FPGAs has not been used in the designs carried out in this thesis.

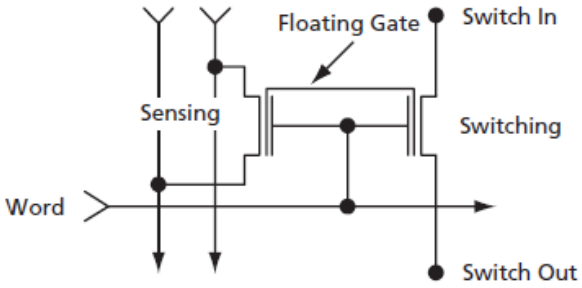


Figure 7: ACTEL FLASH switch

3.1.3. SRAM FPGAs

SRAM-based FPGAs hold their configuration in static memory. The state of both, the programmable LUTs and the switches for signal routing, is controlled by the output of these SRAM memory cells. This way, the configuration of the FPGA is carried out in two steps or layers. On the one hand the configuration data is written and stored into the SRAM configuration memory and on the other hand the output of this memory is the one that directly controls the switches or states of the diverse components in the FPGA. A simple scheme of this type of configuration could be observed in Figure 4 a few pages back. A detail of the way the interconnection switches are controlled is shown in Figure 8 [Lew'04]. Due to the nature of the SRAM memories, this type of FPGAs is volatile, and its configuration is lost whenever the power supply is interrupted. Consequently, the configuration bitstream has to be loaded to the FPGA every time it is switched on. This makes it necessary to include in the system a non-volatile memory in which to store this bitstream, and makes this type of FPGA quite vulnerable against IP hacking. These devices are also quite sensitive to SEUs, attributable to the memory-based configuration architecture. However, there are several characteristics that have turned these FPGAs the most common in the market and the ones that the two main FPGA manufacturers (Xilinx and Altera) focus their development efforts on. On the one hand the circuits used in SRAM FPGAs can be fabricated with standard VLSI (Very-large-scale integration) processes that are nowadays able to implement 28nm transistors. This leads to the achievement of very high density devices compared to other FPGA configuration technologies and therefore to the reduction of FPGA size. On the other hand, this memory-based configuration allows easy in-system reconfiguration and makes it possible the dynamic partial reconfiguration. This characteristic, which will be detailed later in an individual section, permits to change the configuration of part of the FPGA while the rest of it continues working. Updating only certain bits in the configuration memory makes the resources controlled by those bits change their functionality. Being dynamic partial reconfiguration one of the mainstays of the present thesis, from now on all the references related to FPGAs will refer to SRAM FPGAs.

3.2. FPGA manufacturers

The proliferation of FPGAs has brought about an increase in the number of manufacturers of this type of devices. Even though most of them use VHDL or VERILOG as programming language, each manufacturer has developed its own programming tools that embrace from simple code synthesizers, to high level tools for SoC design, software development or implementation of dynamic partial reconfiguration. Moreover, due to the important differences in the fabrication procedures that the abovementioned configuration technologies have, it is usual that each manufacturer just focuses on the development of a certain technology [Donthi'03]. Therefore, it is necessary to analyse the different manufacturers and to choose the one to continue working with.

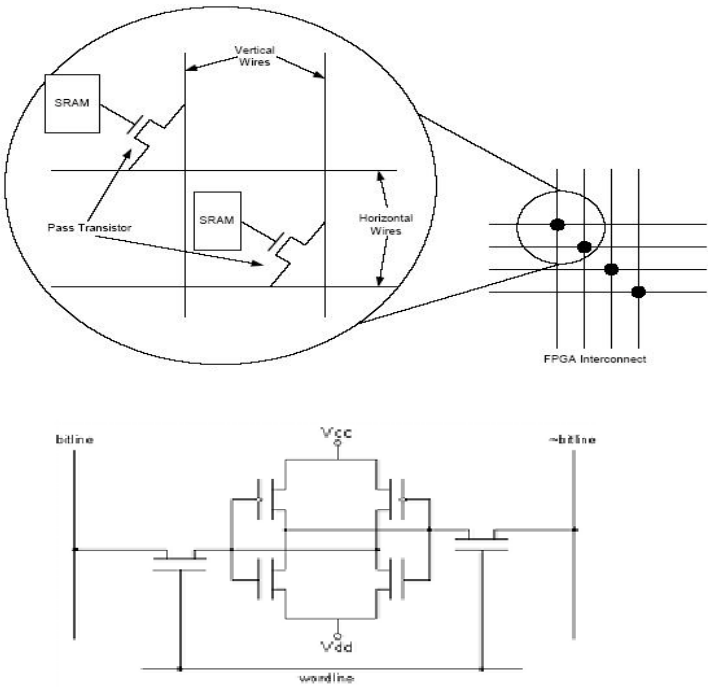


Figure 8: SRAM FPGA interconnection detail and SRAM cell construction

3.2.1. Xilinx

Xilinx [XILINX'12d], together with Altera [ALTERA'12], is one of the FPGA market leaders. Although it started working both with the anti-fuse and the SRAM technologies, since 1996, it is completely focused on the development of this second type of FPGAs. Additionally, Xilinx also develops a wide variety of tools for FPGA design implementation. In regard to dynamic partial reconfiguration, Xilinx is one of the pioneers in its use and hence nearly all the diverse FPGA families they offer support it in one or another way [XILINX'12b].

3.2.2. Altera

As mentioned before, Altera [ALTERA'12] is the other big FPGA manufacturer. This company also works only with SRAM FPGAs but, unlike Xilinx, the possibility of applying dynamic partial reconfiguration to their FPGAs is quite new [ALTERA'10], even if technologically they could have supported it before. Up to this inclusion of dynamic partial reconfiguration, Altera worked with the so-called "Software Programmable Reconfiguration" [ALTERA'08]. This approach, more software-like, suggests maintaining an static hardware with facilities for the software tasks to reuse the different parts of the hardware and hence changing the functionality of the device. In addition, the true on-the-fly reconfiguration of certain resources like PLLs or SerDes components was available.

3.2.3. Actel

Actel [ACTEL'12] is a company that mainly focuses its efforts on the development of non-volatile FPGAs. Therefore, they primarily work with anti-fuse and FLASH FPGAs. Due to the technological

restrictions these FPGAs have and that have been explained before, Actel's FPGAs do not support dynamic partial reconfiguration. However, Actel do have developed the possibility of "on-the-fly" reconfiguring certain parameters of their PLLs [ACTEL'04] what is a great advantage for some king of applications.

3.2.4. Atmel

Atmel [ATMEL] is also focused only in SRAM FPGAs. Besides, Atmel's FPGAs support dynamic partial reconfiguration [ATMEL'02] in the way this thesis understands it, that is, as a means for reusing parts of the FPGA. Unfortunately, Actel has not reached up to now the miniaturization degree of other FPGA manufacturers; hence they do not offer FPGAs with the necessary amount of resources for holding a complex SoC or SDR.

3.2.5. Lattice

Lattice [LATTICE'12] is probably the only FPGA vendor that maintains its position in both the volatile and non-volatile FPGA market. They work with SRAM and FLASH FPGAs. Regarding SRAM FPGAs, they also support dynamic partial reconfiguration [Donthi'03].

3.2.6. Other manufacturers

As stated before, with the spreading of the FPGAs, a proliferation in the FPGA manufacturers has also occurred. Therefore, apart from the aforementioned FPGA manufacturers, there are plenty of smaller companies such as Acroflex, Quicklogic, SiliconBlue Technologies or Achronix also dedicated to FPGA fabrication. These companies do not usually work with general purpose FPGAs but look for they place in the market with the development of FPGAs with exclusive characteristics like low power, small size or very high speed. Taking into account the scope of this research work, these FPGA manufacturers are out of it and hence not analyzed more deeply.

Table 1: Summary of FPGA manufacturers

MANUFACTURER	MAIN FPGA TECNOLOGY	DPR SUPPORT	MAIN CHARACTERISTICS
XILINX	SRAM	YES	Pioneer in dynamic partial reconfiguration
ALTERA	SRAM	PARTIALLY	Newcomer in dynamic partial reconfiguration
ACTEL	FLASH	NO	Focused on non-volatile FPGAs
ATMEL	SRAM	YES	Focused on small FPGAs
LATTICE	SRAM, FLASH	YES	Develop both volatile and non-volatile FPGAs

Considering all the above exposed information (summary in Table 1), there are only two companies that offer FPGAs able to carry out the implementation of Software Defined Radios in the design framework proposed by this thesis: Xilinx and Lattice. The FPGAs from these companies are SRAM-based, support dynamic partial reconfiguration and have the necessary amount of resources to implement designs as complex as SDRs. Considering that Xilinx takes nearly the 50% of the FPGA market, the information and number of works documented with their FPGAs is significantly higher than the ones from Lattice. This, together with the fact that the tools offered by Xilinx for the development of designs with dynamic partial reconfiguration are more elaborated, makes the FPGAs from this company the best candidates for the research work carried out by this thesis. Therefore, from now on, all the works and references made to FPGAs will be referred to Xilinx FPGAs. In this context, it has also to be stated that due to available equipment, all the implementations have been carried out on FPGAs from Xilinx's Virtex 4 family. In any case, this choice is not completely exclusive for the application of the design framework and partitioning methodology proposed by this thesis. That is, the work presented below can be applied with minor changes to Lattice FPGAs or to any other present or future FPGA that supports dynamic partial reconfiguration in the terms Xilinx does.

3.3. FPGA Dynamic Partial Reconfiguration

Partial reconfiguration, by definition, is the process in which part of the FPGA changes its configuration while the rest of it remains unaltered. Under this definition two types of partial reconfiguration can be distinguished: the static and the dynamic one (also known as active). In the static partial reconfiguration, although only part of the configuration of the FPGA is changed, the whole device is deactivated during the process. Only when the new configuration is completely downloaded to the device the FPGA is reactivated. In dynamic partial reconfiguration, in turn, the part of the FPGA that is not reconfigured, commonly known as 'static part', remains active during the reconfiguration process. Moreover, this static part can even be the one in charge of controlling the reconfiguration of the reconfigurable part. This is known as dynamic partial self-reconfiguration.

3.3.1. Operating principle

As has been stated before, partial reconfiguration is a feature that comes from the constructive nature of SRAM FPGAs. In this type of FPGA, both, the states of the logical blocks (LUTs), the configuration of other resources in the FPGA like DSPs or RAMB16, or the state of all the interconnections, is controlled by the output of SRAM memories. The set of all these memories is the FPGA is usually known as 'Configuration memory'. Therefore any change in these memories affects the functionality of the part they control. Furthermore, taking into account that SRAM memories can be re-written on-the-fly and that the final control of the resources is made opening or closing transistor-like components (Figure 9 [Bobda'07]), this reconfiguration can be done without stopping or switching off the device. However, and as will be seen later in this chapter, special care has to be taken when performing a dynamic partial reconfiguration as this technique could potentially lead to the destruction of the FPGA.

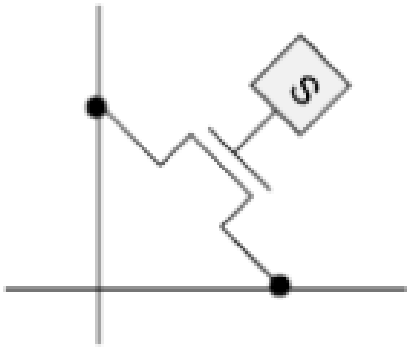


Figure 9: Connection transistor between two wires and SRAM control bit (S)

Analysing dynamic partial reconfiguration from a more operational point of view, it can be simplified just to a download, and subsequent storage, of several bits to the FPGA configuration memory. Usually, a full bitstream with the configuration of the whole FPGA (both static and reconfigurable parts) is downloaded in the first place and later the bitstream holding just the bits to be reconfigured is used. These bitstreams holding only the bits to be changed are commonly known as 'partial bitstreams'. It is precisely the generation of these partial bitstreams the key point for a successful dynamic partial reconfiguration.

Dealing with the download of the partial bitstreams to the FPGA, Xilinx FPGAs offer several access ports to the configuration memory. On the one hand the traditional external configuration ports are available [XILINX'09]. These are the JTAG interface (common name of the IEEE1149.1 standard), the serial configuration interface and the SelectMAP interface. These interfaces allow the configuration or reconfiguration of the FPGA from an external device, whether it be a PC, an onboard processor or another FPGA. Moreover, each interface has its own characteristics and writing speeds (Table 2) what covers a wide range of configuration possibilities. On the other hand, Xilinx's FPGAs are also provided with the Internal Configuration Access Port (ICAP). This port permits the access to the SRAM configuration memory from the inside of the FPGA. That is, a design implemented in the FPGA itself can have access to the configuration memory and hence carry out the dynamic partial reconfiguration. In addition, the 32 bits wide and 100 MHz maximum frequency of the ICAP port offer, together with the SelectMAP interface, the highest bandwidth for FPGA reconfiguration, what makes it the most common interface when dealing with dynamic partial reconfiguration.

There is a final fact that has to be considered in the use of dynamic partial reconfiguration: the storage of the partial bitstreams. Although the use of partial reconfiguration usually leads to a reduction of the FPGA size, it is necessary the inclusion of a memory (in case there was not one in the design) for bitstream storage. This fact is usually not considered as a size overhead since the presence of memories is quite usual in FPGA boards. However, the choice of the memory is quite important as the access speed to this memory will be determinant in the speed the dynamic partial reconfiguration can achieve, as will be seen later.

Table 2: Summary of Virtex 4 FPGA configuration interfaces

INTERFACE	BUS WIDTH	MAXIMUM CLOCK FREQUENCY	MAXIMUM BANDWIDTH
IEEE1149.1 (JTAG)	1 bit	66 MHz	8.25 MBps
SERIAL	1 bit	100 MHz	12.5 MBps
SELECTMAP	8 or 32 bit	100 MHz	400 MBps
ICAP	8 or 32 bit	100 MHz	400 MBps

3.3.2. Types of reconfiguration

Attending to the amount and organization of the resources to be reconfigured, Xilinx defines two types of dynamic partial reconfiguration: difference based reconfiguration and module based reconfiguration [XILINX'04]. Both of them end up in the generation of partial bitstreams, nonetheless, the design flow for achieving them has some differences.

a) Difference based partial reconfiguration

In this type of reconfiguration the number of changes is minimum. These are usually reduced to changes in the equations implemented in the LUTs, changes in the input/output standards (LVTTL, LVCMOS...) or changes in the data stored in the BRAMs. Therefore, and as the name indicates, the generation of the partial bitstreams in this type of reconfiguration is achieved making the direct comparison between the original bitstream and the modified one and extracting the differences. Generally, the design flow starts with a base design that has already been synthesized, mapped and routed. It is opened with a tool like FPGA Editor [XILINX'00b] where the design can be analyzed and where minor changes like the aforementioned ones can be carried out. This secondary design that holds the changes is saved with a new name and finally the differences, and with them the partial bitstreams, are extracted and generated with the Bitgen tool [XILINX'11b].

b) Module based partial reconfiguration

Module based partial reconfiguration is oriented towards providing with reconfiguration capabilities systems designed with the same philosophy: the modular design [XILINX'01]. Modular design allows a team of engineers to independently work on different pieces or "modules" of a design and later merge these modules into one FPGA design. This is an historical design flow in Xilinx tools that has now evolved to what has been called "Design preservation" [XILINX'12a]. This design flow adds to the traditional module based design flow the ability to preserve the implementation results of a module for its use in the next implementation iteration. When dynamic partial reconfiguration is applied to these design flows, the aim is to change the complete functionality of one or several of

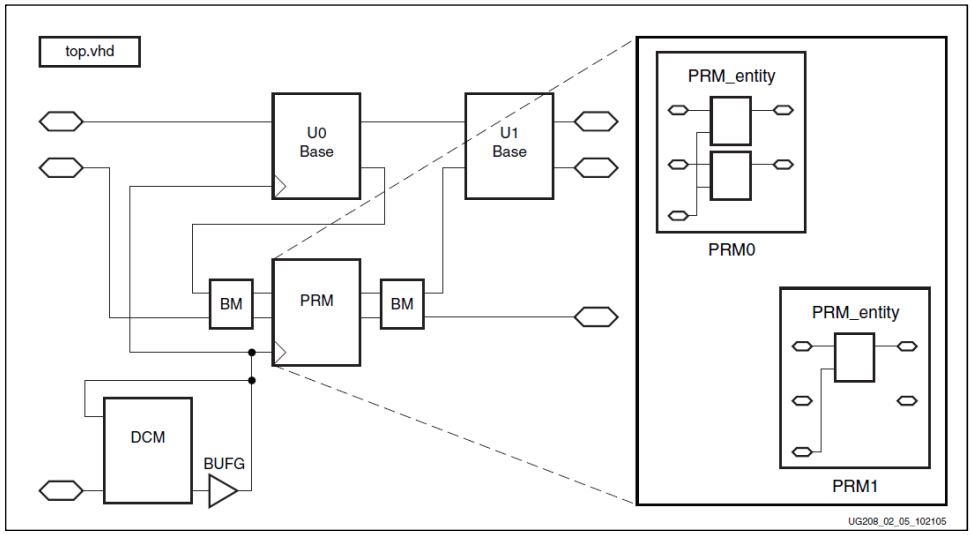


Figure 10: Scheme of module based partial reconfiguration

these modules. This way the reconfiguration is not restricted to minor characteristic changes but can affect nearly any resource in the FPGA including routing resources. Figure 10 gives an idea of how module based partial reconfiguration works. The figure shows a schematic view of the different modules instantiated in the "Top" design. There are several static modules such as U0 Base or U1 Base and a reconfigurable module named PRM ("Partially Reconfigurable Module"). The detail on the right shows the two different implementations the PRM module can have: PRM0 and PRM1. As can be observed the input/output ports have to be maintained but the internal implementation can be completely different. The PlanAhead software [XILINX'11d] is the most useful one when dealing with the partial bitstream generation in this type of reconfiguration. Initially, the software permits the designer to generate the "Top" project. Then, the files that describe the functionality of each module are linked to their instantiation (a single functionality for the static modules and various ones for reconfigurable ones). Finally area restrictions within the FPGA are associated to each of the modules and the partial bitstreams are generated.

3.3.3. Benefits and inconveniences of dynamic partial reconfiguration

Once the philosophy and the operating principles of dynamic partial reconfiguration have been understood, the advantages this technique offers are quite clear:

- A reduction in the necessary area to implement some types of systems (such as SDRs) is achieved due to the reutilization of the resources of the FPGA. Consequently, and even though it becomes necessary to include a memory for storing the partial bitstreams, the overall physical size of the system is reduced.
- The power consumption of the system is also reduced as smaller FPGAs can be used.
- Uninterrupted operation of the system is possible even if a part of it is being reconfigured.
- The implementation of applications based on time-multiplexing is enabled.

Unfortunately this technique also entails some disadvantages that have to be taken into account:

- It is considered a low level programming technique and therefore a deep knowledge on the design flow is required.

- A wrong use of dynamic partial reconfiguration (i.e. generating an internal short-circuit) may potentially lead to the destruction of the FPGA, hence, special care needs to be taken.

- It is a relatively new technique so the available information and experience may be scarce. This novelty, together with the low level nature of this technique leads to a still low efficiency of the design flow, being quite slow and laborious.

- There are certain FPGA resources that must reside in the static region of the design, that is, that do not support partial reconfiguration [XILINX'11c]. They are usually related to global clocks or clock modifying logic (BUFG, PLL, DCM and similar) or to architecture feature components such as BSCAN, STARTUP, etc. Besides, IP restrictions may occur if any of these components are used to implement them (i.e. ChipScope ICON or the MIG controller).

- Some effects that affect system's performance such as reconfiguration time (time in which the reconfiguration is underway and in which the area to be partially reconfigured is not functional) or timing degradation also appear with the use of dynamic partial reconfiguration. These effects need further consideration and consequently they will be explained in detail in Section 6.

3.3.4. Applications for dynamic partial reconfiguration

Even though it is a recent technique, as well as Software Defined Radios [Delahaye'07, He'12], there are in the literature several applications that take advantage of dynamic partial reconfiguration:

- Self-healing architectures [Custodio'07, Wichman'06]. It is possible that certain resources of the FPGA get damaged during operation and hence producing a malfunction of the device. "Self-Healing" architectures are able to detect these damaged resources, read back their configuration and, via dynamic partial reconfiguration, move their functionality to a non-corrupted part of the FPGA. In case the damage is not permanent in the resource (i.e. due to a SEU [XILINX'10]), dynamic partial reconfiguration also permits to recover the resource without altering the rest of the system.

- Dynamically reconfigurable crossbar switches [Young'03]. Crossbar switches are devices able to connect M inputs to N outputs in a matrix way, that is, being able to connect any input to any output. FPGAs are widely used for their implementation due to the input-output resources they provide. The use of dynamic partial reconfiguration makes it possible to maintain several interconnections working while others are being reconfigured.

- Hardware operating systems. Traditional operating systems load, execute and pre-empt software tasks as they are scheduled. Hardware operating systems, in turn, use dynamic partial reconfiguration to load and unload hardware tasks to the FPGA fabric. The position of this hardware modules can be fixed, being always loaded to the same position of the FPGA, or dynamic, in which an allocator controls the free space in the FPGA and places each module in the optimal position [Hong'11a]. R3TOS (Reliable Reconfigurable and Real-time Operating System) [Iturbe'10] is a hardware operating system developed by the University of Edinburgh. In collaboration with them, the implementation of several data coding functions for SDRs over R3TOS and using the design framework and partitioning methodology presented in this research work have been carried out. A deeper look into this work is presented in Section 3 of Chapter 7.

4. RAPID PROTOTYPING TOOLS

Rapid prototyping tools in the field of Information and Communication Technology (ICTs) are programs situated at a higher level than the traditional code compilers/synthesizers. These tools look for easing or improving the code design by adding characteristics like simulations, graphic programming, automatic code generation or co-simulation to the aforementioned code compilers/synthesizers. Under this description very different types of tools can be included: software development tools like Visual Basic [MICROSOFT'12], simulation tools like Matlab/Simulink, system description languages like SystemC [ACCELLERA'12] or FPGA/DSP development tools like System Generator[XILINX'12c], CatapultC [MENTOR'12] or SystemVue [AGILENT'12a].

As stated before, due to the importance FPGAs are gaining in the market, the rapid prototyping tools oriented to this type of devices are also becoming a key element. Moreover, taking into account the difficulty that the design of high performance algorithms in hardware description languages such as VHDL or Verilog entails, these tools become essential. FPGA rapid prototyping tools range from simple code generators that, from the selection of certain parameters, generate VHDL or Verilog code to be subsequently integrated into a bigger project, to highly complex tools. This type of tools includes a graphical environment in which programming is performed interconnecting predefined blocks. These blocks usually represent functions or components that are typically used in signal processing or in the digital logic world, from logic gates to multiplexers, filters or microprocessors. This way, programming is very intuitive and the signals that connect the different blocks represent the real, physical, data paths. Once a design is finished, these tools are able to generate the VHDL code that describes the system, synthesize it, and even, upload it directly to the FPGA. Furthermore, many of these tools work under the Matlab/Simulink environment, so before executing this synthesis and upload, it is possible to carry out a functional simulation, making available all the resources this environment offers, like scopes, customizable data sources or spectrum analyzers, for the results' analysis. This allows the user to iteratively test and adjust in deep the algorithm or system being implemented without having to execute the code synthesis and place and route, usually a time consuming task (Figure 11).

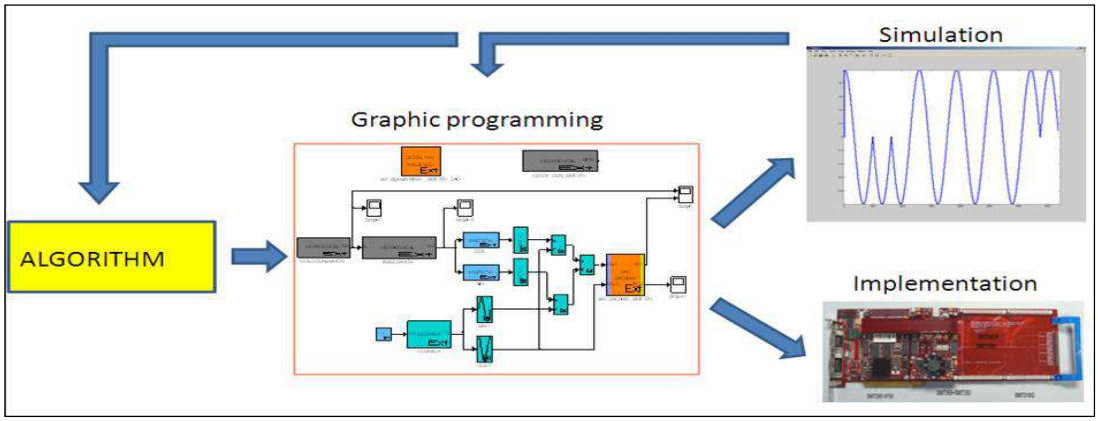


Figure 11: Design flow with rapid prototyping tools

In short, the use of the characteristics that rapid prototyping tools offer, in comparison with the traditional design flow in pure VHDL/Verilog, leads to a reduction in the development time needed for the implementation of a FPGA application. This time reduction has been estimated by some authors in a 10:1 ratio [Haessig'05, Haessig'06].

4.1. FPGA rapid prototyping tool summary

A short survey of the diverse FPGA rapid prototyping tools available on the market is presented below:

4.1.1. Core Generator

Core Generator [XILINX'00a], from Xilinx, is a rapid prototyping tool restricted to code/IP generation. With a catalogue of user-customizable functions as starting point, the designer selects the IP to be generated and sets its corresponding parameters. The available IP functions range in complexity from commonly used functions, such as memories and FIFOs, to system-level building blocks, such as filters and transforms. Subsequently, the tool generates the HDL code that implements the selected function. It remains in hands of the designer the integration of the generated code in the final design.

4.1.2. CatapultC

CatapultC [MENTOR'12] from Mentor Graphics is defined as a high level synthesis tool for FPGAs and ASICs. This is a rapid prototyping tool able to generate synthesizable code for these two types of devices starting from designs carried out in the ANSI C/C++ programming language or in SystemC. When dealing with the final HDL code generation, this tool allows the configuration of several hardware-related characteristics needed by FPGAs or ASICs that cannot be programmed in the aforementioned languages. Area or timing restrictions or IO technologies are some of these characteristics. Once the HDL code is generated, the tool gives the possibility of simulating it. However, for the final implementation of the system, it is necessary the use of an external synthesis software.

4.1.3. AccelDSP

AccelDSP [XILINX'08a] is another rapid prototyping tool from Xilinx (unfortunately discontinued since software version 10.1) that generates HDL code from algorithms implemented in MATLAB language. The design flow in AccelDSP gradually transforms the initial "infinite precision" algorithm into several fixed-point models until the final HDL is generated. During this process it is possible to simulate, evaluate and adjust the precision loss that the algorithm is suffering in each of the steps. As it is presumable, not all the functions available in MATLAB are supported by AccelDSP as they are not implementable in hardware. Nevertheless, most of the functions used in signal data processing are available [XILINX'06]. This is a very interesting tool for those designers with wide experience in the development of algorithms in plain MATLAB.

4.1.4. DSP Builder

DSP Builder [ALTERA'09] from Altera allows the designer to go from a system definition in MATLAB to its integration and simulation in Simulink and, finally, to the VHDL code generation, synthesis and hardware implementation. The tool appears as a toolbox in the Simulink library in which the available functional blocks (filters, FFTs, basic functions...) and the tool-control blocks are present. Both the HDL code and the tcl scripts (for the simulation and implementation of the system) generated by DSP Builder are optimised for being used with Altera's synthesis tool: Quartus II. This tool is also prepared for its integration with the Qsys software, Altera's tool for complex system integration.

4.1.5. SystemVue

SystemVue [AGILENT'12a], developed by Agilent, is defined as an electronic design automation (EDA) environment for electronic system-level (ESL) designs. It permits, via different building blocks, the design, simulation, implementation and test of signal processing algorithms for communication systems. One of the key points of SystemVue is that it is not restricted to the digital part of the systems, but also covers the analog and Radio Frequency (RF) parts of the design. Besides, being Agilent's main activity the design and manufacturing of measurement instruments, SystemVue allows an easy connection with many of these devices enabling system-level verification. Although SystemVues does not work over MATLAB/Simulink it offers a similar graphic environment that eases the design process. Concerning the rapid prototyping characteristics for FPGAs, SystemVue FPGA Architect [AGILENT'12b] is the specific module in charge of it. It adds fixed-point hardware-true simulation models and a VHDL/Verilog hardware implementation path. Unfortunately, as Agilent is not concerned about the final implementation process, so the generated HDL code needs to be ported to an external implementation soflware.

4.1.6. System Generator

System Generator [XILINX'08b] is a rapid prototyping tool, with the appearance of a toolbox in MATLAB/Simulink, developed by Xilinx for designing high-performance DSP systems using FPGAs. The toolbox offers several predefined blocks that represent, from generic, basic functions like adders, logic gates or comparators to complex signal processing functions such as filters, encoders or channel emulators. All these blocks can be simulated in the MATLAB/Simulink environment taking

advantage of all the available resources it offers, like customizable data sources, scopes, spectrum analyzers or the possibility of exporting data to the MATLAB workspace where it can be deeply analyzed. In subsequent stages these blocks are translated into VHDL code, and, harnessing the complete integration of all the Xilinx tools, synthesized, place and routed and, if desired, downloaded to the target FPGA.

Two other characteristics make System Generator an interesting rapid prototyping tool: the possibility of integrating user VHDL code via the so called "Black Boxes" and the feasibility of co-simulation. On the one hand, "Black boxes" are a special type of block that permits the inclusion of pure VHDL code by the designer. In addition, this code can be bitwise simulated together with the rest of the design. This way System Generator eases the implementation of certain operations that could be hardly designed with the predefined blocks. On the other hand, co-simulation is understood as the possibility of mixing the FPGA execution of the implemented design with the analysis resources that the MATLAB/Simulink environment offers. System Generator offers the possibility of generating the input data in MATLAB/Simulink, downloading and processing it in the design implemented in a FPGA and finally, showing the output data in the tool. This way simulation times are reduced considerably and the implemented design is validated in hardware.

4.1.7. Codesimulink/SMT6040

Codesimulink/SMT6040 [SUNDANCE'09] is a software developed by Sundance, similar to System Generator. It is also based on predefined blocks and is executed on MATLAB/Simulink. The main peculiarity this tool has is that is has been designed to target the Xilinx FPGAs that Sundance uses in the development boards they commercialize (which is in fact their main business area). Therefore, besides general purpose function blocks, this tool is also provided with concrete blocks that control certain devices/resources present in the development boards such as clock signal generators, high speed communication buses or analog-to-digital (ADC) and digital-to-analog converters (DAC).

Another interesting characteristic provided by Codesimulink/SMT6040 when dealing with certain development boards is the possibility of carrying out Hardware/Software co-design. Sundance includes in certain boards not only an FPGA but also a DSP, and this tool enables the possibility of determining which functions are implemented in which processor. When the code generation has to be performed, those functions to be executed in the FPGA are translated into VHDL code, while the ones for the DSP are generated in C. In any case, simulations are not affected by this circumstance. Finally, the latest versions of Codesimulink/SMT6040 are oriented towards their use with the Diamond tool [3L'05] from the company 3L. This software is conceived as a design tool for hardware operating systems. It is in charge of the management of the different functions and data communications to be carried out in the system and compiles and downloads each of them to its rightful device. Besides, it allows the combination of code generated by different rapid prototyping tools like Codesimulink/SMT6040 and System Generator into a single design.

4.1.8. Model-Based Design Kit

Commonly known as MBDK [LYRTECH'12], this software kit by Lyrtech offers the necessary tools for combining in a single design the MATLAB/Simulink design environment, System Generator, the Simulink Coder (formerly the Real-Time Workshop, a tool that generates and executes C and C++ code from Simulink diagrams) and the development boards manufactured by this company. Even though it is not a complete rapid prototyping tool, due to the possibility this kit offers of making this tool combination it has been added to the survey. The kit adds to the abovementioned tools the necessary information about Lyrtech's development boards and some code for controlling the communication resources. In this way, operations such as co-simulation or Hardware/Software co-design are simplified.

4.2. Benefits and inconveniences of FPGA rapid prototyping tools

On the basis of the above, the use of FPGA rapid prototyping tools offers the following benefits:

- A reduction on the overall design time is achieved. The automatic code generation and the possibility of simulating and correcting the implementation on early design steps lead to it.
- A deep knowledge on HDL coding or on FPGAs to target complex designs is not necessary. The automatic code generation provided by these tools and the possibility of using several programming languages (MATLAB language, C++, graphical coding...) as starting point allows this.

In contrast these are the main inconveniences that they bring about:

- This automatic generation of the HDL code from the programming blocks entails a loss of control over this code by the designer. Due to the necessity on tool's part of generating a scalable code for each block/function, that must also be compatible and connectable to any other one, the generated code is often oversized and relatively chaotic. In case this HDL code has to be analyzed or modified by hand, this becomes a hard task.
- Similarly, this flexibility needed by the generated code makes it more inefficient than it could be. A dedicated, manual programming of the target algorithms would probably offer a better design in terms of processing speed or resource optimization, but at the expense of a longer design time.

5. CRITICAL REVIEW OF THE PROPOSED SDR DESIGN FRAMEWORK

In the preceding sections the analysis of both FPGA dynamic partial reconfiguration and rapid prototyping tools has been carried out. Traditionally, this two technologies, and even FPGAs themselves, have been used for just prototyping purposes. In other words, the preliminary design tests and implementations to check the feasibility of a certain implementation where realised using these tools and devices but the final implementations were carried out over devices like ASICs that

were manually programmed. The reason for this behaviour lies in the poor performance, in relation to the price, and in terms of size, power consumption and processing capabilities that FPGAs had in their first years and in the lack of reliability of the first rapid prototyping tools. Nevertheless, the evolution FPGAs and rapid prototyping tools have suffered in the last years now makes their use possible as a final implementation framework.

At the best of the author's knowledge, and as has been stated up to now, there is not a tool or design framework nowadays that combines rapid prototyping and FPGA dynamic partial reconfiguration neither for the implementation of Software Defined Radios, nor for the development of any other type of design. The available tools in the market allow dynamic partial reconfiguration or ease algorithm design with rapid prototyping techniques, but do not merge both possibilities. Bearing in mind the advantages and benefits of both technologies that have been presented previously, the proposed design framework seems a promising opportunity for continuing with the spreading of FPGAs as a final device for the implementation of SDRs of any other design.

Unfortunately, this design framework is not immune to the various disadvantages that each of the technologies that take part on it has. The subsequent section will analyze the diverse drawbacks that have been identified.

6. SDR DESIGN FRAMEWORK'S DRAWBACKS

6.1. Reconfiguration time

The main drawback of dynamic partial reconfiguration is the so-called "Reconfiguration time". While the reconfiguration is underway, that is, while the writing procedure that updates certain bits in the configuration memory of the FPGA is ongoing, the resources controlled by those bits are not operative. More precisely, although once a concrete bit is updated the corresponding resource acquires its final functionality, due to the serialized nature of the memory writing procedure, it is not possible to assure a proper operation of the reconfigurable area until the whole partial bitstream is downloaded. Therefore, it is known as "reconfiguration time" the necessary time to write a partial bitstream into the configuration memory. It is a clear drawback and hence it should be minimized.

Since this time is directly related with data writing into a memory, its value depends on the amount of data to be written and in the writing speed. Considering this last point and as already mentioned, between the several access ports to the configuration memory, the ICAP port is the one that offers the highest access bandwidth (theoretical 400 MBps). Therefore it is the most common access interface when seeking small reconfiguration times.

Dealing with the amount of data to be written, several aspects have to be analyzed. The size of the partial bitstreams is directly related to the size and shape of the reconfigurable area. Consequently, the smaller this area is, the smaller the partial bitstream is and hence, the faster the reconfiguration time is. Obviously, a reduction in the size of the reconfigurable area also entails a reduction in the available resources for the intended implementation. Ultimately, it is necessary to reduce the number

of resources/blocks/functions to be reconfigured in order to reduce the reconfiguration time. Fortunately, the nature of SDRs offers an opportunity for achieving this reduction. Many SDR waveforms, being signal processing algorithms, are usually built with similar functions or blocks. Parts like constellation mappers, phase and frequency recovery loops, or forward error correcting blocks are common to most of the waveforms. Consequently, it is possible to maintain them implemented statically and therefore reducing the reconfiguration time. Unfortunately, this concept is not straightforward and can produce undesired effects like an increase in the design size as will be explained in the next subsection 6.2

The last aspect related to the size of the reconfigurable area concerns the percentage of use of the resources in this area. In case the default implementation flow is used in the synthesis tools, the size of the generated partial bitstream is only affected by the size of the reconfigurable area that has been established regardless of the resources used in it. That is, two designs with the same reconfigurable area, one in which the 100% of the resources are used and the other with only a LUT occupied would generate partial bitstreams with the same size. Consequently, it is advisable to adjust the size of the selected reconfigurable area to the size of the greatest design to be implemented in it. In order to reduce the size of the bitstreams with less utilization percentage, it is also possible to enable the 'bitstream compress' characteristic. This characteristic removes the redundant data from the bitstream hence reducing its size. Nevertheless, the unused resources within a reconfigurable area cannot be used by another one, so, even if this characteristic is enabled, it is always recommendable to make this adjustment in the extent possible (unluckily, an excessive adjustment looking for a smaller reconfiguration time it is also inadvisable as will be explained in subsection 6.3.)

6.2. Design size increment

It has been stated that many of the SDR waveforms use common functions that are susceptible to be reused and not reconfigured; however, it is more precise to define them as similar functions. Although these functions share a common name (i.e. interleaver or FFT), each of them has different configuration parameters depending on the waveform. Operating frequencies, filter coefficients, generation polynomials or FFT sizes are some of these changing parameters. In order to establish these functions statically it is necessary to modify them so that these parameters can be changed on the fly. This procedure, that has been named as 'parameterization' and that will be widely analyzed in Chapter 4, generally produces an increase in the number of the resources used by the function in scope. Therefore, the amount of resources used by the final design may increase when trying to reduce the reconfiguration time so it is a factor to be evaluated.

6.3. Timing degradation

The design flow of dynamic partial reconfiguration forces the synthesis tools to place all the resources and routes within the defined reconfigurable area. This is a limitation that hinders the search for the optimal routing of the clock signals, hence reducing the maximum clock frequency that the designs with dynamic partial reconfiguration can support. Xilinx estimates this degradation in a

10% [XILINX'11c]. Nevertheless, this is an average estimate that changes depending on the designs. An excessive occupation of the reconfigurable area makes the optimal routing harder and worsens this degradation. Moreover, occupations above an 80% usually lead to the impossibility of implementing the design.

Besides, the more reconfigurable areas there are and the more nets that go in and out of these reconfigurable areas, the worse the timing degradation is. In-out ports in a reconfigurable area must be fixed in order to be reused by the different partial designs. This fact is also a limitation for the synthesis tools and therefore this number of nets also affects the design's timing.

Finally, rapid prototyping tools, as a drawback of the flexibility and facilities that the automatic code generation offers, usually achieve worse timing characteristic than a manual coding would do. The necessity of generating HDL code for each block able to be merged with a wide range of blocks and able to support the modification of several parameters generates this worsening.

In view of the above, the maximum clock frequency that a certain design is able to support is also a factor to be under control.

6.4. Flip-Flop initialization

Finally, and placed in a more functional level, an issue related with the initialization of the Flip-Flops when placed in a reconfigurable area has to be solved. In static implementations one of the last commands present in the bitstream and that has to be executed by the FPGA is GRESTORE. This command initializes all the Flip-Flops in the FPGA to the default values programmed in the corresponding HDL language. However, this command is not present in the partial bitstreams as it affects the whole FPGA and its execution could corrupt the static area. Therefore, after a partial bitstream has been downloaded to the FPGA the Flip-Flops present in the new design are not initialized and maintain their previous value. This circumstance may not be acceptable in some designs (i.e. if the coefficients of a certain filter are stored using Flip-Flops), so it has to be resolved. The designed solution is out of the scope of the design methodology itself but is part of the achievements of the present work so it will be presented in Section 3.4.3 of Chapter 7. This chapter presents a real case example of an implementation that requires this type of initialization. This circumstance motivated the development of the Flip-Flop initialization procedure.

7. DESIGN METHODOLOGIES

The use of design methodologies that try to optimize some of these issues, determining the best way of undertaking the implementation of a certain design, is widespread. The next subsection lists several design methodologies present in the literature that face this optimization of the implementation from different approaches.

Berthelot et al. [Berthelot'08] propose a complete design methodology for automatic design generation in heterogeneous architectures. These architectures are made up of several "processing

elements" such as general purpose processors, DSPs or FPGAs. These last ones even supporting dynamic partial reconfiguration. This methodology starts with the generation of a model for both, the target architecture and the algorithm/system to be implemented. Subsequently, the methodology determines the optimum mapping of tasks into the processing elements. The criterion for choosing the best placement looks for minimizing the reconfiguration cost in terms of reconfiguration time. Taking advantage, if possible, of the sequential execution of certain tasks, the methodology attempts to parallelize the reconfiguration of some tasks with the execution of others with no data dependencies. This technique is known as *Configuration Prefetching*.

Göhringer et al. [Göhringer'10] in turn, present a design methodology that tries to identify the optimum architecture for a certain application. Starting with a description of this application made with a high level language like C/C++, the methodology first performs a software/software partitioning of the application into several modules or tasks. Each of these modules represents a processing element in the final implementation. In this case, the criterion for making this partitioning looks for minimizing a closeness function that evaluates several factors of the system such as the processors' workload, the necessary memory resources, the timing requirement for a real-time execution or the size of the FPGAs in case they are needed. A second stem in the methodology carries out a hardware/software partitioning deciding which modules from the first step should be implemented in software and which in hardware. This decision is performed looking for the maximum local efficiency of each of the processing elements.

Another type of optimization is the one proposed by Athalye and Hong [Athalye'05]. Their work tries to determine the best way of mapping the several tasks in an application into the different, predefined, reconfigurable areas available in a Xilinx FPGA with use dynamic partial reconfiguration. They use a communication cost based heuristic criterion that analyzes the number of lines that cross from a static area to a reconfigurable one and vice versa. They assume that the higher this factor is, the worse is the system's performance since more routing issues appear.

Returning to the hardware/software co-design approach, Noguera and Badia [Noguera'01] present a partitioning algorithm for dynamically reconfigurable architectures. This algorithm is divided into three stages: application stage, static stage and dynamic stage. The first stage is in charge of modelling the application to be implemented with a set of interrelated processing objects which communicate among them using events. The static stage converts the information from the application stage into independent tasks and, after a execution time, memory usage and area estimation, decides which of these tasks will be executed in hardware and which in software. Finally, the dynamic stage schedules the execution of the tasks in the different processing elements. Focusing on the hardware/software partitioning, the algorithm tries to minimize a cost function that measures the execution time of the whole system. On this purpose, the algorithm iteratively evaluates the cost function against the different possible partitions. In addition, the algorithm takes into account a configuration prefetching mechanism for reconfiguration latency minimization.

Alaus et al. [Alaus'09] focus their efforts on the development of a design methodology for Software Defined Radios. They work in the concept of parameterization of SDRs, defined as the identification

of all the common aspects of the mobile communication modes and standards to be implemented. These common aspects become one common processing procedure installed in the device and can be executed by a single "call". Thereby a gain of both size and time with regards to the amount of code to be downloaded or read in order to modify the radio behaviour is achieved. The authors face this parameterization with the development of two techniques: the Common Functions technique and the Common Operators technique. Each of the techniques deals with a different granularity of the parameterization. The Common Functions, as its names suggests, looks for similarities at "Functionality" level, with functions like mapping, filtering, synchronization and so on. In turn, the Common Operators technique looks for common elements based on structural aspects. FFT, Cordic or LUTs are some of these operators. This last technique claims to be independent of the standards by finding the smallest set of highest-level operators which are used by the maximum number of functions. In this context, the proposed design methodology completes these two techniques with several steps that cover the rest of the design flow. This methodology and particularly the Common Functions and Common Operators techniques complement perfectly with our proposed design framework and partitioning methodology, hence they have been ported and used as a part of our design methodology. Further information about this point is discussed in Chapter 4.

Bearing in mind the still slow and laborious process of using FPGA dynamic partial reconfiguration in a design, authors like Cancare [Cancare'07] propose a methodology for automating this design flow. The proposed methodology relies on Simulink as high-level development framework for reconfigurable systems and on an enhanced version of the Caronte flow [Donato'05] as low-level implementation generator. This low-level implementation phase embraces and automates all the necessary steps for the generation of partial bitstreams from a VHDL description of a system.

Finally, the reconfiguration time being one of the main drawbacks of the dynamic partial reconfiguration, there are in the literature various works that try to resolve this issue speeding up the access to the configuration memory. Although they are not design methodologies themselves, it is worth considering them.

Liu et al. [Liu'09] explore the design space of ICAP architectures and propose and evaluate five different designs in which different storage sources and communication interfaces for the partial bitstreams are tested. Taking the base designs for ICAP provided by Xilinx as a starting point (reconfiguration speeds below 1 MB/s) they achieve a maximum ICAP speed of 371 MB/s, close to the theoretical maximum speed of 400 MB/s in a Virtex 4 architecture.

Similar approaches are presented in [Bonamy'12] and [Hansen]. Bonamy et al. present an ultra-fast power-aware reconfiguration controller (UPaRC) that boosts the reconfiguration throughput up to 1.433 GB/s. This huge improvement in relation to the theoretical maximum is achieved breaking the limitation self-imposed by Xilinx to maximum clock rate applicable to the ICAP port. In a similar way, Hansen et al. implement an enhanced ICAP hard macro able of raising the reconfiguration speed up to 2200 MB/s. Both approaches are based on Virtex 5 devices.

Finally, and looking for spreading the possibility of using self-reconfiguration to devices that initially do not support it, Bayar and Yurdakul [Bayar'08] have developed the PCAP: Parallel Configuration Access Port for Spartan-III FPGAs. These FPGAs do not implement an internal ICAP, hence they do not support dynamic self-reconfiguration and they require an external device to be reconfigured. The presented solution implements a loopback between the pins of the SelectMAP interface and some user-IO pins in the FPGA. This way the PCAP core, implemented in the FPGA, controls the SelectMAP interface and self-reconfiguration is enabled.

8. CRITICAL REVIEW OF THE STATE OF THE ART OF DESIGN METHODOLOGIES

All the aforementioned methodologies improve the performance of the SDR in the terms we intend to. However, many of those related with FPGA dynamic partial reconfiguration start their work from a pre-partitioned design. They need to know beforehand which functions or parts of the design will be implemented in the static part of the FPGA, and which ones in the reconfigurable one. Furthermore, the final physical distribution of the FPGA, in terms of number and size of reconfigurable areas, need to be also predefined. Usually, all this work is done manually by the designer, based on his knowledge of the system and of the application.

However, at the best of author's knowledge, this handmade partitioning can also be optimized, especially as the designs' complexity grows up. Therefore, the second goal of this thesis is the design, validation and automation of a design methodology that looks for the optimal partitioning in terms of static or reconfigurable implementation of the functions present in a Software Defined Radio.

9. METHODOLOGY OBJECTIVES AND INITIAL HYPOTHESIS

In view of the above, the proposed design methodology aims to fulfil the following objectives:

- Optimize the implementation of Software Defined Radios in terms of reconfiguration time, amount of resources needed and maximum clocking frequency.
- Reduce the design time and simplify the design flow for implementing SDRs using FPGA dynamic partial reconfiguration and rapid prototyping tools

The initial hypothesis that motivates the development of the methodology is defined as follows:

- The use of a design methodology that determines which parts/functions of a Software Defined Radio has to be implemented statically and which in a reconfigurable way, when using a design framework made up by FPGA dynamic partial reconfiguration and rapid prototyping tools allows for the following. On the one hand, reduce the design time if compared with a manual implementation. On the other hand the securing of an optimal implementation. It is meant by optimal the implementation with less value of a cost function

generated with the reconfiguration time, the amount of resources needed and the maximum clock frequency.

10. SUMMARY

The present chapter has made a deep analysis on the state of the art of the three main technologies that take part in the development of this thesis: Software Defined Radios, FPGA dynamic partial reconfiguration and rapid prototyping tools. The benefits and inconveniences of such a design framework have been evaluated and the design methodologies present in the literature that try to optimize those drawbacks have been examined. Ultimately, a critical review of these methodologies has been carried out and a novel design methodology that addresses static/reconfigurable partitioning has been identified as a promising technique for optimising the implementation of SDRs with FPGA dynamic partial reconfiguration and rapid prototyping tools.

Chapter 3

Design Methodology

3. Design Methodology

1. INTRODUCTION

In this chapter the basics of the proposed design methodology will be presented and explained. Based on the information analyzed in the foregoing state of the art, firstly the degree of freedom over which the methodology will act is decided and explained. Subsequently, the several steps that make up the design flow of the methodology will be exposed. Finally, the basics of an automation procedure for the methodology that aims to ease the design flow and hence reduce the design time will be presented.

2. DEGREE OF FREEDOM: RECONFIGURATION GRANULARITY

Taking into account the drawbacks of the proposed SDR design framework and the compilation of the existing design methodologies exposed in the state of the art, there are several parameters or characteristics over which a design methodology can act in order to optimize a design. Among them, and being, at the best of author's knowledge, a novel approach, the partial reconfiguration granularity is the degree of freedom that the current design methodology modifies in order to optimize the SDR design. Partial reconfiguration granularity is defined as the design level where the dynamic partial reconfiguration takes place.

By way of example, in order to change the functionality between two SDR waveforms it is possible, on the one hand, to erase and reconfigure the whole waveform. This is considered a coarse grain reconfiguration and has several advantages (like minimum resource utilization) and disadvantages (very high reconfiguration time). On the other hand, another approach for carrying out the same objective consists of just reconfiguring the concrete functions that are not common to both waveforms, maintaining the rest of the design static. This would be a fine grain reconfiguration and generally entails a reduction in the reconfiguration time at the expense of an increase of the resources needed by the design and a degradation of the supported maximum clock frequency. Between these two reconfiguration granularities there are several other partitioning possibilities that affect the design factors under our scope. Due to the non-linearity of the relation between these factors and the reconfiguration granularity, there are certain partitions that offer an overall better design performance. Figure 12 graphically shows this concept. It should be noted that this is a hypothetical representation of the expected evolution of these factors. Its verification is precisely one this research work's tasks.

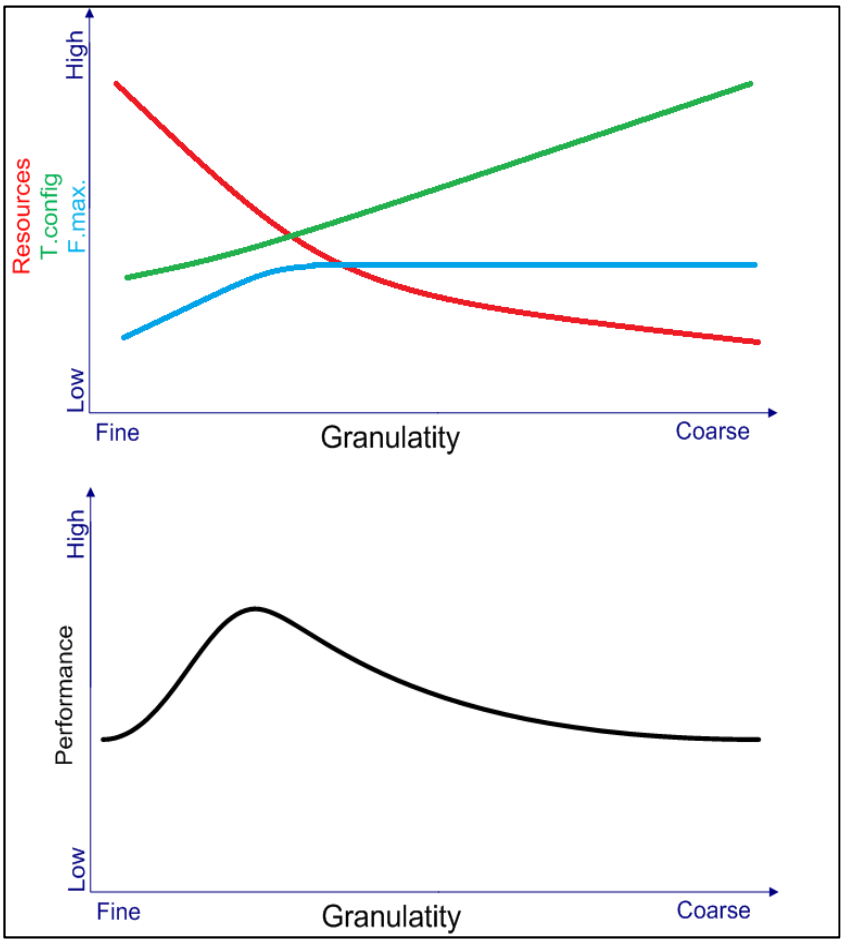


Figure 12: System performance vs. Reconfiguration granularity

Consequently, the aim of the proposed design methodology is to quantitatively evaluate the way the most important system factors evolve in relation with the reconfiguration granularity and determine the optimum implementation.

3. METHODOLOGY DESIGN FLOW

Hereafter, the methodology design flow will be introduced. A detailed description of the several design steps that make up the methodology will be presented. However, the two most important features, namely, the establishment of common parts and its surrounding proceedings, and the mathematical definition of the cost function will be individually analyzed in Chapter 4 and Chapter 5 respectively.

Figure 13 shows the basic design flow of the proposed design methodology.

As suggested above, the design methodology is broadly an iterative procedure in which several iterations at different levels are necessary to finally obtain the optimal implementation.

Firstly, a theoretical approach to these steps will be presented. This theoretical approach aims to individually cover all the concepts involved in each of the steps and represents a manual application

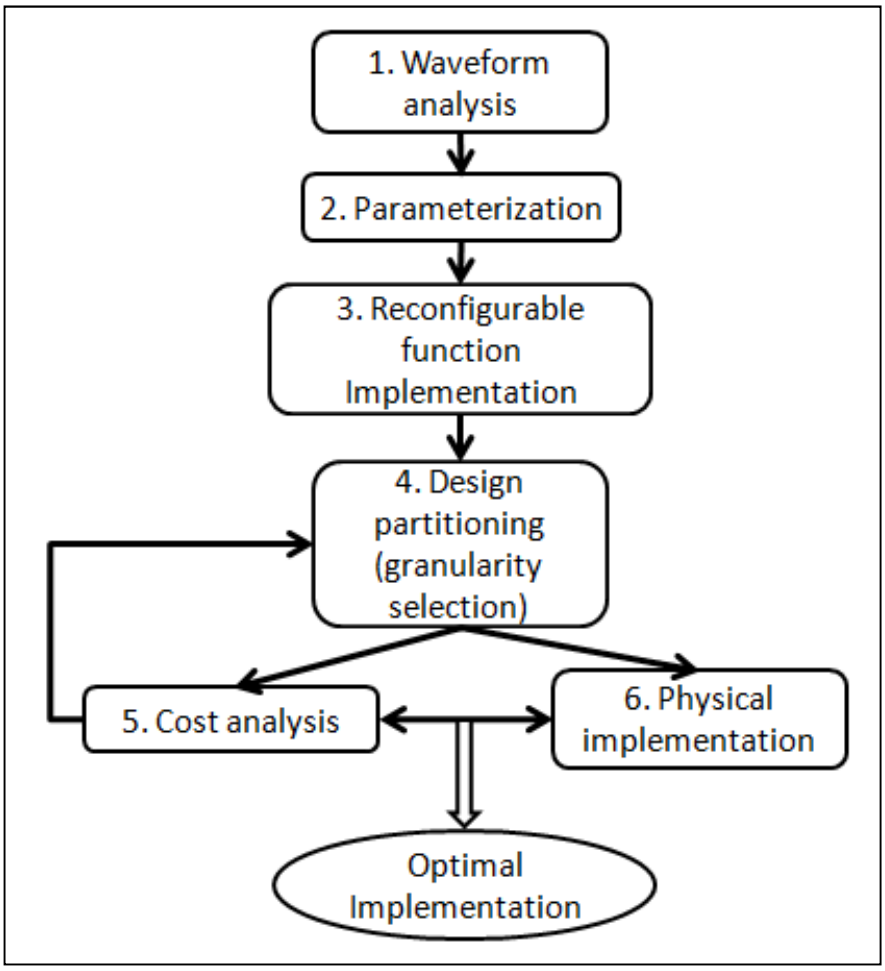


Figure 13: Partitioning methodology design flow

of the methodology. That is, a brute force approach to the methodology that covers all the design possibilities at the expense of incrementing the design time. Bearing in mind that one of the objectives of the design methodology is to simplify the design flow and reduce the design time, these steps can be later simplified and automated as will be seen in Section 4.

3.1. Analysis of the waveforms to be implemented: "Common functions/common operators" technique

The first step and starting point of the design methodology covers the analysis of the waveforms or communication systems to be implemented. This step, essential and common to nearly all the design methodologies and design processes whatever the field of application is, aims to seek the basic characteristics and factors of the target SDR.

Taking into account the degree of freedom that this methodology uses in order to optimize the design and being the existence of common parts in the waveforms to be implemented the keystone of this methodology, the main objective of this step is to gather information about the possible candidates to be reused. On this purpose the design methodology uses a modified version of the general

"Common functions/common operators" technique [Alaus'09]. Figure 14, extracted from the original presentation of this technique, gives an overview of the way the commonalities are chosen.

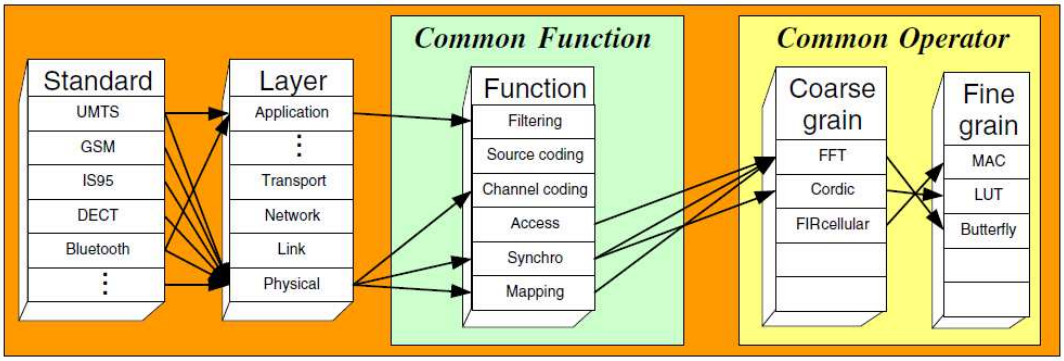


Figure 14: Common functions/common operators

The characteristics of both the original technique and the modified version are widely addressed in next chapter. Nevertheless, after the application of the technique a list, named "Function commonality list", of the design parts, functions or functional blocks susceptible to be reused is generated. This list contains all the parts in the design and, on the one hand, labels them as "reconfigurable" if they are not common to any of the waveforms and hence have to be implemented in a reconfigurable area. On the other hand, if the function or part of the design can be shared between the different waveforms, it is labelled as "common" and will be methodology's work to decide whether it has to be implemented statically or as a part of a reconfigurable area. It has to be stated that all this analysis is carried out from a high-level point of view; that is, without performing a deep analysis of the involved functions. The "function commonality list" is then completed with the concrete functional characteristics of each function (e.g. input-output interfaces, number of coefficients, generation polynomials... and so on) needed by subsequent design steps.

Finally, and also as a part of the information compilation procedure but in another design level, this step collects certain information about the target FPGA (if known) and about the performance requirements of the final application. Basic information like target clock frequency, maximum available resources or maximum assumable reconfiguration time are passed to the next steps so that the methodology can act consequently. In case any of the parameters is not determined, it is left in hands of the methodology the election of a target value. This information is collected in the so called "Requirement sheet".

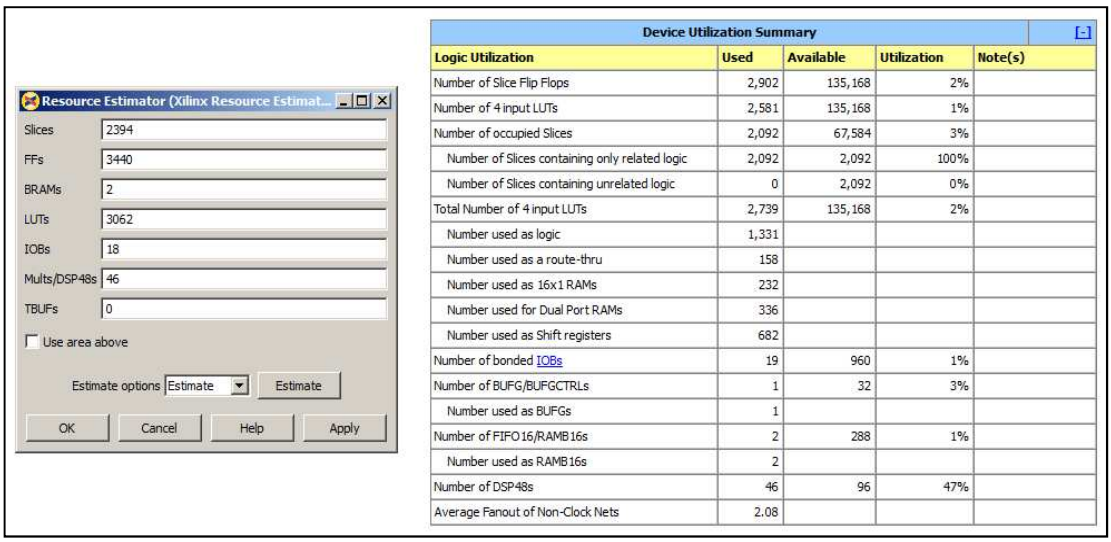
3.2. Establishment of common parts: Parameterization

Once the analysis of the waveforms to be implemented has been carried out and the list of functions susceptible to be reused is generated, it is necessary to confirm and definitely establish the common parts. As already presented in Section 6.2 of the state of the art, many functions present in SDR systems have the same name; however, a deeper look at them reveals that they are not equal but similar (different generation polynomials, different filter coefficients... etc.) and hence need some

modifications in order to be reused. Bearing in mind that the classification carried out in the first steps has only considered function names for its generation, this in-depth analysis of the candidate functions to be reused is necessary.

The design procedure that carries out the necessary modifications in these functions to make to make them reusable has been named "Parameterization". This name responds to the fact that these functions have to be able to change some of their parameters on-the-fly, that is, they have to be parameterizable. Chapter 4 addresses "Parameterization" in detail and presents several parameterized functions that have been developed during this research work.

Definitively, the aim of this step is to analyze the functions labelled as "common" in the "function commonality list" and decide whether they are ready to be reused directly or they need to be parameterized. In this second case, the design of both, the straightforward and parameterized versions of each function is carried out over rapid prototyping tools and the necessary information, in terms of reconfiguration time and amount of FPGA resources used, is generated and added to the "Function commonality list". This data can be generated in various ways depending on the required accuracy. On the one hand the resource estimators available in rapid prototyping tools can be used. These estimators offer a rough estimation of the resources used by a certain function or design prior to any code generation or synthesis at the expense of a lower accuracy. On the other hand, the complete design flow can be executed. It is a time consuming task but the performance of the automatic code generation, synthesis and mapping (place and routing does not modify the resources needed) delivers exact results. Figure 15 shows a comparison between the estimated and exact results obtained for the IFFT function of the WiMAX standard used in the multi-standard modulator presented in Chapter 6. The accuracy of the estimation varies for each design, as the rough resources estimated for each function can be optimized by the synthesis tools. These tools are able to apply certain resource optimisation techniques such as SLICE packaging, equivalent register removal or unused logic trimming that lead to reduction in the overall resources used. However,



Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	2,902	135,168	2%	
Number of 4 input LUTs	2,581	135,168	1%	
Number of occupied Slices	2,092	67,584	3%	
Number of Slices containing only related logic	2,092	2,092	100%	
Number of Slices containing unrelated logic	0	2,092	0%	
Total Number of 4 input LUTs	2,739	135,168	2%	
Number used as logic	1,331			
Number used as a route-thru	158			
Number used as 16x1 RAMs	232			
Number used for Dual Port RAMs	336			
Number used as Shift registers	682			
Number of bonded IOBs	19	960	1%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of FIFO16/RAMB16s	2	288	1%	
Number used as RAMB16s	2			
Number of DSP48s	46	96	47%	
Average Fanout of Non-Clock Nets	2.08			

Figure 15: Resource estimation vs. Implementation results

during the implementations carried out in this research work a maximum error of 15% has been detected between the estimated and the real values. This is an acceptable accuracy and consequently the estimated values can be used in the design methodology.

It is noticeable that the maximum clock frequency is not taken into account in this analysis although it is one of the important factors in the system. This factor cannot be estimated and is highly dependent on the final physical implementation; consequently, it will be addressed later in the design methodology. Besides, the reconfiguration time is also a factor that cannot be directly obtained neither from the estimators, nor from the implementation results. Its real value can only be measured on the final system. However, it can be estimated from the resource utilization value. Chapter 5 will present the estimation function used on this purpose.

Obviously, both in this design step and in the next one, in which the implementation of the functions labelled as "reconfigurable" is carried out, the full potential that rapid prototyping tools offer is exploited. Functional simulations and performance tests are carried out in order to ensure proper operation of designs.

3.3. Reconfigurable function implementation

As mentioned above, once the "common" functions have been implemented, both in their straightforward and parameterized versions, and their relevant information compiled, the same procedure has to be carried out for the "reconfigurable" functions. In this case, obviously, there is only one version for each function as these functions do not need to be reused. Nevertheless, all the other aspects commented in previous step (data generation procedures, use of rapid prototyping tools and so on.) are applicable here.

To sum up, once the previous three steps have been completed, the SDR design is in the following state (Figure 16):

- A list of the different functions that make up the whole SDR has been generated (*Function commonality list*). Each function has been classified as "reconfigurable" if it is not common to any other waveform in the SDR or “common" if the function appears in more than one waveform.

- A requirement sheet has been filled out with the basic characteristic that the final SDR has to meet (*Requirement sheet*).

- All the functions involved in the SDR have been individually designed with rapid prototyping tools. There is a straightforward implementation for the "reconfigurable" functions and a straightforward and a parameterized version for the "common" functions.

- The *Function commonality list* is completed with the resource utilization and reconfiguration time for each of these functions.

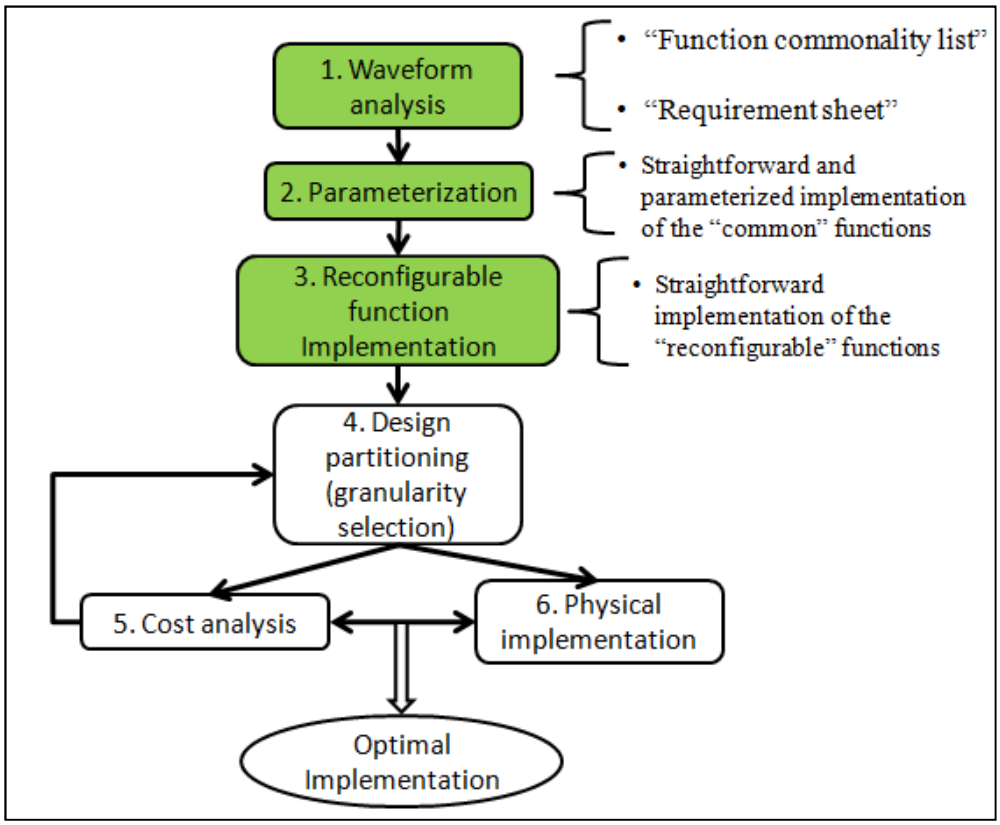


Figure 16: Design flow evolution and generated data

3.4. Design partitioning (granularity selection)

With all this information, the optimization loop that will determine the optimal static/reconfigurable design partition starts. This loop involves the partitioning step, the design cost analysis performed in the next step, and optionally, the final implementation presented in subsection 3.6. That is, steps 4, 5 and 6 on Figure 16.

This step's main objective is to generate all the possible partitions that can be obtained from the combination of the two implementation possibilities (static or reconfigurable) of each "common" function. Remembering the explanation made in Section 2 about the degree of freedom of the methodology, between the whole reconfiguration of the waveform and the reconfiguration of just the functions that change from waveform to waveform, there are several reconfiguration granularities that must be analyzed. This step's mission is precisely the generation of the designs that stem from these different granularities. It should be noted that facing the design of complex SDRs, the number of possible partitions can grow exponentially. In that case it is not possible to generate all the designs and other solutions are needed. Some of these solutions will be explained later in this section.

A base partition situated in one of the granularity ends is generated as a start point for this design step. This initial partition looks for a minimum reconfiguration time so it implements all the "common" functions statically, that is, with their parameterized version. From this point on, the rest of the partitions are generated. Individually, each "common" function is passed from its static to its

reconfigurable implementation until the coarse grain reconfiguration is reached. This is the other reconfiguration granularity end in which the whole waveforms are reconfigurable. Although the words "generate" or "implement" have been used throughout this paragraph, it has to be stated that up to now all the obtained partitions are just theoretical. Each partition is just composed of a list of the status (static or reconfigurable) of the different functions.

Once each partition is generated, their optimality has to be evaluated. This objective is ultimately carried out in the next step. However, the methodology offers here two alternatives: on the one hand each partition's cost can be directly evaluated based on the already obtained factors; that is, based on the factors estimated in steps 3.2 and 3.3. This option is straightforward and does not need further actions prior to executing step 3.5. Consequently, a very fast evaluation is obtained but at the expense of two main drawbacks: the available data has been obtained via estimation, hence it may suffer deviations, and the maximum clock frequency is not available. This factor is dependent on the final physical implementation and therefore it is not available at this point. On the other hand, the second design option that the methodology offers considers the real physical implementation (step 3.6) of each partition prior to the evaluation of its performance. This way, the available data for the evaluation is exact and the maximum clock frequency has already been obtained. Unfortunately, this implementation process (made up of the HDL code generation and the synthesis, map, place and route processes executed in the EDA tools) is a time-consuming task that can considerably slow down the design process if it is executed many times.

In order to avoid this issue the methodology foresees two actions. On the one hand a mixture of both the estimated and the real design evaluation procedures can be used. Initially a fast estimation can be performed in order to discard the worst designs. Once a reasonable number of partitions are left, the physical implementation can be executed in order to obtain the exact design factors. On the other hand, the total number of partitions can also be reduced beforehand. The quantification of the number of partitions that can be considered "reasonable" is dependent on the available design time. However, from the experience gained in this research work, as a guideline, the complete implementation of more than 10 designs is inadvisable.

Regarding the aforementioned reduction of the number of partitions, in the literature there are several optimization algorithms that can face up with this issue and reduce the number of tests. The Simulated Annealing algorithm (SA) [Kirkpatrick'83], the Stochastic Tunnelling (STUN) [Wenzel'99], or evolutionary algorithms are some of them. Unfortunately, their inclusion in the design methodology is out of the scope of this work, which uses a brute force algorithm to tests all the possibilities. As will be explained later within the future work section, the inclusion of this kind of algorithms in the design methodology would be really interesting.

3.5. Design cost analysis via cost function

This step is in charge of quantitatively evaluating the optimality of each of the partitions generated in the previous step. On this purpose, a cost function that computes a weighted sum of the most important factors has been designed. Chapter 5 will analyze all the aspects related with this cost

function and its application, however, as an introduction, a conceptual, basic version of the cost function can be seen in the equation (1) below:

$$Cost = Resources + T_{RECONF.} + T_{CLK\_MIN.} \quad (1)$$

As can be seen, the function takes into account the three factors that, due to their relevance and variability, have been identified as important in Section 6 of Chapter 2: the design size (measured via the FPGA resource occupation), the reconfiguration time and the maximum clock frequency. Please note that the minimum period, that is, the inverse of the maximum clock frequency (2), is used in the third term of the cost function. The maximum clock frequency is the usual way of representing this factor (e.g. in the reports generated by synthesis tools), moreover, it is the way that is used along this document. However, the use of the minimum clock period suits better in the cost function, as this way the three terms that are evaluated need to be reduced in order to obtain a lower cost. Therefore, this small transformation has been carried out.

$$T_{CLK\_MIN.} = \frac{1}{F_{CLK\_MAX.}} \quad (2)$$

Furthermore, harnessing the computations carried out by this cost function, this step checks that each design fulfils the performance requirements collected in step 3.1. Even if a design obtains the lowest cost, in case any of the design factors exceed the maximum established limits it has to be discarded or, at least, re-implemented.

Finally, relative to the afore-mentioned possibility of estimating the cost of a partition prior to its final implementation, another consideration has to be taken into account. The same cost function is used in both, the real and the estimated case; however, bearing in mind that certain factors are missing in the estimated case, a procedure for computing them is needed. Therefore, in addition to the general cost function, several estimation functions are available within this step in order to calculate these missing factors. These functions will also be presented in Chapter 5.

3.6. Physical implementation

Once an optimal design partition has been elected, or as part of the aforementioned optimization loop made up of steps 4, 5 and 6 of Figure 13, the final step is the physical, real, implementation of this design. This step carries out all the necessary procedures in order to obtain the base bitstream that holds the static parts and the partial bitstreams in order to perform the functionality changes. These procedures combine two design paradigms. On the one hand, bearing in mind that rapid prototyping tools do not support dynamic partial reconfiguration in their standard design flow, some actions and checks have to be performed so as to make the HDL code generated by these tools compatible with it. On the other hand this step covers the standard dynamic partial reconfiguration design flow [XILINX'11c, XILINX'11d].

Addressing the compatibility of rapid prototyping tools with dynamic partial reconfiguration two main issues have to be resolved. Both of them are related with the automatic code generation carried out by rapid prototyping tools, as it entails a lack of control over this code:

- Partial design homogenisation: Xilinx's standard dynamic partial reconfiguration design flow is based in the Plan Ahead software as will be seen later in this section. This design flow uses a "Top" design in which all parts of the design are initially instantiated in the form of black boxes. That is, only the high level definitions of the functions/parts of the design are initially instantiated. Dealing with the reconfigurable parts, a unique black box per dynamically reconfigurable area (PRA) is instantiated. It is later in the design flow when as many dynamically reconfigurable modules (PRM) as different functionalities for this reconfigurable area are linked to it. That is, a single PRA is present in the "Top" design but, there are several PRMs to describe its different functionalities. Consequently, it is compulsory that every PRM and its corresponding PRA has the same name and same connections. Bearing in mind the possible different naming that can be used by rapid prototyping tools it is necessary to create a wrapper file for each PRM generated with these tools. In this wrapper file the original design is instantiated and the entity name and signal towards the external world are homogenized. Figure 17 shows this process.

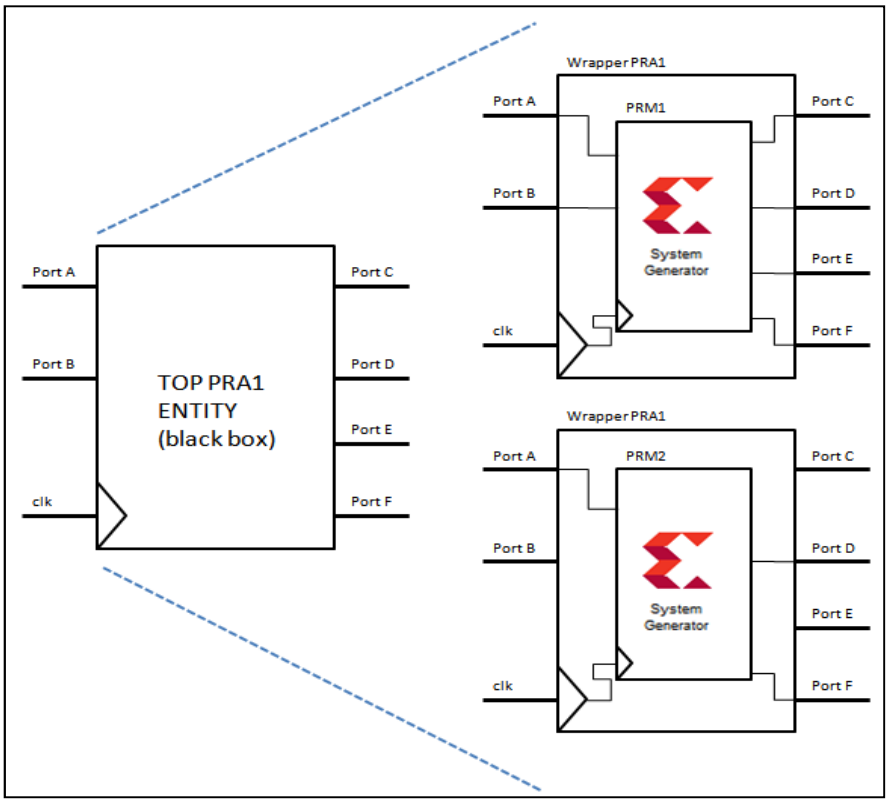


Figure 17: PRM homogenization procedure detail

- Non-reconfigurable resource detection: As has been already explained in Section 3.3.3 of Chapter 2, due to FPGA constructive limitations there are several resources that cannot be reconfigured. Taking into account that rapid prototyping tools are not aware of this limitation and that they usually do not allow such a low-level design control, it is necessary to examine the generated code looking for the use of these resources. In case they are used, some modifications have to be carried out in order to remove or replace them. If they are required by the design, the generated code has to be modified so as to move the non-reconfigurable resource to the static part while preserving its connection. This way this resource remains static but the reconfigurable design makes use of it.

The Plan Ahead software is the keystone of Xilinx's dynamic partial reconfiguration design flow. This software is in charge of collecting the source files of all the parts involved in a particular reconfigurable design and generating the necessary files for carrying out the partial reconfiguration. This design flow is standardized and perfectly detailed in [XILINX'11d] hence does not need further introduction in this section. Just certain characteristics related with the proposed design methodology are to be explained.

- The physical definition (size, shape and position) of the reconfigurable areas within the FPGA is one of the procedures carried out in Plan Ahead. As already mentioned, several design factors such as partial bitstream size or maximum clock frequency are dependent on this definition, therefore it requires special attention. In order to appropriately adjust the size of the reconfigurable areas, Plan Ahead offers a tool that estimates their resource use percentage (Figure 18). Considering that an excessive resource occupation may bring about undesired effects in the maximum clock frequency, a trial and error process is advisable. Unfortunately, it is necessary to perform the whole implementation process in order to obtain the timing information. Consequently, to avoid delaying the design process, this adjustment should only be carried out once a concrete partition has been elected.

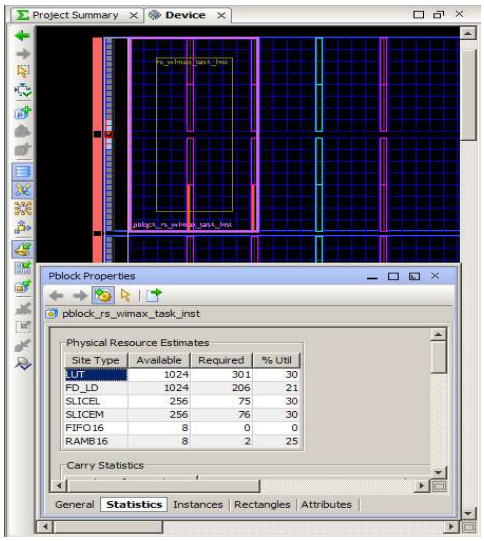


Figure 18: Detail on Plan Ahead floor-planning and resource percentage estimation

- Regarding this fine grain design adjustment, Plan Ahead includes several implementation strategies that are able to optimize the final implementation in different ways. Each strategy modifies the internal algorithms of the synthesis tools looking for a speed, area o balanced optimization. The obtained results, although not statistically significant compared to the ones obtained by the design methodology (please note that these optimization strategies provided by Plan ahead should not be confused by the optimization strategies considered in the proposed design methodology), permit a further improvement of the design performance. As before, this kind of optimization is only advisable in the latest design steps in order to reduce the design time.

- To conclude, we would like to remark that this tool generates several reports where all the necessary information related with design factors to be measured is present.

4. METHODOLOGY AUTOMATION

One of the objectives of this methodology looks for the reduction of the design time of Software Defined Radios within the proposed design framework made up of FPGA dynamic partial reconfiguration and rapid prototyping tools. Therefore, the aforementioned design steps are intended to be automated using software tools, reducing, as far as possible, the necessity of manual work. Unfortunately, due to the necessity of limiting the scope and duration of this research work, only the basics that this automation should meet have been established and certain tests on its feasibility carried out. As it will be later presented within the "Future work" section (4) of Chapter 8, the complete development of this automation would be the direct continuation of the work presented by this thesis.

Two of the characteristics of the proposed design methodology and design framework are precisely the key point of the intended automation. The fact that many functions are very similar to each other in different radio communication systems and the use of rapid prototyping tools in the function design permit on the one hand the reutilization of functions between different designs. A data base can be generated into the libraries of the aforementioned tools, hence being possible to reuse previous implementations. This way, time consuming tasks such as function design or performance measurements can be avoided. On the other hand, these tools also offer some computing capacity and the possibility of executing user-defined programs; therefore, the design partitioning procedure and the cost analysis can also be carried out automatically. Finally, taking into account the possibility of executing many of Xilinx's tools via command line [XILINX'11b], the automation of the final physical implementation of the designs is also feasible. Nevertheless, the next subsections will further explain the intended automation.

4.1. Software tools for the automation

The use of "software tools", generically, has been presented as the base for carrying out the automated version of the design methodology. However, tests on two concrete types of tools have

been performed in order to check their suitability for this work: Matlab/Simulink and Domain Specific Language (DSL) tools [Mernik'05]. Both tools are intended to act as high level design tools in the proposed design methodology and design framework. That is, these tools' main task is to automatically carry out the steps that make up the design methodology and to control the rest of the tools that take part in the design (rapid prototyping tools, tools for synthesis and implementation of FPGAs... etc). Consequently, it has been checked that these tools, on the one hand, are able to execute used defined code, (e.g. code programmed in C++ that would be in charge of evaluating the cost function), making remote calls to the FPGA synthesis tools and so on. On the other hand, the tools also have to be able to automatically generate designs using the format required by rapid prototyping tool.

Matlab, and its graphical programming environment, Simulink, is the first tool that has been evaluated. Matlab is able to execute both, scripts programmed in its own language and C/C++ compiled functions or programs. Consequently, it is able to carry out those steps in the methodology that require mathematical computations or the execution of algorithms. Besides, System Generator, one of the rapid prototyping tools used during this research work, is precisely executed over Matlab/Simulink. Therefore, it is possible to control this rapid prototyping tool from the aforementioned scripts. Moreover, dealing with the need of automatic generation of designs in System Generator, Matlab provides certain functions that are able to manage this design procedure. Actions such as inserting a block from a library, interconnecting blocks or editing block parameters can be carried out from Matlab.

DSL tools are the second possibility that has been evaluated. These tools able to generate a domain-specific language, that is, a programming language or specification language dedicated to a particular problem domain. In our case this problem domain could be summed up as the implementation of Software Defined Radios over rapid prototyping tools. DSL tools are some kind of rapid prototyping tool and are usually provided with a graphical interface that permits a simple and intuitive programming of the target application. Afterward, these tools automatically generate the end-point implementation of this application in the selected programming language. The main particularity of this type of tools is that the output language is customizable. Regarding the aforementioned two characteristics that are needed for the automation of the methodology, DSL tools are also able to execute user-programmed C/C++ code; hence the consequent methodology operations can be carried out. In turn, the code generation possibility, that constitutes the key point of this type of tools, can be harnessed in order to generate the designs in the corresponding rapid prototyping tool. Focusing on System Generator, the generated models with this tool (which are really based on Simulink models), are stored in files with extension .mdl. These are text files which store all the information about the graphically created model. Therefore, the language in which this description is carried out can be used as output language of the DSL tool. Definitely, DSL tool are also able to automatically generate designs in the selected rapid prototyping tool.

4.2. Automated design methodology

The automated version of the design methodology is intended to cover the following design steps. In should be noted that the order of certain design steps may change due to the automation:

- The design process would start generating, in the selected high level tool, a simple description diagram that contains the functions present in each waveform. These diagrams, that would be similar to the ones presented in Figure 26, Figure 27 or Figure 28, just contain black boxes (i.e. empty boxes) and the name and characteristics of the corresponding functions.

- Next, this diagram would be parsed and the analysis planned by the first two steps of the design methodology (namely: analysis of the waveforms and establishment of the common parts) carried out. It should be noted that the aforementioned capability of executing used-defined code, which is required for high level tools, would enable the automation of these processes. During the process of analysing the different implementation possibilities of each function, the already available information of previous designs would be used. As will be seen later, once a successful design is carried out, the generated function implementations would be stored in the library of the rapid prototyping tools so that they could be used in the future. Dealing with the implementation of the functions (both in their parameterized or waveform-linked version), if the implementation of a certain function is not available in the library, the tool would ask the designer to introduce it. Obviously, this process would be carried out by hand with the corresponding rapid prototyping tool.

- Once the analysis is carried out, the graph model of the system would be generated. Please note that the explanation of what this graph model represents will be later presented in Section 2.1 of Chapter 4. Besides, the "function commonality list" and the "requirement sheet" would also be automatically generated at this point.

- Taking into account that step number 3 in the design methodology ("Reconfigurable function implementation") would have already been covered by previous steps, the optimization loop (made up of steps 4, 5 and 6) could start. The generation of the possible partitions and the evaluation of their cost via the cost function would be automatically carried out in this step. These procedures would also be carried out by the execution of user-defined code. Besides, the possibility of using optimization algorithms such as Simulated Annealing, which has been presented in Section 3.4, would also be introduced in this design step.

- In regards to the final, physical implementation of the design, the high level tool would be in charge of both, the generation of the designs in the corresponding rapid prototyping tool and the management of the process of VHDL code generation and final synthesis and implementation. On this purpose, the blocks available in the libraries would be

automatically placed and connected in a new design, (based on the information introduced in the first step) and the necessary implementation programs called-back.

- Finally, once a successful design would have been carried out, those new functions that would have been implemented would be stored in the library. This way any future design could make use of them. Besides, information on the design factors of these functions (i.e. FPGA resource consumption or reconfiguration time) would also be stored in a database also to be used by future developments. It should be noted that as the number of designs carried out with the automated design methodology increases, the available functions within the library would be bigger and the benefits of using it would be enhanced.

To sum up, it has to be remembered that the abovementioned automated steps are just a proposal. Further work and analysis on the concrete drawbacks and limitations of this automation is needed. Nevertheless, the basics of the proposed automation seem, a priori, feasible and highly interesting.

5. SUMMARY

The design flow of the proposed design methodology has been presented in this chapter. Initially, the degree of freedom over which the methodology acts has been introduced. Later, the basics of the steps that make up the design methodology have been thoroughly explained. Finally, an introduction to the intended automation of the methodology has been carried out. Unfortunately, it has not been possible to develop this automation and remains as a future work. The next chapters will individually address the details of the key steps within this methodology, i.e. the establishment of common parts and the evaluation of system's performance via a cost function. Besides, a use case example of the application of the methodology will be presented with the implementation of a multi-standard modulator.

Chapter 4

The 'Common functions / common operators' technique and parameterization

4. The 'Common functions / common operators' technique and parameterization

1. INTRODUCTION

The keystone for reducing the undesired reconfiguration time within any implementation using FPGA dynamic partial reconfiguration resides in the reduction of the amount of resources to be reconfigured. On this purpose, and focused on the implementation of Software Defined Radios, the common parts or functions of the different waveforms to be integrated in the SDR can be implemented statically and hence not reconfigured. Consequently, a proper identification of these common parts is crucial as it has already been stated in the methodology design flow.

This chapter thus introduces the "Common functions / common operators" technique. This well known technique precisely aims to "*search for and find all the commonalities between several different standards in order to optimize the resources during the equipment's implementation and/or the execution phases*" [Alaus'09]. The original and generic technique is firstly presented and later adapted for its use on the design framework presented by this thesis. Subsequently, the second part of this chapter addresses the necessity of "Function parameterization". This procedure is in charge of making the similar parts of functions in the design really compatible. Therefore the chapter concludes with the presentation of the basics of function parameterization and with examples of several functions that have been parameterized throughout the development of this research work.

2. THE ORIGINAL TECHNIQUE

In the context of identifying the common aspects of the mobile communication modes, the common functions technique was firstly introduced by Rhiemeier in 2002 [Rhiemeier'02]. Subsequently, several authors evolved and improved the concepts proposed by the original technique and in 2006 Moy et al. presented the common operator approach [Moy'06]. Research and use of these two techniques, however, is still an ongoing task nowadays looking for the improvement of SDR implementations.

The original common functions technique addresses the search of commonalities from a, if you will forgive the repetition, function level. It analyzes the similarities between several communication systems in terms of high level data processing functions such as coding, equalization, synchronization and so on. This way, functions at this granularity level are the ones proposed to be reused. Subsequent research considered this technique too fixed, as the functions are highly dependent on the communication standards. Therefore, the common operators technique was developed in order to find similarities in a finer grain level. This technique aims to find low level operators, based on structural aspects, which can be reused by any of the functions within the SDR. That is, the reuse of the common operator does not only happen when a standard change happens but also during a normal execution when different functions are called. From an operational point of

view, the common operators technique breaks up the functions targeted by the common functions technique into the operators that make them up. This operation can be performed several times hence obtaining common operators also in different functional levels, e.g. FFT, butterfly and MAC unit. Figure 19, presented in [Godard'07] perfectly shows this multi-level organization. It should be noted that the technique considers the possibility of identifying several implementation possibilities for each function. This way the probability of finding a common operator is increased.

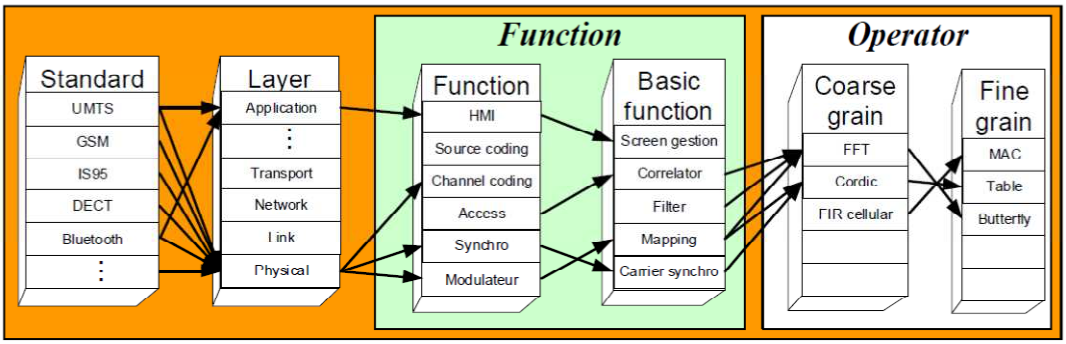


Figure 19: Multi-level organization of possible common operators

Considering the common operators technique just as an evolution of the common functions one, the approach they use to identify or create possible commonalities can be extended to both of them. Ultimately, this technique uses a graph model for the identification of these commonalities and an optimization process in order to find the most relevant ones.

2.1. Graph model

The process of successive function or operator breakdown is carried out, with a graphical approach by hand. A graph, with several layers depending on the granularity of the considered function/operator, is gradually generated. The graph is made up of "nodes" (representing the processing elements) and "hyperarcs" (showing the dependencies between the "nodes" within the different levels). These dependencies between the so called "parent nodes" and the nodes placed in underlying levels ("descendant nodes") are arranged in two ways. On the one hand, the OR hyperarc (represented by a direct arrow) indicates that only one of the descendant nodes is necessary for the implementation of the parent node. On the other hand the AND hyperarc (represented by an inverted Y) means that all the descendant nodes are necessary. It should be noted that a certain node may have both AND an OR dependencies.

The graph generation starts with the placement of the highest-level nodes, i.e. the communication standards to be implemented. Then, each of the nodes can either be directly implemented in the system or split into the several lower-level nodes that make up its functionality. Note that the functionality of a certain node may not only be achieved in a single way (this circumstance is precisely represented by the OR hyperarcs). Then, the development of the graph continues

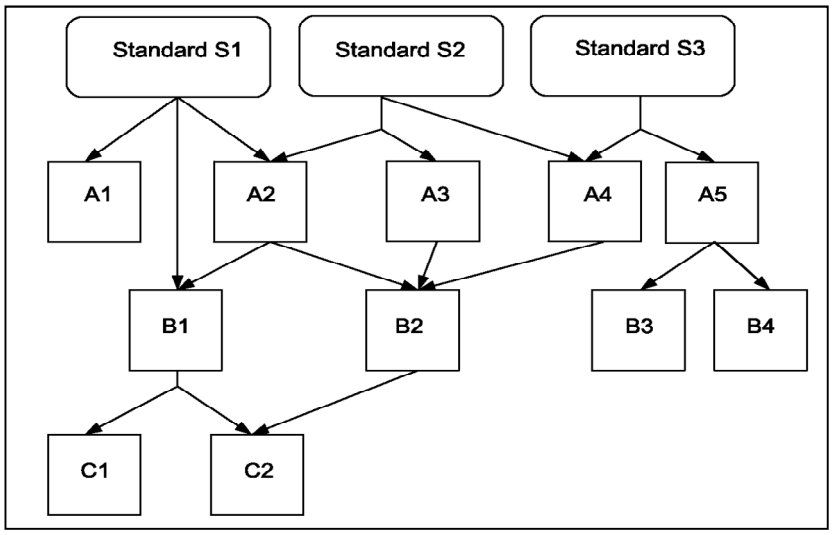


Figure 20: Graph of a generic tri-standard SDR

progressively splitting each node into simpler functional modules until nodes with indivisible operators are reached.

Figure 20 gives an overview of the generated graph of a generic, tri-standard, SDR implementation. The multi-level organization of the nodes and the AND and OR hyperarcs can be easily observed. Regarding the hyperarcs and by way of example, Standard S1 could be implemented through any of the nodes A1, B1 or A2. However, S3 requires both A4 and A5 for achieving its functionality. In turn, S2 could be implemented either via A2 and A3, or simply through A4.

2.2. Optimization process

Once the graph is completely generated and the possible common operators identified, the technique aims to select the optimal ones. The optimization problem that arises is to be solved in terms of "*monetary cost*" and "*computational cost*". Due to the generic nature of this technique, it is not oriented towards any concrete implementation technology. Consequently, the definition of both the monetary and computational cost is a complex task. The authors propose a generic cost function that evaluates these two factors. A certain common operator is intended to be "installed" (in form of HW accelerator for a DSP, hard macro for a FPGA... and so on) only once in the final implementation. However, it can be called as many times as necessary by the higher level modules or functions. Consequently, the monetary cost (e.g. FPGA resources or ASIC development cost) is computed only once by the cost function while the computational cost is affected by the number of calls to the operator under scope.

3. TECHNIQUE ADAPTATION TO THE CURRENT DESIGN FRAMEWORK

As it can be observed, the research area that the above presented technique covers has several similarities with the objectives that our design methodology pursues. Unfortunately there are two characteristics that make it unsuitable for our design framework. On the one hand there is the

general nature of the technique, in terms of implementation technologies. The technique would need further development to fit with the particularities of FPGA dynamic partial reconfiguration and rapid prototyping tools. On the other hand, the software-like orientation towards a single implementation and multiple executions of the common operators is totally opposite to the pipelined orientation achieved by rapid prototyping tools. In this respect, the fine grain granularity achieved by the common operators technique, addressing basic operators such as shift registers or multiply-accumulate units, seems also excessive and inefficient for our approach. In other words, due to the characteristics of our proposed design framework, the granularity addressed by the original common functions technique is the one that most closely resembles our approach.

Consequently, only certain characteristics of these techniques are ported to our SDR design framework and design methodology. Namely: the graphical approach for the determination of the possible commonalities, and the concept of multiple possible implementations of a certain function. This last concept is the one addressed by the AND and OR hyperarcs. Besides, although the fine granularity achieved by the original common operators technique is not considered in our approach, the division of a certain function into lower-level nodes is harnessed in the parameterization of functions. This fact will be covered in the next section.

Definitively, the first step in the proposed design methodology, i.e. the waveform analysis step, firstly generates the aforementioned graph. This step, in the non-automated version of the methodology is carried out by hand, that is, the designer has to generate the graph based on its experience or in a search within the state of the art. However, in other to avoid the use of paper and in order to obtain a clear representation, generic drawing applications (e.g. Microsoft Visio) can be used to draw the graph. Moreover, the graphical interface that most rapid prototyping tools have can also be harnessed on this purpose, hence avoiding the use of more programs. Nevertheless, it should be remarked that just the drawing possibilities (boxes, arrows, text... etc.) of these tools are to be used on this step.

As in the original technique, the graph generation starts with the placement of the standards to be implemented in the SDR. Subsequently, the progressive function simplification procedure is carried out. In our approach, the functions present in the libraries of the concrete rapid prototyping tool to be used, are the lowest level ones to be considered. This leads to final nodes still representing quite high-level functions. It also should be noted that, generally, this is an experience-based process. The knowledge on the different possibilities of implementing a certain function or the "descendant nodes" that make up the functionally of a "parent node" have to be identified in literature or obtained from personal experience.

Once the graph is completed, the commonalities are identified by seeking the nodes (i.e. functions) used by at least two waveforms. Besides, also the stand-alone functions that are only present within a single waveform are indentified. With this information, the "function commonality list" presented in Chapter 3 and available in Annex 1 can start being filled. First of all, just the names of all the functions or operators identified in the design are listed. Afterwards, each function is labelled as "common" if it is present in more than one waveform or "reconfigurable" if it is used only once and

hence has to be implemented in a reconfigurable area. Finally, in other to ease the parameterization step presented later in Section 4, the "function commonality list" is completed with the concrete functional characteristics of each function (e.g. input-output interfaces, number of coefficients, generation polynomials... and so on). It should be noted that if a certain "common" function has different characteristics in each waveform, a piece of information per each different set of characteristics has to be filled.

If the methodology is automated as explained in Section 4 of Chapter 3, the advantages of rapid prototyping tools can be harnessed in order to ease the deployment of the common functions/operators technique. Briefly, taking advantage of the possibility of storing already implemented functions in the libraries of the rapid prototyping tools, the automated methodology uses this information in order to identify the commonalities. In this way, the graph generation is automatically carried out, based on the information stored beforehand. Just introducing the names of the high level functions present in each waveform, the tool makes a search within its libraries looking for already used implementation possibilities. This implementations contain the information or both, the lower-level functions needed (AND hyperarcs for the graph) and the different implementation possibilities (OR hyperarc). In case there is not a previous implementation of a certain function, the tool requests this information from the designer.

4. FUNCTION PARAMETERIZATION

As already explained in Chapter 3, the functions identified as "common" in the previous design step still need to be analyzed to decide whether they are suitable for being reused among different waveforms or not. This is due to the change of internal characteristics that a certain function has when compared across different communication standards. In order to be reused, that is, in order to be implemented statically, the target function has to permit the change of certain parameters on runtime. The process of providing a certain function with this possibility is known as "parameterization". At this design step, the "function commonality list" already has the necessary information on the functional characteristics required by each function in each standard. Therefore, the function parameterization design steps aims to analyze this information and design each function so that it admits the necessary changes in its characteristics. Besides, and regarding the ultimate target of deciding if a certain function has to be implemented statically or in a reconfigurable area, this design step is also in charge of designing the reconfigurable version of each function.

Consequently, in case the function design has to start from scratch, that is, in case there are no previous implementations from other designs, the non-parameterized versions of each function are designed firstly. These designs (one per each different set of characteristics) are carried out harnessing all the possibilities offered by rapid prototyping tools. Bearing in mind the reconfigurable character of these implementations, the designs should look for the smallest possible form-factor. Information on the different implementation possibilities can be extracted from the graph generated in the previous design step. Once the designs are completed, the information on FPGA resources used

and reconfiguration time has to be generated (or estimated) and added to the "Function commonality list".

The design flow continues then with the design of the parameterized version of each function. This is usually an experience-based step. However, the information compiled both in the "Function commonality list" and in the function graph is highly valuable. On the one hand the "function commonality list" provides data on the type of characteristics that have to be parameterized. Consequently, the designer can start evaluating the best implementation structure so as to ease this procedure. On the other hand, the function graph, if we focus on a particular function, offers information on the operators or sub-functions that make it up. Although this methodology does not consider further action with these operators, the required change of characteristics usually only affects one of the operators while the rest remain unchanged. Therefore, the design of the function can be developed from this point of view. That is, the designer can identify the operator that needs to be changed and generate an appropriate implementation to support it. The multiplexing based parameterization presented later in Section 4.1 is an example of this approach.

To conclude, just as in the design of the non-parameterized versions of the functions, once the design of the parameterized versions is finished, the corresponding information on the factors that describe their performance have to be added to the "Function commonality list". It should be also noted that the function parameterization step considers two exceptions to the above explained procedure: the direct use of the common function and the impossibility of carrying out the parameterization. On the one hand two waveforms may use exactly the same function. In that case, parameterization is not necessary and the function can be directly used. On the other hand, certain functions may not be parameterized. Due to their implementation structure, or due to design limitations of rapid prototyping tools, the parameterization of this type of functions would imply a complete change of the function. Consequently, the benefits achieved in terms of reconfiguration time would not be enough compared with the increase suffered by resource utilization. In such a context, the designer can directly change function's status in the "Function commonality list" to "Reconfigurable" or may continue the design process and leave to subsequent design steps the rejection of the static implementation.

4.1. Parameterization techniques

During the development of this research work, three main parameterization techniques have been identified:

- Direct parameterization on rapid prototyping tools' functions. Several functions available in the libraries of rapid prototyping tools already provide the possibility of changing certain parameters on runtime. Focusing on System Generator [XILINX'12c], and by way of example, both the Fast Fourier Transform block and the FIR Compiler provide this possibility. On the one hand it is possible to change the transform length of the FFT block. On the other hand the filter coefficients of the FIR compiler can be reloaded on runtime. Subsection 4.3 will later show concrete examples of these implementations.

- Multiplexing based parameterization. As stated before, it is usual that the required change of characteristic within a function is restricted to one of the operators that make it up. Therefore, a simple functional way of achieving the parameterization consists on the multiplexing of that operator. That is, two different implementations of the changing part of the function to be parameterized are implemented and multiplexed, while the rest maintains its original structure. This dual implementation and multiplexing is one of the responsible for the size increment of the parameterized functions explained in Section 6 of Chapter 2.

- Direct ICAP access parameterization. In order to lessen the aforementioned size increment, FPGA dynamic partial reconfiguration can be harnessed in certain functions. Although function parameterization aims to maintain common functions implemented statically, hence reducing reconfiguration time, it is possible to avoid the use of multiplexers at very low cost by using localized dynamic partial reconfiguration. In some functions, the characteristic that changes from one waveform to another is stored in resources that are directly accessible via ICAP. Consequently, instead of multiplexing the different versions of the target characteristic, it can be overwritten by the ICAP. By way of example, and further explained in the next subsection, the interleaving pattern within an interleaver is usually stored in a BRAM. BRAMs' content being one of the resources easily accessible via ICAP, it is not necessary to multiplex two BRAMs in order to have two different interleaving patterns. Just overwriting the BRAM content with the new pattern the goal is achieved. Besides, due to the small amount of data to be transferred, the reconfiguration time is minimum. Figure 21 shows the addressable resources within the FPGA and the way of generating the FAR (Frame Address Register) addresses [XILINX'09] needed to perform this direct access. In regards to the usual, time consuming, design flow of dynamic partial reconfiguration, note that this parameterization technique does not modify any of the routes of the design. Consequently it is not necessary to follow this design flow and the implementation is straightforward.

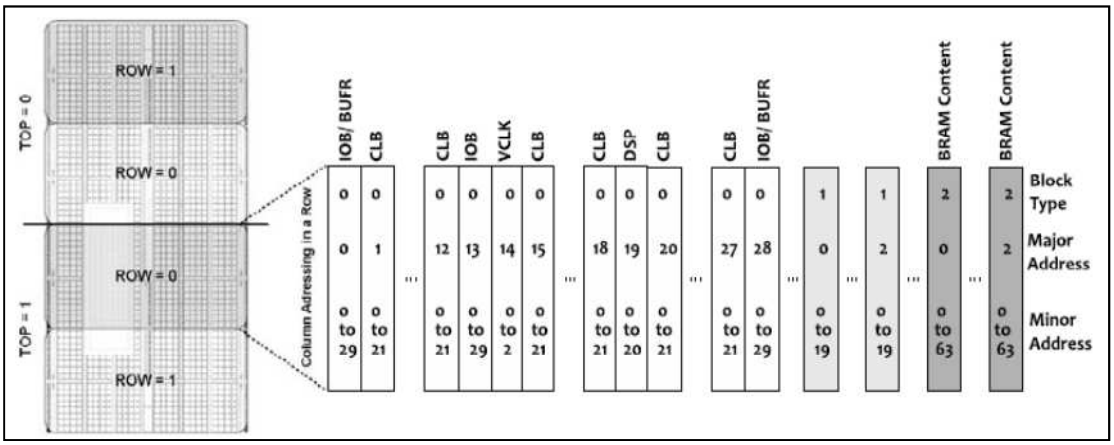


Figure 21: Virtex-4 frame addressing scheme (XC4VFX12 part)

4.2. Sample time parameterization

In another design level, another characteristic that is susceptible to be parameterized and that needs a detailed analysis is the sample time. Due to the pipelined architecture generated by rapid prototyping tools, multi-rate designs are very common. In these designs, each function in the processing chain may work with a different sample time depending on the data flow generated by its surrounding functions. Consequently, although a certain function is common to several waveforms, it may use a different sample time in each of them, hence also requiring the parameterization of this feature.

Bearing in mind that clock management is essential in any synchronous logic design, FPGAs are provided with dedicated clock nets. These nets deliver clock signal properly along the whole device. Therefore, carrying out a manual multiplexing of the different clocks needed by a certain function it is not a good design practice. The introduction of a logic element such as a multiplexor makes clock routing use the normal routes within the FPGA, hence suffering a performance loss. The ideal solution for achieving sample time parameterization lies in the use of Digital Clock Managers (DCM). These resources are able to manage a proper generation of clock signals with different frequencies. Unfortunately DCMs are neither partially reconfigurable, nor directly parametrizable in some FPGA families [He'12]. Therefore, for those families in which direct runtime parameterization is not possible an alternative solution is needed.

To achieve this, the particular way of addressing multi-rate designs by System Generator has been harnessed. In order to avoid an excessive use of the dedicated clock nets in this type of designs, System Generator bases multi-rate design in the use of clock enable signals. This design practice uses a single clock signal with a frequency that is the greatest common divisor of the different frequencies present in the system. Besides, a clock enable signal is also delivered to each element indicating when the general clock is valid. These signals are precisely the ones that generate a virtual multi-rate system. Taking into account that these clock enable signals are implemented through the normal routes of the FPGA multiplexing them does not generate any further drawback. Figure 22 shows a functional diagram of the proposed solution for sample time parameterization:

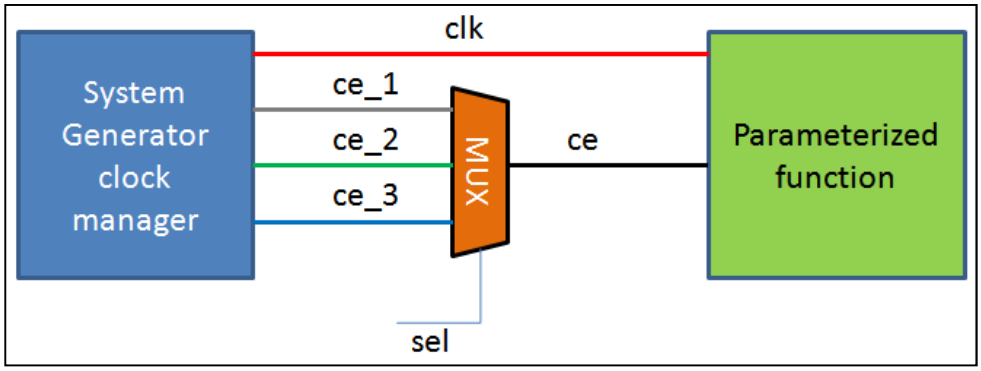


Figure 22: Solution for sample time parameterization

Clock management is usually a hidden characteristic in rapid prototyping tools. Therefore, the proposed sample time parameterization procedure considers a local modification of the generated VHDL code in order to achieve the pretended objectives.

Finally, it should be noted that dynamic partial reconfiguration can also be harnessed in order to change the sample rate of a certain function. The use of regional clocking resources (BUFR) can be used on this purpose as has been presented in [Iturbe'12b]. Unfortunately this approach, originally developed for the R3TOS operating system that will be presented in Section 3.2 of Chapter 7 needs further research in order to be fully compatible with our design framework.

4.3. Developed SDR parameterized blocks: examples and characteristics

Some examples on real parameterized functions developed during the multi-standard modulator presented in Chapter 6 will be displayed below to illustrate the information introduced in this chapter.

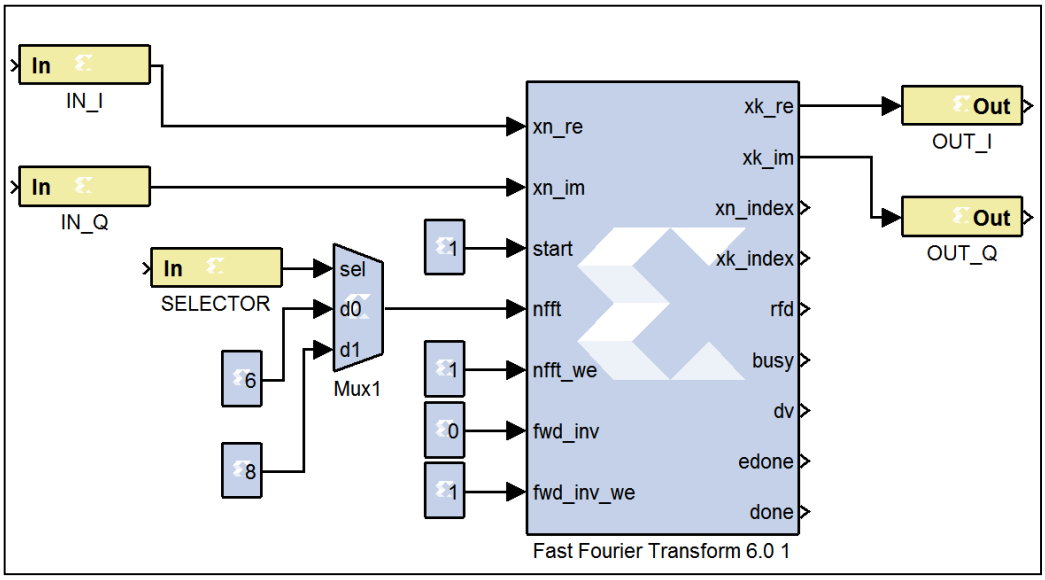


Figure 23: Parameterized IFFT function

Figure 23 shows the parameterized version of the Inverse Fast Fourier Transform function. As already stated, the IFFT block provided by System Generator permits the direct parameterization of the transform length. This length is indicated via the 'nfft' port in the format of $\log_2$(IFFT length). That is, the shown implementation can perform a 64 tap or 256 tap IFFT. As can be observed the election of the IFFT length is carried out multiplexing the corresponding constants in this case. However, the direct ICAP access could also be used to overwrite the content of a single constant (usually implemented in LUTs) and therefore suppress the use of a multiplexer.

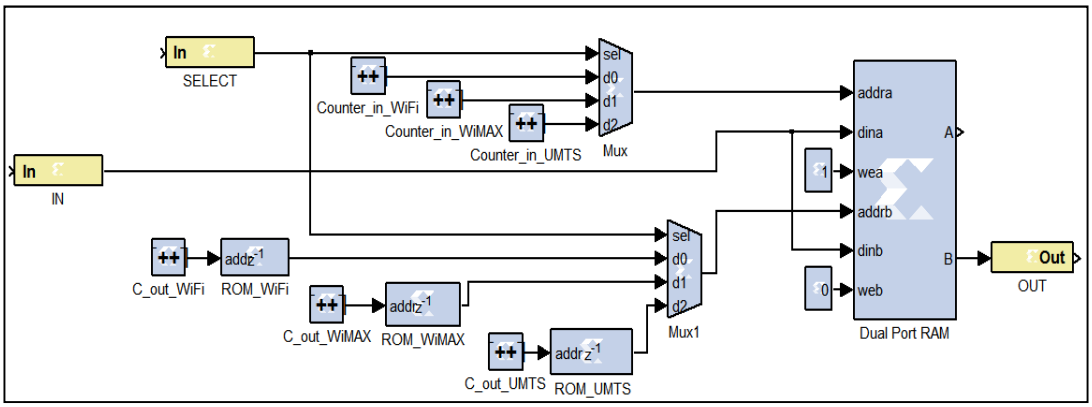


Figure 24: Parameterized interleaver function

As an example of the multiplexing based parameterization, Figure 24 shows the implementation of a parameterizable interleaver. In this case, both the interleaving pattern and the interleaver length are parameterized in order to fulfil with the requirements of the WiFi, WiMAX and UMTS communication standards. As can be observed, the operator common to all the standards is the dual port RAM while a pair of multiplexers selects the input and output counters and the interleaving patters stored in ROM memories. It should be noted that the dual port RAM must be as big as the biggest needed implementation. Regarding the three ROM memories that store the interleaving pattern, a second possible implementation of this function is possible. Harnessing again the direct ICAP access, a single ROM whose content is updated with this method could be used, hence removing two out of three memories (usually a limited resource in the FPGA). This new implementation can be observed in Figure 25.

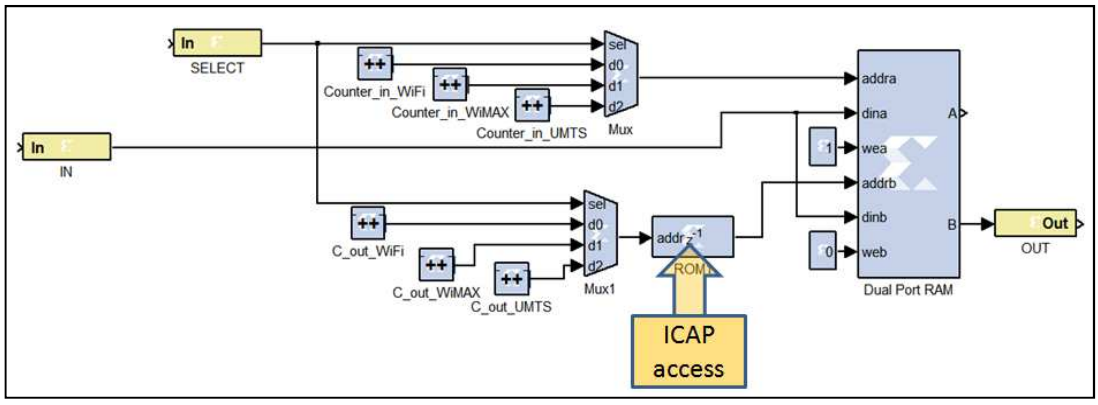


Figure 25: Parameterized interleaver (direct ICAP access version)

Further function parameterization examples are given in Chapter 6.

5. SUMMARY

The present chapter has covered two important design steps within the design methodology proposed by this research work, namely: the waveform analysis via the common functions / common operators technique and function parameterization. These two steps aim to identify and properly

design the common parts between the several waveforms to be implemented in a SDR, hence enabling their static implementation. It has been demonstrated that the original common functions / common operators technique is not suitable for our design framework due to dissimilarities in both the target granularity and the way the functions aim to be executed. Consequently, this techniques' port to our design methodology has been explained. Later, function parameterization, has been defined and explained. This procedure makes the functions identified as common in the previous design step suitable for a static implementation. At this step, the first real implementations of these functions are carried out with rapid prototyping tools and the necessary information for subsequent design steps is generated.

Chapter 5

Cost Function

5. Cost Function

1. INTRODUCTION

The present chapter will introduce the cost function that quantitatively evaluates the optimality of each of the partitions generated in the "Design partitioning" step of the design methodology. As has been mentioned several times before, three main factors describe the performance of a certain system; namely: resource utilization, reconfiguration time, and maximum clock frequency. Consequently, the proposed cost function takes into account these three factors. Depending on the selected design flow, some of these factors may have not been generated when the cost function has to be applied. Therefore, this chapter will also present other auxiliary functions that make an estimation of them.

2. GENERAL COST FUNCTION

In order to evaluate and compare the performance of the design partitions generated in the "Design Partitioning" step of the design methodology, a cost function has been designed. This function quantitatively evaluates the three factors that have been identified as most relevant on the Software Defined Radios implemented with the proposed design methodology. These factors that have already been presented in Section 6 of Chapter 2 are: design size, reconfiguration time and maximum clock frequency. The design size, represented by the number of resources occupied within the FPGA, is the factor that aims to be reduced with the introduction of FPGA dynamic partial reconfiguration. Reconfiguration time is the main drawback that partial reconfiguration introduces. Finally, maximum clock frequency is affected by both dynamic partial reconfiguration and rapid prototyping tools; hence it also needs to be evaluated.

Initially, the general, complete cost function will be presented. This cost function evaluates the aforementioned three factors. It has to be taken into account that depending on whether the final physical implementation of each of the partitions has been carried out or not, some of these factors may not be available at the time of applying the cost function. Consequently, these factors may be neglected or estimated from any of the remaining available factors. On this purpose, Section 3 will later present some estimation functions that have been generated from the experience gained during this research work.

The general cost function for a concrete partition "*p"* is defined as follows (3):

$$Cost(p) = \alpha \cdot \frac{Resources_{Max.}(p)}{\overline{Resources_{Max.}}} + \beta \cdot \frac{T_{RECONF.Avg.}(p)}{\overline{T_{RECONF.Avg.}}} + \gamma \cdot \frac{T_{CLK\ MIN\ Max.}(p)}{\overline{T_{CLK\ MIN\ Max.}}} \quad (3)$$

Please note that as it was previously introduced in Section 3.5 of Chapter 3, the maximum clock frequency has been substituted in the cost function by its inverse, i.e. the minimum clock period, as it is mathematically more coherent.

Besides, it should be noted that parameters α, β and γ multiplying each of the factors are weighting parameters that permit the adjustment of the optimization goal of the methodology. Further details will be explained in Section 2.4.

Further explanation on the diverse parameters and sub-equations present in the cost function will be explained below:

2.1. Normalized design size calculation

The first term (4) in cost function (3) calculates the relative value of the resources occupied by a certain partition "*p*":

$$Term\ 1: Normalized\ Resources(p) = \frac{Resources_{Max.}(p)}{\overline{Resources_{Max.}}} \quad (4)$$

It should be noted that in order to properly add the different factors present in the cost function, they have to be normalized beforehand. This normalization is carried out with the average value obtained from the whole set of partitions ("P"). Consequently, each term does not evaluate the absolute value of the corresponding factor but the relative one. In this way, a cost value of 1 indicates that a certain partition falls within the average value. A bigger than one value represents the relative increase in relation to the average, and a less than one value, a decrease.

Subscript "Max." indicates that the value that needs to be used for this computation is the maximum number of occupied resources. We note that each partition "*p*" is made up of "N" configurations. Each configuration "n" corresponding to one of the waveforms present in the SDR. That is, an SDR like the one presented in Chapter 6, which implement the WiMAX, WiFi and UMTS standards, has N equal to 3. Although the complexity of each standard is different, hence occupying different number of resources, the most complex one establishes the minimum number of resources needed in the FPGA. This is way the number of resources of the configuration with higher number of resources is used in the evaluation of the design size.

Consequently sub-function (5) describes the aforementioned maximum resources of a certain partition, "p":

$$Resources_{Max.}(p) = Max\ \{Resources(p,n)\} \quad , \quad n = 1,2,\dots,N \quad (5)$$

It has to be taken into account that the number of resources occupied by each waveform "*n*" of a certain partition "*p*" is the result of adding the resources of all the individual functions that make up the whole waveform. That is, the resources occupied by both the functions implemented statically

and in a reconfigurable way have to be added. Furthermore, in order to properly evaluate the resource consumption of the whole system, the resource overhead introduced by the infrastructure that enables the dynamic partial reconfiguration (ICAP controller, a microprocessor or reconfiguration manager, memory controllers...) also has to be added. Section 3.2 will later address this issue.

Sub-function (6) shown the way of computing the average value of the resources occupied by the different partitions:

$$\overline{Resources_{Max.}} = \frac{1}{P}\sum_{p=1}^{P} Resources_{Max.}(p) \quad (6)$$

"P", already introduced in the previous paragraph, represents the number of different possible partitions. That is, the number of partitions to be evaluated with the cost function. Therefore, equation (6) computes the arithmetic mean of the occupied resources.

The presented equations use the generic word "Resources" in order to represent the partition size. In order to carry out the cost calculation, this generic representation has to be substituted by an appropriate design parameter that represents the size. The generic nomenclature is used in the cost function to generalize its use to any FPGA manufacturer. Although the FPGA structure is similar in all the manufacturers, the way of naming the resources is different (e.g SLICE in Xilinx, LE - Logic element- in Altera), therefore, the generic word "Resources" is the one that best represents the factor that intend to be evaluated. FPGA manufacturers used to provide the "equivalent gate count" or "system gate count" to represent the design size, as it is a common parameter in the ASIC design world. Unfortunately, with the complexity increase suffered by FPGAs the estimation of "equivalent gate count" become also more complex and started not to be reliable. Therefore, most of the FPGA manufacturers no longer provide it [Maxfield'10].

In the specific case of this research work, we have established the number of occupied SLICEs as the number that represents the size of the design. SLICEs are the elementary programmable logic block in Xilinx FPGAs. Therefore, SLICEs including other resources such as LUTs or Flip-Flops, they give an overall overview of FPGA usage rate. Current FPGAs are provided with other hardware resources such as BRAMs or embedded DSP48 elements that also need to be taken into account. Section 2.5 will later address this issue.

2.2. Normalized reconfiguration time calculation

The second term (7) within the general cost function is in charge of evaluating the reconfiguration time of a certain partition "*p*".

$$Term\ 2: Normalized\ reconfiguration\ time(p) = \frac{T_{RECONF._{Avg.}}(p)}{\overline{T_{RECONF._{Avg.}}}} \quad (7)$$

Similarly to the computation of the design size, the reconfiguration time is also normalized with the average value obtained from the whole partition set "P". However, the way of calculating this average value, and the value selected to represent the reconfiguration time of a certain partition "*p*" differ slightly from the ones used with the design size. Unlike the design size, in which the biggest design represents the size of all the set of configurations N, the value selected to represent the reconfiguration time of each partition "*p"* is the average value of the reconfiguration times. Sub-function (8) carries out this computation:

$$T_{RECONF.Avg.}(p) = \frac{1}{N(N-1)} \sum_{n_i=1}^{N} \sum_{\substack{n_j=1 \\ n_j \neq n_i}}^{N} T_{RECONF.}(n_i, n_j, p) \quad (8)$$

It can be observed that for a certain partition "*p*", the average value of the reconfiguration times for all the possible transitions between its waveforms (n_i being the initial waveform and n_j the target waveform) is calculated. That is, for a SDR with 3 waveforms A, B and C (therefore N = 3), the average value of the reconfiguration time of the 6 possible transitions (A -> B, B -> A, A -> C, C -> A, B -> C and C -> B) is calculated. Besides, the reconfiguration time of each waveform is made up of the addition of the reconfiguration times of all the reconfigurable areas that change within it. It should be noted that if more than two waveforms are present in a SDR, a transition between two of them may not require reconfiguring all the reconfigurable functions. Some of the functions may remain unchanged if they are related with a third waveform. Further discussion on this issue will be carried out in Section 4. Taking into account that an individual partial bitstream is generated per reconfigurable area, the overall reconfiguration time is the addition of the necessary time to download the appropriate bitstreams through the ICAP port.

The average value of the reconfiguration time has been chosen to represent this factor (instead of its maximum value) because in this case it does not establish a design limitation. Section 2.5 will later present the additional checks on design requirements that are applied to each of the partitions. The maximum acceptable reconfiguration time is one of them. However, if this requirement is met, the average reconfiguration time is a better indicator of the quality of the partition on this particular factor.

Consequently, the way of calculating the overall average reconfiguration time for the complete set of partitions P has also changed. The sub-function that carries out this computation is presented below (9) :

$$\overline{T_{RECONF.Avg.}} = \frac{1}{P} \sum_{p=1}^{P} T_{RECONF.Avg.}(p) \quad (9)$$

This equation shows that the arithmetic mean of the reconfiguration times is now carried out using the average reconfiguration time of each partition "*p*", that is, using the value obtained from sub-function (8).

2.3. Normalized minimum clock period calculation

The third and last term (10) of cost function (3) evaluates the minimum clock period applicable to each of the partitions "*p*". Please note that it is a mathematical representation of the usual maximum clock frequency factor (2):

$$Term\ 3: Normalized\ minimum\ clock\ period\ \ (p) = \frac{T_{CLK\ MIN\ Max.}(p)}{\overline{T_{CLK\ MIN\ Max.}}} \qquad (10)$$

Dealing with minimum clock period, the value chosen to represent this factor for a certain partition "*p*" with N configurations of waveforms, is the biggest value among them. Similarly to what happens with the design size and the necessity of considering the biggest waveform, in this case the biggest value of the different minimum clock periods applicable to the waveforms has to be used. It fixes the minimum period even if some of the waveforms support a lower one. In order to represent this issue and in accordance with the nomenclature used with the design size, the subscript "Max." has been included in the equation.

Sub-function (11) carries out this calculation:

$$T_{CLK\ MIN\ Max.}(p) = Max\ \{T_{CLK\ MIN}\ (p, n)\} \quad , \quad n = 1,2, \dots, N \qquad (11)$$

We would like to remind the reader that each partition "*p*" is made up of "N" configurations. Each configuration "n" corresponding to one of the waveforms present in the SDR.

Once the definition of the maximum value of the minimum applicable clock period has been carried out, sub-function (12) calculates its average value for the complete set of partitions P. This average value is the one used in order to perform the normalization.

$$\overline{T_{CLK\ MIN\ Max.}} = \frac{1}{P}\sum_{p=1}^{P} T_{CLK\ MIN\ Max.}(p) \qquad (12)$$

2.4. Weighting parameters

One of the objectives of the proposed design methodology is the election of the optimal static/reconfigurable partition in terms of design size, reconfiguration time and minimum clock period. However, it is an interesting feature to have the possibility of prioritizing the optimization of one of the factors. That is, letting the methodology select as optimum a partition that, even if does not have the overall minimum cost, has a significant low cost on the factor that has been prioritized. On this purpose, each of the factors present in the general cost function (3) is weighted by parameters α, β and γ respectively. Setting the appropriate value of these parameters the optimization goal of the design methodology can be modified. Besides, these parameters can also be used (setting them to zero) in order to disable any of the factors evaluated by the cost function. That is, reducing the

number of factors to be evaluated in the optimization (e.g. if there is evidence that a certain estimated data does not have the necessary accuracy).

Different sets of weighting parameters have been predefined in order be easily used during the methodology application. They are presented in Table 3 below. However, they can be modified if necessary.

Table 3: Predefined sets of weighting parameters

	Weighting parameters		
Optimization goal	α	β	γ
Neutral	1	1	1
Design Size (hard)	10	1	1
Reconfiguration speed (hard)	1	10	1
Max. Clock Frequency (hard)	1	1	10
Design Size (soft)	2	1	1
Reconfiguration speed (soft)	1	2	1
Max. Clock Frequency (soft)	1	1	2

The presented sets can be divided into three groups. The first set, named “Neutral” is the default one, in which the three factors evaluated by the cost function are weighted equally. This way, the methodology is free to select the overall optimum partition. The next three sets, labelled “hard” strongly prioritize the optimization of one of the factors. The number 10 present in the coefficient of prioritized factor indicates that the cost of the corresponding factor is penalized with a factor of ten. That is, the factor’s cost has to be ten times smaller than the other two factors in order to result in an overall same cost. Finally, the last three sets, labelled “soft” also prioritize one of the factors but is a softer way. In this case, the correction is made with a factor of 2. Examples on the influence of the different sets of weighting parameter can be seen in the validation presented in Chapter 6. Please note that maximum clock frequency appears in the table instead of minimum clock period as it is the most usual way of addressing this factor.

2.5. Design requirement check

The presented cost function is in charge of quantitatively evaluating the performance of each of the partitions in order to choose the optimal one. However, the election of the best partition based just on the results obtained from the cost function is not enough. Additional checks must be carried out in order to determine if a partition meets other design requirements. For example, it is feasible that the partition with less cost has an unacceptable reconfiguration time, hence being necessary to select another partition or to carry out a re-design. This type of issue is particularly enhanced if the weight of a certain factor is reduced and therefore considered less important in the optimization.

Consequently, it is necessary to check if every partition (made up of N different waveforms) meets with the design requirements listed in the "Requirement sheet" that was generated in the first step

(Waveform analysis step) of the design methodology. If this fast checks are carried out initially, those partitions that do not meet any of the requirements can be directly rejected. Therefore the number of partitions to be evaluated with the cost function is reduced. Checks on the three factors addressed by the cost function have to be carried out.

On the one hand, if a target FPGA has been defined, it is necessary to check that all the waveforms within a certain partition fit in the available resources. The number of occupied SLICEs has been established as the indicator of the design size. Moreover, this value is the one used in order to evaluate a partition's performance in terms of design size. However, it is not enough to check only the SLICE availability in the FPGA to decide if a waveform fits in it or not. As has been stated before, FPGAs nowadays provide several hardware-implemented resources such as BRAMs, DSP48, or DCMs that are not contained in the SLICEs. Consequently, their availability has to be checked too. If these checks are to be performed based on estimated data, information on all of these resources may not be available. However, based on the experience obtained within this research work, the usage of BRAMs and DSP48 is usually the critical factor. Fortunately, data on these two resources is usually provided by the resource estimators available in rapid prototyping tools. Continuing with the problems of using estimated data, it is advisable to use a 10 - 15% safety margin to decide if a waveform fits in the FPGA or not. As it was introduced in Section 3.2 of Chapter 3, a 15% difference between the estimated and real resources can be given (mainly in the SLICE estimation). Although this difference is usually given because the final implementation optimizes resource occupation, hence achieving a resource reduction, the use of the safety margin is still advisable. To conclude with the checks on design size, we would like to remark that the overhead of the reconfiguration infrastructure has to be computed.

On the other hand, checks on the maximum acceptable reconfiguration time and on the maximum clock frequency also need to be performed. These checks are simpler, so both of them will be addressed in this paragraph. The maximum acceptable reconfiguration time is established by the SDR application itself; hence no further discussion is needed. Any partition with a waveform that has a reconfiguration time higher than the threshold has to be discarded or re-designed. Similarly, the clock frequency to be provided to the FPGA is usually also a predefined factor. This way, those partitions with a maximum clock frequency lower than the one established should also be discarded.

To sum up it should be noted that all the checks carried out within this design step are related to implementation factors. Other functional requirement checks such as maximum data throughput, BER tests or communication range are to be carried out once the final SDR is implemented and therefore they are out of the scope of this research work.

3. ESTIMATION FUNCTIONS

During the explanation of the design methodology in Chapter 3, two alternatives for reaching the design cost analysis have been presented. On the one hand, the final physical implementation of the partitions can be carried out. This way, real information on all the design factors considered by the cost function is fully available. Unfortunately, the implementation process is a time-consuming task

that can considerably slow down the design process. Therefore, this possibility is not advisable when an initial selection of the best partitions is to be done. This is why on the other hand the design methodology foresees the possibility of carrying out the initial optimization process based on estimated data. That is, with data obtained prior to any design synthesis.

Estimated data can usually be obtained from rapid prototyping tools. These tools usually provide a block (e.g. "Resource Estimation" block in Xilinx's System Generator) able to make this estimation. However, this block may not estimate all the necessary data or may take too long to estimate it. Consequently, this section will present some functions that have been designed in order to carry out this estimation. Unfortunately, maximum clock frequency (or its inverse, minimum clock period, used in the cost function) can neither be estimated by the resource estimation block in rapid prototyping tools, nor be calculated by hand. Therefore, this factor is eliminated in the cost function when dealing with estimated data and it is only considered once the final implementation of certain partitions has been performed.

3.1. Reconfiguration time estimation

One of the factors that is not estimated by rapid prototyping tools is reconfiguration time. Moreover, this factor is not provided by synthesis tools either once the complete implementation has been carried out. The real information on reconfiguration time has to be measured once the design has been downloaded into the target FPGA. On this purpose, internal timers or external measurements on signals prepared to that end can be used. In order to have access to the reconfiguration time earlier in the design flow two estimation functions have been designed.

In order to avoid the aforementioned measurements on the reconfiguration time, the first function (13) estimates the reconfiguration time based on the size of the generated partial bitstreams. The function intends to be used in case the complete implementation of the partitions has been carried out, that is, once the partial bitstreams has been generated. The estimation (noted by the caret ^ in the name) is carried out for the transition from waveform "n_i" to waveform "n_j" of a certain partition "*p*":

$$\widehat{T_{RECONF.}}(n_i, n_j, p) = \begin{cases} 2.63 \sum_{a=1}^{A(n_i,n_j,p)} Size_{.bit}(a,p), & ICAP\ speed = 380\ MBps \\ 222.22 \sum_{a=1}^{A(n_i,n_j,p)} Size_{.bit}(a,p), & ICAP\ speed = 4.5\ MBps \end{cases} \quad (13)$$

We introduce the parameter "A(n_i, n_j, p)" as the number of different partially reconfigurable areas (PRA) within the FPGA that change from waveform "n_i" to waveform "n_j" for partition "p". In turn "$Size_{.bit}$(a,p)" denotes the size in Kilobytes of the reconfigurable area "a" of the partition "p". This value is obtained from the partial bitstream. We note again that, as will be later explained in Section 4.3, not all the reconfigurable areas have to be reconfigured in a waveform change. Function (13)

adds the sizes in Kilobytes (KB) of the partial bitstreams that need to be downloaded to perform the waveform change and calculates the reconfiguration time in microseconds (us).

Taking into account that the reconfiguration time is also dependent on the ICAP speed, the function considers the speeds of the two ICAP controllers used during this research work. On the one hand the 380 MBps achievable by the ICAP controller of R3TOS (that will be later presented in Section 3.2.1 of Chapter 7) and on the other hand the 4.5 MBps reached by the standard ICAP controller provided by Xilinx. Data from Xilinx's ICAP controller has been obtained from the implementation presented in Section 2 of Chapter 7. We note that in both cases the maximum speed does not reach the theoretical maximum speed of 400 MBps of the ICAP port. The values that multiply the summations in function (13) have been directly calculated from these speeds.

The presented estimation function uses the final partial bitstreams for its calculations. In order to avoid this drawback we have defined a second function for estimating reconfiguration time that is intended for earlier design steps. Consequently, the reconfiguration time has to be obtained based on the available information. In this case, the estimation function uses as input the number of occupied SLICEs. As explained before, the number of occupied SLICEs is part of the information that the resource estimators available in rapid prototyping tools provide. Taking into account that function (13) is valid to calculate the reconfiguration time from the partial bitstream sizes, it is enough to estimate these bitstream sizes to obtain the complete estimation of the reconfiguration time. Function (14) is in charge of this calculation:

$$\widehat{Size_{.bit}}(a,p) = 0.163 \cdot SLICEs(\mathrm{a},\mathrm{p}) \tag{14}$$

This function estimates the partial bitstream size in Kilobytes (KB) for a certain partially reconfigurable area (PRA) "a" for partition "p". The complete development of the function is available in Annex 1. We would like to note that the function is valid for a PRA with a height of a complete clock region (i.e. 16 CLBs in Virtex 4 FPGAs) and a resource occupation of a 70%. This PRA size and shape is the optimal one in order to reduce reconfiguration time. The height of a clock region is the minimum reconfigurable height and higher resource occupations may lead to the impossibility of implementing the design. However, if other shapes or occupations are needed the function can be re-defined following the steps in Annex 1.

Once function (14) is substituted in (13) the complete function for estimating the reconfiguration time based on the occupied SLICEs is obtained:

$$\widehat{T_{RECONF.}}(n_i,n_j,p) = \begin{cases} 0.429 \sum_{a=1}^{A(n_i,n_j,p)} SLICEs(\mathrm{a},\mathrm{p}), & ICAP\ speed = 380\ MBps \\ 36.22 \sum_{a=1}^{A(n_i,n_j,p)} SLICEs(\mathrm{a},\mathrm{p}), & ICAP\ speed = 4.5\ MBps \end{cases} \tag{15}$$

Dealing with the accuracy of the proposed estimation functions, function (13) is generated based on the real reconfiguration speed that has been measured for the two presented architectures. Consequently, this speed being a fixed parameter, the obtained reconfiguration time should be exact. Only the MicroBlaze's access to the system's bus adds some uncertainties to this calculation (less that 1%). In turn, functions (14) and (15) have been obtained based on certain simplifications on FPGA's internal structure. Besides, these functions use estimated data on design size as input for the estimation of the reconfiguration time. Therefore, deviations may exist between the estimated values and the real ones. However, during the tests carried out in the multi-standard modulator that will be presented in Chapter 6, a maximum error of a 15% has been detected. Taking into account that the methodology foresees the final physical implementation of certain partitions, hence obtaining real data to be applied to the cost function, we consider this error acceptable.

3.2. Reconfiguration overhead estimation

Another factor that has to be taken into account when dealing with dynamic partial reconfiguration is the overhead that this technology introduces. In order to manage the reconfiguration process, certain logic is needed. This logic is in charge of several functions such as the control of the access port to the configuration memory of the FPGA or the control of the memories that store the partial bitstreams. Therefore, although the main benefit of partial reconfiguration is the reduction on the number of needed resources that it achieves, the resources occupied by this control logic have to be added to the cost function of every partition. This addition has no effect on the election of the best partition as a constant value is added to all the partitions, however, it is necessary for determining the maximum size of the design.

Later, in the implementations presented in Chapter 7, two different architectures for managing dynamic partial reconfiguration will be exposed. Both of them have the ICAP port as the final access point to FPGA's configuration memory, however, the logic needed to control this port is different. On the one hand a small form factor controller based on a PicoBlaze processor and a finite state machine is used. This is the control logic used by R3TOS [Iturbe'10]. On the other hand, the standard controller provided by Xilinx and based on a MicroBlaze processor is also used.

Taking into account that the design flow for generating these architectures is out of the scope of rapid prototyping tools, it is not possible to estimate the number of resources they need. That is, the complete implementation (synthesis, mapping, place and routing) of the system has to be carried out in order to know the resource occupation. However, since a similar architecture is used in all the designs, the obtained results for a certain SDR can be extrapolated to any other. Consequently, the information on the above mentioned two infrastructures for partial reconfiguration, which has been obtained in the implementations of Chapter 7, will be presented here as a reference for future designs. Table 4 sums up this information.

Table 4: Overhead of the infrastructure for partial reconfiguration

	Resource utilization					Speed
Option	SLICES	Flip-Flop	LUT	BRAM	DSP	-
uBlaze ICAP cntrlr.	3074	3004	3943	33	3	4,5 MBps
R3TOS ICAP cntrlr.	1793	1157	2778	6	0	380 MBps

There is a noticeable difference both in design size and in the achievable speed. R3TOS' ICAP controller, being significantly smaller, has an access speed to the configuration memory nearly 85 times higher. Unfortunately, the PicoBlaze-based architecture (programmed in assembler to speed up the execution) is difficult to program and to tune up. In turn, the MicroBlaze-based ICAP controller is much more versatile. On the one hand, the C/C++ programming enables an easier use of the ICAP port. On the other hand, the characteristic that a 32-bit microprocessor such as MicroBlaze offers can be harnessed in order to carry out other system tasks. Unluckily, this versatility penalizes design size and extremely reduces reconfiguration speed. Nevertheless, both architectures are valid, so data is presented here so that it can be used to estimate design size and reconfiguration time.

4. ADDITIONAL REMARKS

We note that due to the complexity of Software Defined Radios and to the flexibility that FPGAs offer, the presented cost function may have considered the evaluation of other design factors, or the application of different evaluation strategies to the actual factors. Consequently, in order to better justify the presented cost function, additional remarks are detailed below:

4.1. Maximum execution frequency

Chapter 3 has already introduced the fact that the maximum clock frequency (or minimum clock period, its inverse used in the cost fucntion) is a difficult factor to estimate as it is highly dependent on the final implementation of the design. Therefore it is necessary to carry out the final physical implementation of the SDR (step 6 of the design methodology - Section 3.6 of Chapter 3) in order to have access to this information. Consequently, the corresponding term in the cost function has to be eliminated or set equal to zero when the cost function is used with estimated data. The complete cost function will be used once certain number of partitions have been physically implemented.

We would like to note that limits for, on the one hand, maximum clock frequency and, on the other hand, design size and reconfiguration times have a priori different natures. Design size and reconfiguration time are factors that are usually constrained by an upper limit. That is, any design with values below the limit is acceptable. In turn, the maximum clock frequency is usually a fixed factor. That is, the design has to be able to work at the established frequency, but even if it can work at a higher frequency, the design is clocked at the predefined one. This fact initially faces with the performance evaluation intended with the cost function and could suggest the removal of the maximum clock frequency from it. However, we consider that this limitation only happens in stand-alone systems where a continuous and fixed data throughput is needed. Novel architectures

including FPGAs such as R3TOS (presented below in Section 3 of Chapter 7), or architectures where the FPGA acts as a co-processor, require data to be processed as fast as possible. Consequently, a partition with a higher maximum clock frequency could be clocked to a higher clock (e.g. harnessing the sample time parameterization presented in Section 4.4.2 of Chapter 4) hence achieving a faster processing. Definitively, the maximum clock frequency being one of the factors affected by the use of dynamic partial reconfiguration and rapid prototyping tools, and regarding this new point of view, its inclusion in the cost function is completely justified.

On this basis, the check on the maximum clock frequency presented in Section 2.5 has to be completed with the set of appropriate timing constrains in the synthesis tools. This way, on the one hand a double check is performed on the lower limit acceptable by this factor and on the other hand the synthesis tool is aware of the target frequency and can optimize the implementation accordingly.

4.2. Power consumption evaluation in the cost function

Power consumption has been one of the traditional drawbacks of FPGAs compared to other devices such as DSPs or ASICs. Moreover, during the state of the art presented in Chapter 2, the reduction of the power consumption has been introduced as one of the benefits that the use of dynamic partial reconfiguration offers. Consequently, the inclusion of this design factor in the cost function seems a reasonable proposal. Unfortunately, collecting data on this factor, both in a real and in an estimated way is a complicated task. In order to obtain real data on power consumption, the design has to be tested on the target FPGA and certain infrastructure has had to be prepared on this purpose. Definitively, it is a complex and time consuming procedure so as to carry out for every possible partition. Dealing with estimated data, synthesis tools usually provide a power estimation tool. Unfortunately, in order to be used, the complete physical implementation flow needs to be executed. Furthermore, at the best of author's knowledge, in order to obtain an acceptable accuracy, an exhaustive work on providing appropriate stimulus to the system has to be carried out. In conclusion, the generation of estimated data on power consumption is also a time consuming task.

In turn, it should be noted that, in general terms, the power consumption of a certain design is related to the complexity of that design. Besides, the more complex a design is the more resources it is supposed to consume. That is, broadly, a bigger design will have higher power consumption that a smaller one. On this basis, we assume that the design size evaluated in the cost function also serves to represent in relative terms the power consumption of a partition.

Nonetheless, the inclusion of the power consumption into the cost function is considered as future work provided the power estimation tools speed up and ease their operation.

4.3. Reconfigurable functions remaining static

Section 2.2 has introduced a new concept that had not been addressed up to now: reconfigurable functions remaining static. During the design methodology definition, the functions have been presented as "reconfigurable" if they were not common to any waveform and therefore had to be

implemented in a reconfigurable area or "common" if they were shared between several waveforms. These "common" functions could then be either parameterized and implemented statically, or implemented in a reconfigurable area. This classification is valid for a simple SDR with just two waveforms. However, as SDR complexity increases and more waveforms mean to be implemented some special cases appear.

In any SDR with at least 3 waveforms, a certain function may be common to two of the waveforms but not be part of the third one. In this case, the function has to be implemented in a reconfigurable area as it has to be removed from the SDR in certain configurations. However, if a transition happens between the two waveforms which the function belongs to, it may not be necessary to reconfigure it. That is, in the particular case of this transition, the function may have been parameterized, hence being considered as "static" (although it is implemented in a reconfigurable area). Consequently, during the execution of step 3 of the design methodology - Establishment of the common parts- the function is considered as "common". Therefore the design flow for this type of function has to be followed no matter if it will not be implemented in a purely static region. In turn, when the cost analysis has to be carried out, in particular when the reconfiguration time has to be evaluated, the aforementioned especial case has to be distinguished. Although the function is implemented in a reconfigurable area, if a certain transition between waveforms does not need the function to be reconfigured, no reconfiguration time has to be added as presented in function (13).

5. SUMMARY

This chapter has presented the procedures that the proposed design methodology foresees in order to quantitatively evaluate the different partitions generated along it. The key point of this evaluation is a cost function that calculates the relative cost of each partition's size, reconfiguration time and minimum clock period. Besides, in order to enable the use of the cost function prior to the final implementation of the partitions, several estimation functions have been presented. These functions generate data needed by the general cost function based on the available information. The chapter concludes with a discussion on several issues related with the factors present in the cost function. This discussion intends to support the cost function definition that has been chosen.

Chapter 6

Methodology verification: Multi-standard modulator

6. Methodology verification: Multi-standard modulator

1. INTRODUCTION

Once all the steps of the design methodology have been detailed, this chapter will present a use case in which the methodology has been applied in order to achieve an optimal implementation. This practical use of the methodology is intended for a better understanding of the application of the different steps and, definitively, for validating it. The selected implementation is a multi-standard modulator implemented over an FPGA and in which the change of the standard is, of course, carried out via dynamic partial reconfiguration. The three standards selected to be implemented in the multi-standard modulator are WiFi, WiMAX and UMTS as they are three of the most used communication standards nowadays [Chechi'11]. Besides, due to the particular characteristics and use cases each of the standards has, it is not common to use more than one of them at the same time (particularly in user devices such as smart phones). Therefore, the use of FPGA dynamic partial reconfiguration to switch from one standard to another, reusing hardware resources and hence achieving a device size (and power consumption) reduction seems a promising opportunity. However, it should be taken into account the possibility of needing a standard change without an interruption in the service. That is, the reconfiguration time should also be the smaller, the better. Definitively, the use of the design methodology in order to fulfil the aforementioned requisites seems interesting.

2. METHODOLOGY APPLICATION

The following sections will present the application of all the steps that make up the complete design flow of the methodology (presented in Section 3 of Chapter 3) for the implementation of the multi-standard modulator. In order to better locate each of the steps, the corresponding subsections have been named in the same way as the ones in Chapter 3. Besides, it should be noted that a manual application of the methodology has been carried out, that is, the automation procedure explained to conclude Chapter 3 has not been used. This way, a better understanding of all the steps that make up the methodology is achieved; unfortunately, at the expense of a longer design time.

2.1. Analysis of the waveforms to be implemented: "Common functions/common operators" technique

This first step is initially in charge of gathering all the necessary information on the application to be implemented. Later, being the existence of common parts in the waveforms to be implemented the keystone of the methodology, it is necessary to determine the possible candidates to be reused. The two sub-sections below carry out these two tasks.

2.1.1. Multi-standard modulator and WiFi, WiMAX and UMTS overview

The selected application considers the implementation of a multi-standard SDR modulator as proof-of-concept of the design methodology. A more realistic and functional application (e.g. for the

integration in a smart-phone) would require the presence of both the modulator and the receiver. However, the implementation and especially the tuning of the reception algorithms, which are significantly more complex than the ones in transmission, would require a time consuming design process that is out of the scope of this research work. Consequently, only the transmission process is addressed. Nevertheless, the system is complex enough so as to harness the benefits of the design methodology and to validate it.

The three standards selected to be part of the multi-standard modulator (WiFi, WiMAX and UMTS) have different application ranges; therefore, it is not common to use them at the same time, so the use of dynamic partial reconfiguration is possible. Besides, they may require the possibility of changing between different standards without losing the communication link (e.g. reception of an IP-based call in the street with UMTS and entrance into a building with a better WiFi coverage), hence needing a reconfiguration time as small as possible. In the same way than the implementation of the reception algorithms, the implementation of the full version of the abovementioned standards entails an important complexity. Not being their accurate implementation one of the main objectives of the present work, simplified versions have been used. These versions contain the main data processing functions that make up the standards but do not implement all the possibilities and variations included in them. A short description of the main characteristics of the standards is presented below:

a) WiFi

WiFi [IEEE'97] is a popular technology that allows an electronic device to exchange data wirelessly over a computer network, including high-speed Internet connections. It is the trademark given to networks operating under 802.11 standards. The latest standard, IEEE80.2.11n, working at 2.4 GHz, achieves a theoretical maximum data rate of 300 Mbps and a range of about 70 meters indoor. This range rises up to 250 metres outdoors. WiFi's physical layer uses single carrier DSSS (Direct Sequence Spread Spectrum) or multi carrier OFDM (Orthogonal Frequency Division Multiplexing) technology depending on 802.11 standard's version. A high level block diagram with the main functional blocks of a typical 802.11n standard transmitter is depicted in Figure 26:

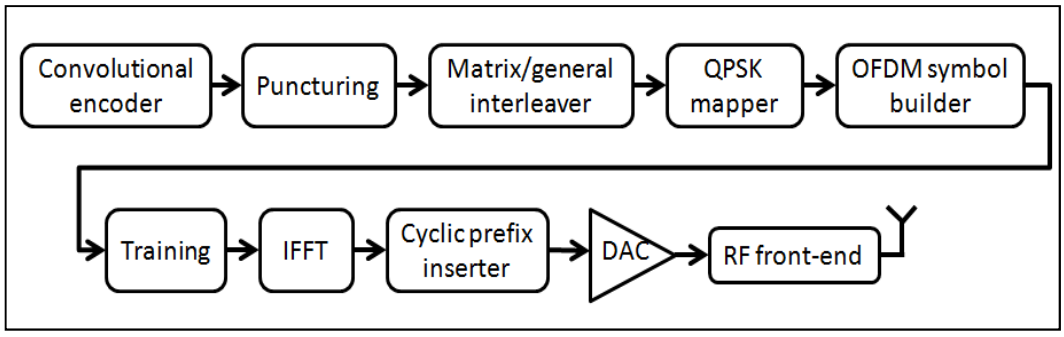


Figure 26: Functional blocks of a WiFi transmitter

b) WiMAX

WiMAX [IEEE'04] (Worldwide Interoperability for Microwave Access) is the commercial name of IEEE 802.16 family of wireless communications standards. In theory in can provide around 70 Mbps with a range of 50 Km, what gives WiMAX a significant advantage over other alternative last mile technologies like Wi-Fi. WiMAX's physical layer uses Scalable Orthogonal Frequency-Division Multiple Access (SOFDMA) transmission mode. Figure 27 shows the diverse functional blocks that make up a WiMAX modulator.

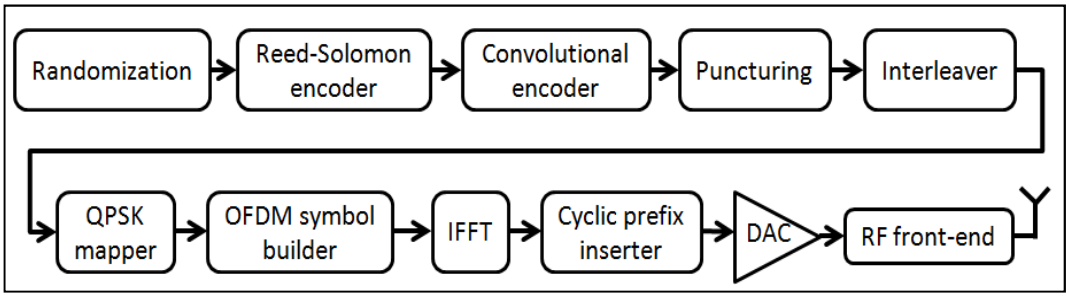


Figure 27: Functional blocks of a WiMAX transmitter

c) UMTS

Universal Mobile Telecommunications System (UMTS) [3GPP'08] is a third generation mobile cellular system for networks based on the GSM standard. UMTS supports a theoretical maximum data rate of 42 Mbps and achieves a range of tenths of kilometres. UMTS uses Code Division Multiple Access (CDMA) as channel access method. The functional blocks present in the simplified UMTS transmitter used in this implementation can be observed in Figure 28:

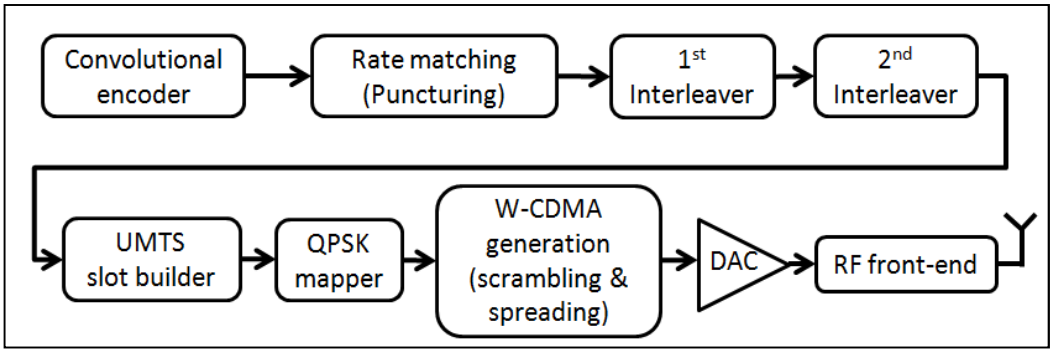


Figure 28: Functional blocks of a UMTS transmitter

2.1.2. Application of the "Common functions/common operators'" technique

Once this brief description of the three standards to be implemented in the multi-standard modulator has been presented, the "common functions/common operators" technique can be applied. We would like to remind the reader that an adapted version of the original technique is used in the

proposed design methodology. This version uses only those characteristics of the original technique that fit within the proposed SDR design framework. Namely: the graphical modelling of the application and the concept of different implementation possibilities for each function. Besides, it should be noted that the objective of this technique is to detect the function commonalities between the different standards in order to later reuse them.

a) Graph model of the multi-standard modulator

As it was presented in Chapter 4, the first step in the application of the "common functions/common operators" technique is the graphical modelling of the application. Figure 29 below shows the generated graphic for the multi-standard modulator. Data on Figure 26, Figure 27 and Figure 28 has been harnessed in order to generate the first levels of the graph. In turn knowledge on the available blocks in System Generator has been used to breakdown, if possible, each function into its lower level operators hence continuing with the generation of the graph.

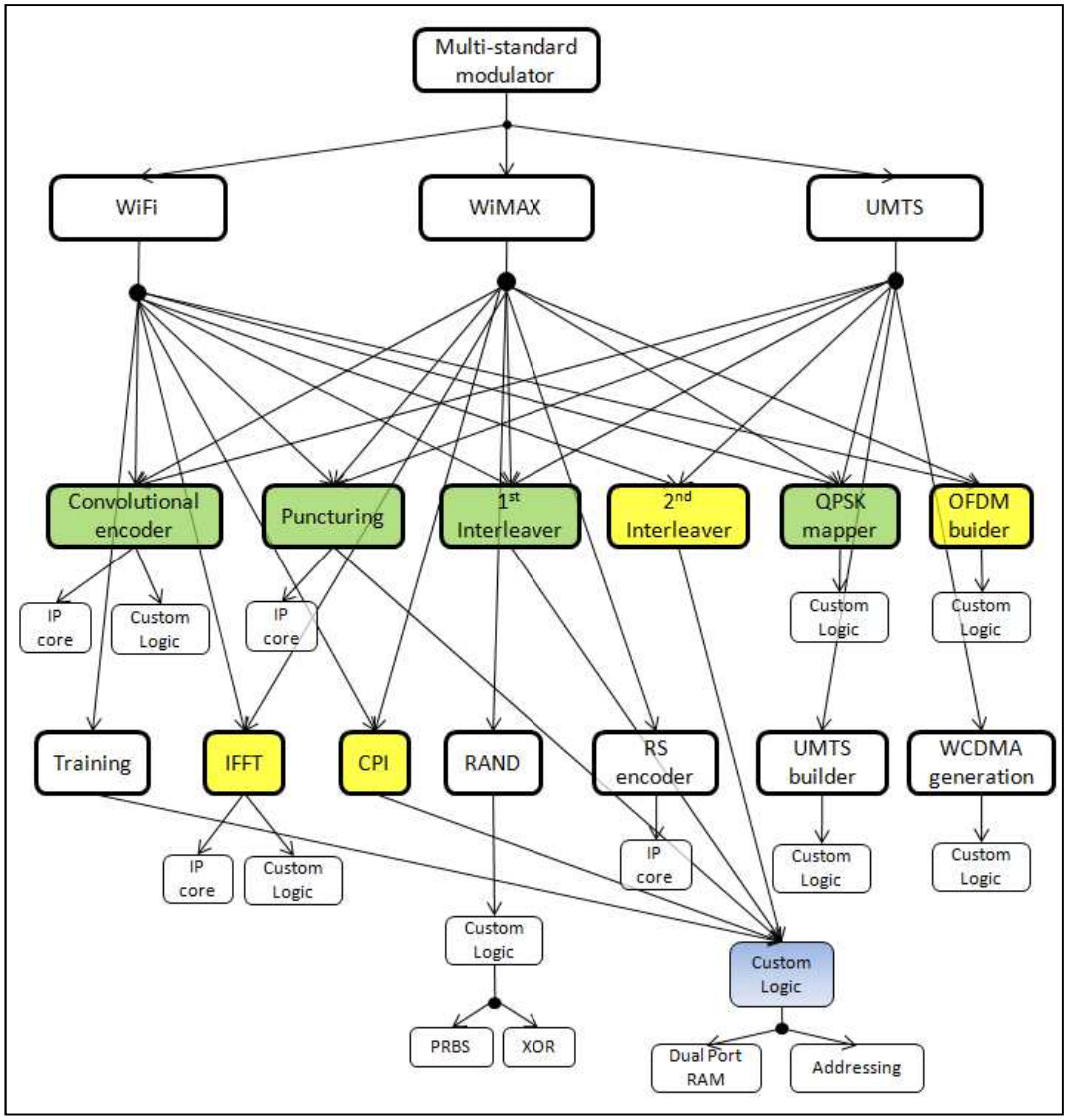


Figure 29: Graph model of the multi-standard modulator

Please note that it has been necessary to reduce and simplify certain parts of the graph in order to make it fit in the figure. Even so, the graph may still result confusing. All the first level functions (e.g. Puncturing, Interleaver... etc., identified by a thick border in the block) are present in the graph although divided into two rows. However, second level functions (e.g. IP core, dual port RAM... etc., identified by a thin border in the corresponding block) have not been further developed. It can be observed that many top level functions have a second level function labelled "Custom logic". This block represents all the sub-functions that make up the parent function. In the original graph that has been used in the development of this implementation, the "Custom logic" blocks are later divided into these sub-functions. An example of this third level granularity has been left on purpose in the Randomization function (labelled "RAND") and in the custom logic (coloured in blue) made up of a dual port RAM and an addressing function that is shared between several top level function. This same structure is used in all the "Custom logic" blocks in the complete graph. Besides, certain function names have been replaced by their acronym in order to reduce its size. Namely: Cyclic Prefix Inserter (CPI), Randomization (RAND), Reed-Solomon encoder (RS encoder) and Pseudo Random Binary Sequence (PRBS). Finally, it should be noted that the blocks labelled "IP core" represent the presence in the corresponding rapid prototyping tool (System Generator in this case) of a concrete block that carries out the parent function. For example, System Generator has a block that performs the Inverse Fast Fourier Transform (IFFT) hence it can be used for this purpose. The IP core block may also represent the use of third party designed functions although they are not present in the rapid prototyping tool and have to be added as a black box.

In order to ease the subsequent design steps, the functions in the graph have been coloured depending on whether they are common to more than one standard or not. Besides, different colours have been used depending on the number of commonalities. The functions coloured in green are common to the three standards, while the ones coloured in yellow are only common to two of them. As it was explained in Section 4.3 of Chapter 5, all these functions will be labelled as "common" in the function commonality list that will be filled in the next subsection. However, those functions only common to two waveforms will be implemented in a reconfigurable area (as it is not possible to implement them statically), but with the possibility of remaining static for certain waveform transitions.

It is also necessary to explain the presence of a "Custom logic" block coloured in blue. Analysing the graph, it can be observed that the custom logic made up by a dual port RAM and an address generation logic is common for at least 5 top level functions (puncturing, two interleavers, training sequence generator and cyclic prefix inserter). According to the definition present in the original "Common functions / common operators" technique, this custom logic could be considered as a common operator. However, taking into account that our approach does not consider further action with this type of operator, this information is only useful in order to properly design the corresponding parent functions.

b) Function commonality list

Once the graph has been generated and analyzed, the function commonality list can be filled out.

Table 5: Function commonality list (characteristics)

Function Commonality List				
Nr.	Function name	Label	Waveform	Characteristics
1	Convolutional encoder	Common	WiFi	Length: 7, generation polynomials: 171 and 133
			WiMAX	Length: 7, generation polynomials: 171 and 133
			UMTS	Length: 9, generation polynomials: 557, 663 and 711
2	Puncturing	Common	WiFi	Ratio: 2/3, vector: [111001]
			WiMAX	Ratio: 3/4, vector: [1011]
			UMTS	Ratio: 2/3, vector: [110]
3	1st interleaver	Common	WiFi	Matrix interleaver. 1920 interleaving points
			WiMAX	384 interleaving points. Interlaving pattern generated through a Matlab script
			UMTS	686 interleaving points. Interlaving pattern described in 3GPP TS 25.212.
4	2nd interleaver	Common	WiFi	1920 interleaving points. Interlaving pattern generated through a Matlab script
			UMTS	420 interleaving points. Interlaving pattern described in 3GPP TS 25.212.
5	QPSK mapper	Common	All	Same QPSK mapper for all the standards
6	OFDM builder	Common	WiFi	64 carriers (48 information)
			WiMAX	256 carriers (192 information)
7	Training	Reconfigurable	WiFi	256 point training sequence
8	IFFT	Common	WiFi	64 tap IFFT
			WiMAX	256 tap IFFT
9	Cyclic Prefix Inserter	Common	WiFi	16 point cyclic prefix
			WiMAX	64 point cyclic prefix
10	Randomizer	Reconfigurable	WiMAX	Generation polynomial: x^15 + x^14 + 1
11	RS encoder	Reconfigurable	WiMAX	(n,k)=(255,239)
12	UMTS slot builder	Reconfigurable	UMTS	15 pilot insertion
13	WCDMA generation	Reconfigurable	UMTS	Spreading factor: 256. Rest of parameters described in 3GPP TS 25.212.

It should be noted that just as with the application graph, the function commonality list presented in Table 5 has been simplified in order to fit in this document. On the one hand just the basic characteristics of each function have been presented. The original function commonality list contains all the necessary information so as to implement each of the functions. On the other hand, some rows and columns of the complete list are not being displayed at this time. Columns to be filled with information on resource occupation and reconfiguration time, or the rows corresponding to the parameterized version of the functions labelled as common have been removed at this time and will be presented later in Table 7.

c) Requirement sheet

The final action to be carried out in this first step of the methodology is the filling out of the requirement sheet. This document compiles certain overall requirements applicable to the complete system such as maximum available resources (i.e. selection of the FPGA to be used), maximum acceptable reconfiguration time or maximum acceptable power consumption. Taking into account that this implementation does not aim to create a completely functional system but a proof -of-concept application of the design methodology, the establishment of some of these parameters makes no sense. Nevertheless, the characteristics of the FPGA that has been used for this implementation and a maximum acceptable reconfiguration time inferred from the video streaming application that will be presented in the next chapter have been set.

Consequently, the requirement sheet is filled out as follows (Table 6):

Table 6: Requirement sheet

Requirement sheet		
Nr.	Requirement	Description
1	Maximum available resources	- Target FPGA: XCV4SX35 - Available resources: - SLICEs: 15360 - LUTs: 30720 - Flip-Flops: 30720 - BRAMs: 192 - DSP48: 192 - IOBs: 448
2	Maximum acceptable reconfiguration time	3 ms

2.2. Establishment of common parts: Parameterization

With the first step in the design methodology completed, all the necessary information on the functions candidate to be reused is available and compiled in the function commonality list. Therefore, the second step of the methodology can be executed. This second step, based on real implementation possibilities, is in charge of determining which of these common functions are really

suitable for being shared between the three waveforms in the SDR. On this purpose, the characteristics of the functions labelled as "common" are analyzed and, if necessary, the functions are parameterized to support the change of characteristics on-the-fly. Finally the implementation (at "rapid prototyping tool level", that is, without carrying out the HDL code generation) of both, the non-parameterized and parameterized version of each function is carried out and the estimated results on occupied resources and reconfiguration time added to the function commonality list.

In the current implementation 8 functions have been labelled as "common". The conclusions obtained from the analysis of their characteristics and from the implementation possibilities for each of them are presented below:

- The "OFDM builder" function cannot be parameterized. The big existing differences both in the number of carriers and in the pilot generation would require a multiplexing based parameterization in which the two complete implementation possibilities would be implemented in parallel. Taking into account that this function is not common to the three waveforms, hence will be implemented in a reconfigurable area, this type of parameterization does not offer any type of benefit. Consequently, the "OFDM builder" will be re-labelled as "reconfigurable" and each waveform will have its own non-parameterized implementation. Figure 30 shows an example of the implementation of the OFDM builder function for WiFi:

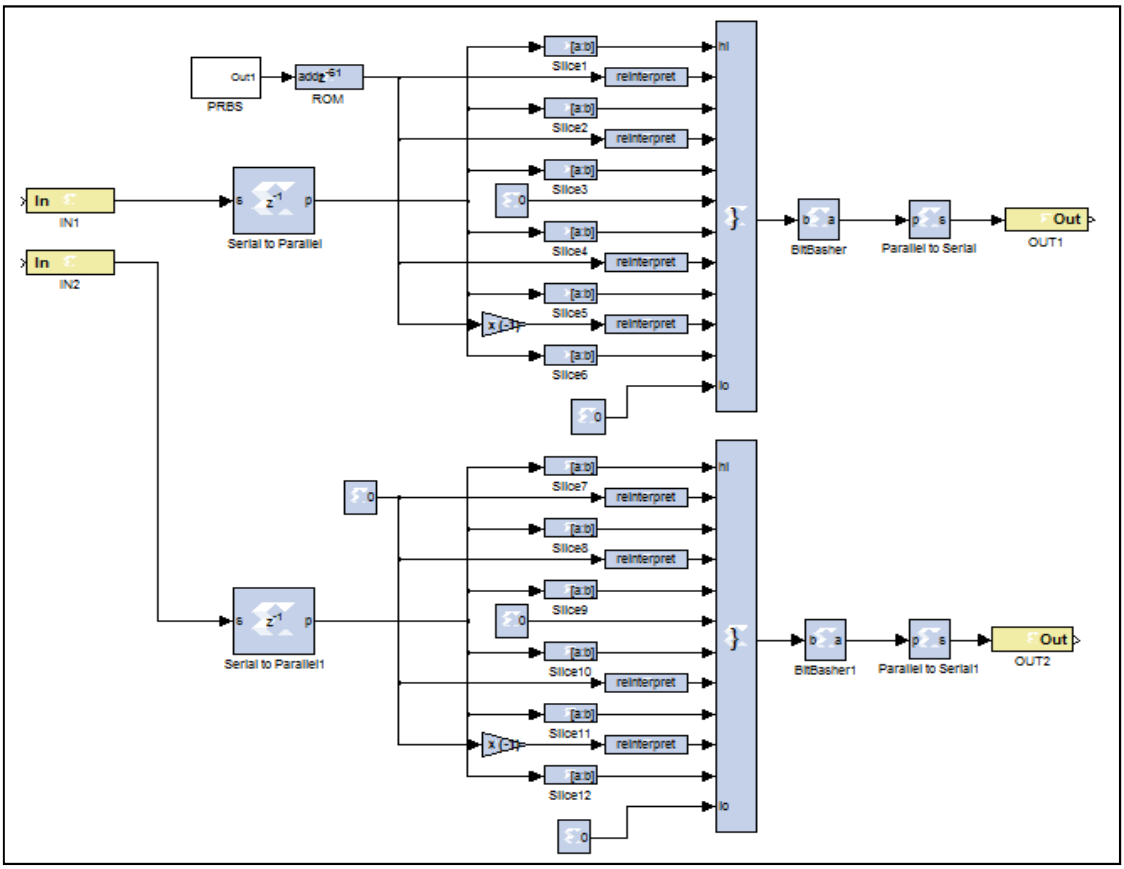


Figure 30: OFDM builder for WiFi

- In turn, the "QPSK mapper" function appears to have the same characteristics in the three standards. Therefore, no parameterization is needed and the function can be used as is, if implemented statically. As the static implementation does not generate any drawback (i.e. does not require more resources) there is no reason for not implementing the function in this way. The selected implementation can be seen in Figure 31 below:

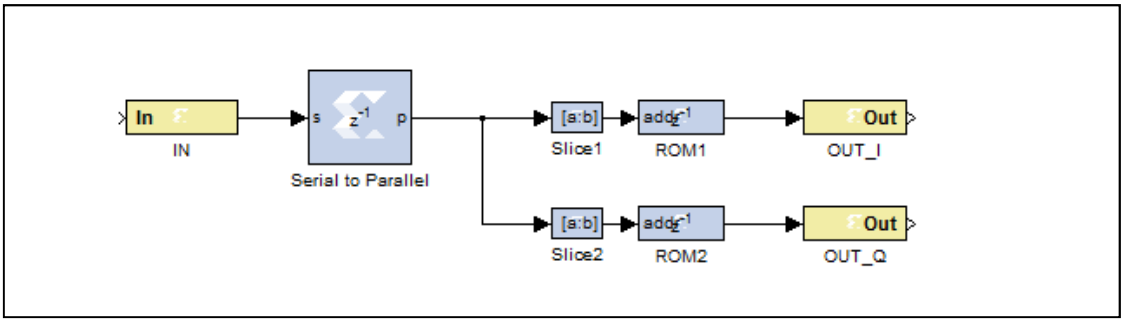


Figure 31: Implementation of the QPSK mapper

- In the graph model can be observed that the Inverse Fast Fourier Transform (IFFT) function, present both in WiFi and WiMAX, can be implemented via an IP core or with custom logic. This function requires the parameterization of the number of IFFT taps; fortunately, the IP core provided with System Generator offers this possibility. Consequently this is the selected implementation for the parameterized version of the function. In turn, the non-parameterized version just does not make use of this possibility. This type of parameterization has already been explained in Sections 4.1 and 4.3 of Chapter 4 and there, in Figure 23, an implementation example is shown. It should be noted that this function, not being common to the three standards, will be implemented in a reconfigurable area. The possibility of using the parameterized version just permits not having to completely reconfigure it when a transition from WiFi to WiMAX or viceversa is needed.

- The "convolutional encoder" function, common to the three standards, can also be implemented with an IP core or with custom logic. However, in this case, the provided IP core does not support the required on-the-fly generation polynomial change. Consequently, although the original IP core is used for the implementation of the non-reconfigurable version of the function, it has been necessary to modify it in order to add this possibility and use it in the parameterized version. The developed implementation is depicted in Figure 32. The subsystems labelled as "Poly", which carry out the implementation of the different generation polynomial, have been reused from the original IP. The input and output multiplexers have been added in order to generate the parameterized version. Please note that the characteristic of these function are equal for WiFi and WiMAX and they change in UMTS. This fact should be taken into account when generating this function's cost as any transition from WiFi to WiMAX or viceversa does not require any action.

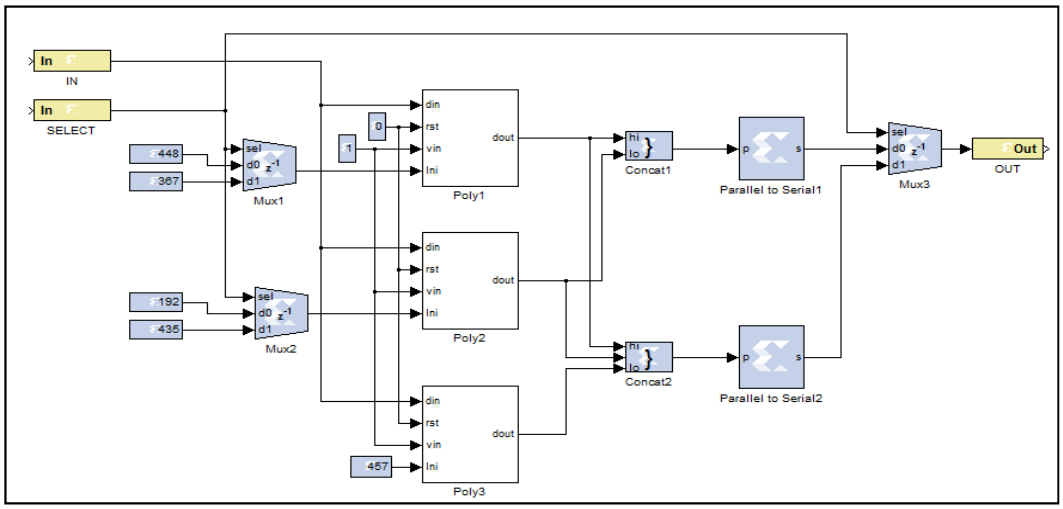


Figure 32: Implementation of the parameterized convolutional encoder

- It can be checked in the graph model that the remaining "common" functions in the function commonality list (namely: Puncturing, Interleavers and Cyclic Prefix Inserter) share the same implementation, made up of a dual port RAM and an address generating logic. All these functions require a parameterization; however, once a parameterization method has been developed for the aforementioned type of implementation, it can be replicated by all of them with minor changes. The designed parameterization, based on multiplexing, was already presented in Section 4.3 of Chapter 4. Figure 24 and Figure 25 in that chapter show an example of it applied to the "1st interleaver" function. We would like to note that although all this functions have a similar implementation, not all of them are common to the three waveforms, hence being implemented in a reconfigurable way. Consequently, the necessary precautions have to be taken when estimating the factors that describe these functions.

Once this analysis is finished, data on resource consumption and on reconfiguration time for these functions (both for parameterized and non-parameterized versions) can be estimated and added to the function commonality list. The estimation is carried out with the resource estimation block provided by System Generator and with the estimation functions presented in Section 3 of Chapter 5.

Additionally it is necessary to clarify that those parameterized functions based on multiplexing does not make use of the "Direct ICAP access parameterization" presented in Section 4.1 of Chapter 4. This technique has been developed and tested in the last part of this research work and was not available at the moment of carrying out this implementation. Its inclusion and validation in this kind of implementation will be proposed as a future work in Chapter 8. Consequently, the reconfiguration time of the parameterized version of the functions will be set to 0, disregarding at this moment the expected time needed to drive, for example from an microcontroller, the control pins of the multiplexers. Besides, the ICAP control architecture has to be also determined at this point so that the reconfigurable time is estimated in the most accurate way. For this implementation, R3TOS'

ICAP controller (presented in Section 3.2.1 of Chapter 7) is used in order to harness the real 380 MBps access bandwidth to the configuration memory that it provides.

In order not to show a figure of the function commonality list each time it is updated, please refer to the complete version of the list (Table 7) presented in the next sub-section where information on all the functions ("common" and "reconfigurable") has been filled.

2.3. Reconfigurable function implementation

A procedure, similar to the one carried out for the functions labelled as "common", has to be performed for the "reconfigurable" functions. These functions, only present in one of the waveforms, are implemented in a reconfigurable area so that they can be removed when the other waveforms are configured. The function commonality list shows that 5 functions have been labelled in this way. Therefore, the implementation of these functions has been carried out with System Generator and the corresponding information added to the function commonality list.

A brief description on the implementation used for each of the functions is presented below:

- The training sequence inserter, named "training" in the graph model, is also implemented with the aforementioned custom logic made up of a dual port RAM and an address generation logic. Consequently, no further explanation is needed as this architecture has already been presented.

- For the implementation of the Reed-Solomon encoder an IP core provided by Xilinx has been used. It should be noted that this IP core, unlike the one that implements the convolutional encoder, does not support the change, on-the-fly, of the code generation polynomials. As this function is only present in one of the waveforms this issue does not have further importance and can be used. However, in case the function had to be parameterized an alternative implementation would be necessary.

- The randomization function, as can be seen in the graph model, is made up of a Pseudo Random Bit Sequence (PSBS) generator and an XOR operator. These two sub-functions have been implemented using basic blocks available in System Generator. Figure 33 depicts this implementation.

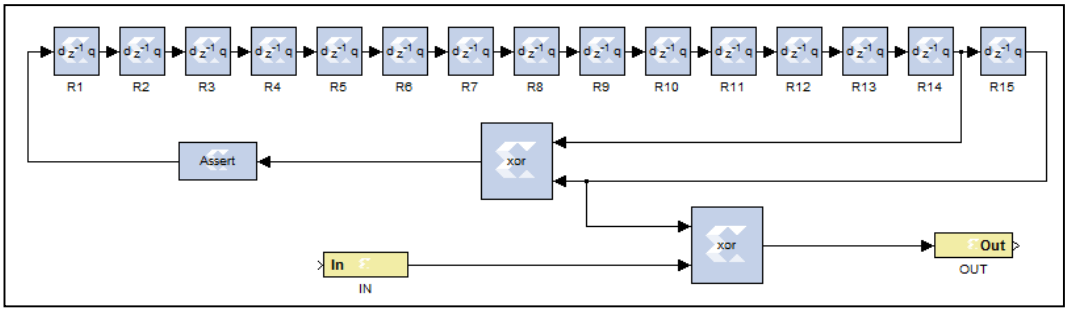


Figure 33: Implementation of the randomization function

- Finally, both the UMTS slot builder and the WCDMA generation, in charge of data scrambling and spreading in UMTS, have also been implemented using custom logic as can be seen in Figure 34 and Figure 35 respectively.

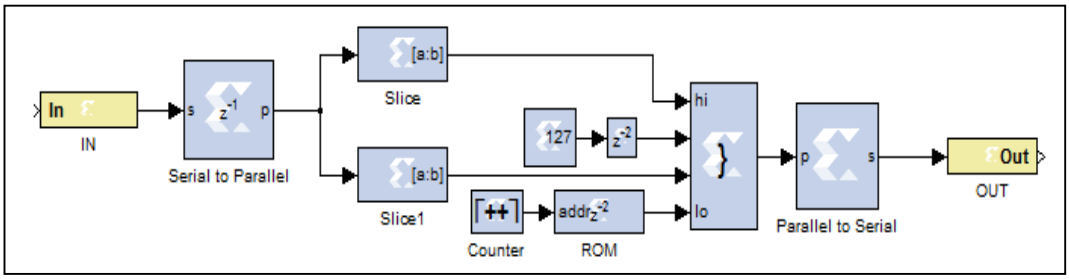


Figure 34: Implementation of the UMTS slot builder function

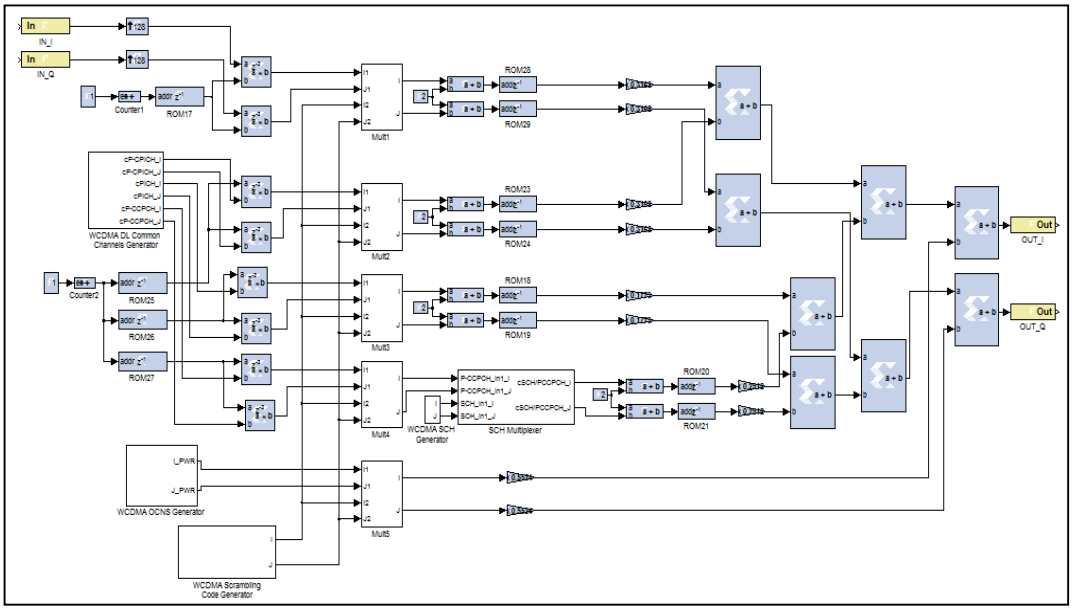


Figure 35: Implementation of the WCDMA generation

Once all the functions have been implemented, the information on resource utilization and on reconfiguration time is estimated and added to the function commonality list. Table 7 shows this list. Please note that the column containing each function's characteristics has been removed in order to reduce table's size.

Table 7: Function commonality list (estimated resources and reconfiguration time)

Function Commonality List									
Nr.	Function name	Label	Waveform	Resources					Reconf. Time (us)
				SLICE	LUT	FLIP-FLOP	BRAM	DSP48	
1	Convolutional encoder	Common	WiFi	39	13	59	0	0	16,73
			WiMAX	39	13	59	0	0	16,73
			UMTS	42	14	62	0	0	18,02
			Parameterizable	44	21	68	0	0	0
2	Puncturing	Common	WiFi	32	36	32	2	0	13,73
			WiMAX	32	26	36	2	0	13,73
			UMTS	30	28	32	2	0	12,87
			Parameterizable	34	36	34	2	0	0
3	1st interleaver	Common	WiFi	1192	1419	1406	4	0	511,37
			WiMAX	280	300	304	2	0	120,12
			UMTS	489	502	506	2	0	209,78
			Parameterizable	1294	1586	1380	7	0	0
4	2nd interleaver	Common	WiFi	1192	1419	1406	4	0	511,37
			UMTS	268	277	282	2	0	114,97
			Parameterizable	1294	1586	1380	7	0	555,13
5	QPSK mapper	Common	All	169	168	173	2	0	0
6	OFDM builder	Reconfigurable	WiFi	509	448	989	1	0	218,36
			WiMAX	1946	1739	3865	1	0	834,83
7	Training	Reconfigurable	WiFi	1138	2116	2142	10	0	488,20
8	IFFT	Common	WiFi	1420	2003	1706	3	26	609,18
			WiMAX	2394	3062	3440	2	46	1027,03
			Parameterizable	2896	3799	3529	3	46	1242,38
9	Cyclic Prefix Inserter	Common	WiFi	148	134	206	3	0	63,49
			WiMAX	466	460	558	3	0	199,91
			Parameterizable	570	740	344	5	0	244,53
10	Randomizer	Reconfigurable	WiMAX	34	4	30	0	0	14,59
11	RS encoder	Reconfigurable	WiMAX	153	71	51	0	0	65,64
12	UMTS slot builder	Reconfigurable	UMTS	65	73	127	1	0	27,89
13	WCDMA generation	Reconfigurable	UMTS	3011	3998	3149	74	67	1291,72
-	ICAP controller	-	-	1793	2778	1157	6	0	0

It can be observed that, apart from the columns that hold the information on resource consumption and reconfiguration time, a row for the parameterized version of the "common" functions has been added to the list. It should be noted that the reconfiguration time of the parameterized version of those functions common to the three waveforms is zero as the function is implemented statically. In turn, if the function is only common to two waveforms, there is a reconfiguration time, as the function is implemented in a reconfigurable area even though it is not reconfigured for certain waveform transitions. Besides, functions 5 and 6 also represent special cases. The QPSK mapper, being equal in the three waveforms, does not need a parameterized version. Moreover, it will be implemented statically; therefore the reconfiguration time is zero. Regarding the OFDM builder, it was determined in Section 2.2 that this function could not be parameterized. Consequently, only the non-parameterized versions of this function appear in the list. Finally, although it is not a function involved in the methodology, information on the ICAP controller has also been added to the list.

2.4. Design partitioning (granularity selection)

At this point, the design methodology is in the state that Figure 16 (presented in Chapter 3) describes. Consequently, the optimization loop made up of steps 4, 5 and 6 can start. As it was

explained in the presentation of the design methodology, it is possible to carry out this process both with estimated or with real data from the waveforms. In order this process to be as fast as possible, initially, the estimated data that has been compiled in the function commonality list will be used in the cost function. Later, once certain partitions have been chosen as optimal, their final physical implementation will be carried out and the best partition will be selected based on the evaluation of real data.

The first step in this optimization loop, (i.e. step 4 of the design methodology) is in charge of generating the design partitioning. In the current design, three functions (Convolutional encoder (function number 1), Puncturing (2) and 1st interleaver (3)) can be implemented either statically (using the parameterized version of the function) or in a reconfigurable way (using the waveform-linked non-parameterized versions of the function). Besides, three more functions (2nd interleaver (4), IFFT (8) and Cyclic Prefix Inserter (9)) although implemented in a reconfigurable area, also have two implementation possibilities. They can be implemented either using a parameterized version (hence not being necessary to reconfigure them in certain waveform transitions) or in a waveform-linked version (being reconfigured in every transition). Finally, one function (QPSK mapper (5)) can only be implemented statically and six functions (OFDM builder (6), Training (7), Randomizer (10), RS encoder (11), UMTS slot builder (12) and WCDMA generation (13)) are implemented in a reconfigurable way and only have an implementation possibility.

To sum up, there are six functions that have more than one implementation possibilities (functions 1, 2, 3, 4, 8 and 9 in the function commonality list), consequently 2^6 = 64 different partitions can be generated. Table 8 shows all the possibilities:

Table 8: Partition table

Partition table													
	Function												
Nr	1	2	3	4	5	6	7	8	9	10	11	12	13
1	S	S	S	RP	S	R	R	RP	RP	R	R	R	R
2	S	S	S	RP	S	R	R	RP	RNP	R	R	R	R
3	S	S	S	RP	S	R	R	RNP	RP	R	R	R	R
4	S	S	S	RP	S	R	R	RNP	RNP	R	R	R	R
5	S	S	S	RNP	S	R	R	RP	RP	R	R	R	R
6	S	S	S	RNP	S	R	R	RP	RNP	R	R	R	R
7	S	S	S	RNP	S	R	R	RNP	RP	R	R	R	R
8	S	S	S	RNP	S	R	R	RNP	RNP	R	R	R	R
9	S	S	R	RP	S	R	R	RP	RP	R	R	R	R
10	S	S	R	RP	S	R	R	RP	RNP	R	R	R	R
11	S	S	R	RP	S	R	R	RNP	RP	R	R	R	R
12	S	S	R	RP	S	R	R	RNP	RNP	R	R	R	R
13	S	S	R	RNP	S	R	R	RP	RP	R	R	R	R
14	S	S	R	RNP	S	R	R	RP	RNP	R	R	R	R
15	S	S	R	RNP	S	R	R	RNP	RP	R	R	R	R
16	S	S	R	RNP	S	R	R	RNP	RNP	R	R	R	R
17	S	R	S	RP	S	R	R	RP	RP	R	R	R	R
18	S	R	S	RP	S	R	R	RP	RNP	R	R	R	R
19	S	R	S	RP	S	R	R	RNP	RP	R	R	R	R
20	S	R	S	RP	S	R	R	RNP	RNP	R	R	R	R
21	S	R	S	RNP	S	R	R	RP	RP	R	R	R	R
22	S	R	S	RNP	S	R	R	RP	RNP	R	R	R	R
23	S	R	S	RNP	S	R	R	RNP	RP	R	R	R	R
24	S	R	S	RNP	S	R	R	RNP	RNP	R	R	R	R
25	S	R	R	RP	S	R	R	RP	RP	R	R	R	R
26	S	R	R	RP	S	R	R	RP	RNP	R	R	R	R
27	S	R	R	RP	S	R	R	RNP	RP	R	R	R	R
28	S	R	R	RP	S	R	R	RNP	RNP	R	R	R	R
29	S	R	R	RNP	S	R	R	RP	RP	R	R	R	R
30	S	R	R	RNP	S	R	R	RP	RNP	R	R	R	R
31	S	R	R	RNP	S	R	R	RNP	RP	R	R	R	R
32	S	R	R	RNP	S	R	R	RNP	RNP	R	R	R	R

Partition table													
	Function												
Nr	1	2	3	4	5	6	7	8	9	10	11	12	13
33	R	S	S	RP	S	R	R	RP	RP	R	R	R	R
34	R	S	S	RP	S	R	R	RP	RNP	R	R	R	R
35	R	S	S	RP	S	R	R	RNP	RP	R	R	R	R
36	R	S	S	RP	S	R	R	RNP	RNP	R	R	R	R
37	R	S	S	RNP	S	R	R	RP	RP	R	R	R	R
38	R	S	S	RNP	S	R	R	RP	RNP	R	R	R	R
39	R	S	S	RNP	S	R	R	RNP	RP	R	R	R	R
40	R	S	S	RNP	S	R	R	RNP	RNP	R	R	R	R
41	R	S	R	RP	S	R	R	RP	RP	R	R	R	R
42	R	S	R	RP	S	R	R	RP	RNP	R	R	R	R
43	R	S	R	RP	S	R	R	RNP	RP	R	R	R	R
44	R	S	R	RP	S	R	R	RNP	RNP	R	R	R	R
45	R	S	R	RNP	S	R	R	RP	RP	R	R	R	R
46	R	S	R	RNP	S	R	R	RP	RNP	R	R	R	R
47	R	S	R	RNP	S	R	R	RNP	RP	R	R	R	R
48	R	S	R	RNP	S	R	R	RNP	RNP	R	R	R	R
49	R	R	S	RP	S	R	R	RP	RP	R	R	R	R
50	R	R	S	RP	S	R	R	RP	RNP	R	R	R	R
51	R	R	S	RP	S	R	R	RNP	RP	R	R	R	R
52	R	R	S	RP	S	R	R	RNP	RNP	R	R	R	R
53	R	R	S	RNP	S	R	R	RP	RP	R	R	R	R
54	R	R	S	RNP	S	R	R	RP	RNP	R	R	R	R
55	R	R	S	RNP	S	R	R	RNP	RP	R	R	R	R
56	R	R	S	RNP	S	R	R	RNP	RNP	R	R	R	R
57	R	R	R	RP	S	R	R	RP	RP	R	R	R	R
58	R	R	R	RP	S	R	R	RP	RNP	R	R	R	R
59	R	R	R	RP	S	R	R	RNP	RP	R	R	R	R
60	R	R	R	RP	S	R	R	RNP	RNP	R	R	R	R
61	R	R	R	RNP	S	R	R	RP	RP	R	R	R	R
62	R	R	R	RNP	S	R	R	RP	RNP	R	R	R	R
63	R	R	R	RNP	S	R	R	RNP	RP	R	R	R	R
64	R	R	R	RNP	S	R	R	RNP	RNP	R	R	R	R

Certain explanations are needed in order to properly understand this table:

- Function numbers 1 to 13 correspond to the numbers present in the function commonality list. Function names are avoided in order to simplify the table.

- Columns coloured in grey (which correspond to functions 5, 6, 7, 10, 11, 12 and 13) do not change their status in any partition. Therefore, they only have one implementation possibility.

- The status of those functions common to the three waveforms (functions 1 to 3) is denoted as follows:

    - S: The function is implemented statically. That is, the parameterized version of the function is used.

    - R: The function is implemented in a reconfigurable area. Therefore, the waveform-linked, non-parameterized versions of the function are used.

- The status of those functions only common to two of the waveforms (functions 4, 8 and 9) is denoted as follows:

    - RP: The function is implemented in a reconfigurable area and uses the parameterized version of the function

    - RNP: The function is also implemented in a reconfigurable area but the non-parameterized versions of the function are used.

As it was explained during the presentation of the methodology, the first partition looks for the minimum reconfiguration time. Therefore, the functions common to the three waveforms are implemented statically and the functions only common to two of them use their parameterized version. Sequentially, each function's status is changed either to their reconfigurable version or to the non-parameterized version until the last partition is reached. This partition has the minimum area at the expense of a high reconfiguration time.

2.5. Design cost analysis via cost function

Once the partition table has been generated and the status of each function in every partition is known, each partition's cost has to be evaluated. This evaluation is performed with the cost function (3) that was presented in Chapter 5. Prior to the application of the cost function some of the parameters used in it have to be defined. Those parameters are:

- N = 3. There are three different waveforms in the SDR.

- P = 64. The total number of partitions

- [α, β, γ] = [1, 1, 1]. Initially a neutral optimization strategy is used.

In order to present the obtained results as clear as possible, avoiding the use of high amounts of data, information will be presented within two sub-sections. Initially, the fulfilment of the design requirements will be checked. Afterward the cost of each partition will be displayed in a graphical way.

2.5.1. Design requirement check

The requirement sheet (Table 6) of this design gathers two design requirements: a maximum reconfiguration time of 3 ms and a maximum resource occupation linked to the available resources in the XCV4SX35 Virtex 4 FPGA from Xilinx. Figure 36 shows the resource occupation (in number of SLICEs) of each of the partitions in order to check the fulfilment of the corresponding requirement.

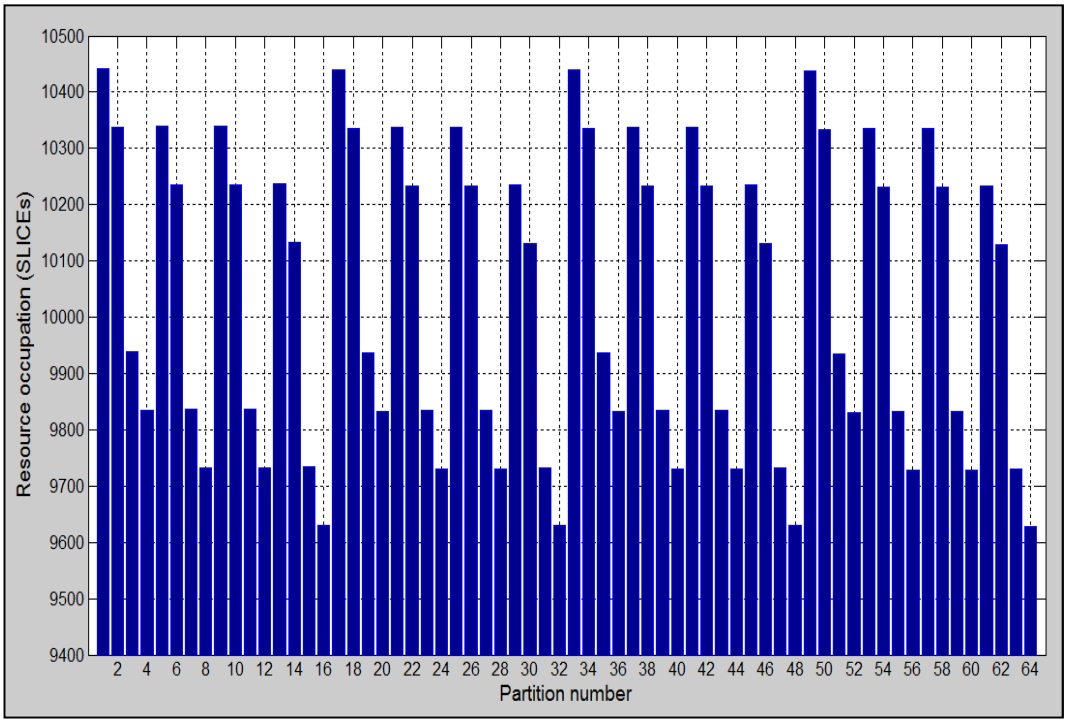


Figure 36: Data on resource occupation

We would like to note that the saw-tooth like evolution of the graphic is related to the sequential generation of the partitions. The X axis just contains the index of each partition, that is, does not represent a progressive evolution of any design parameter. Consequently, the abrupt change of the resource occupation between two close partitions corresponds to the status change (static to reconfigurable or viceversa) of several function at the same time.

Regardless of the above explanation, it can be observed that the maximum resource consumption (achieved by partition 1, in which all the functions are implemented in their parameterized version) occupies 10441 SLICES. Therefore, it is below the maximum number of available SLICES (15360). Besides, as it was explained during the presentation of the cost function in Chapter 5, apart from the maximum number of SLICEs, it is necessary to check the use of other FPGA resources such as

BRAMs or DSP48. The maximum use of these resources, 103 BRAMs and 67 DSP48s, also fits within the 192 BRAMs and 192 DSP48s available in the used FPGA.

Regarding reconfiguration time, Figure 37 shows information on this factor:

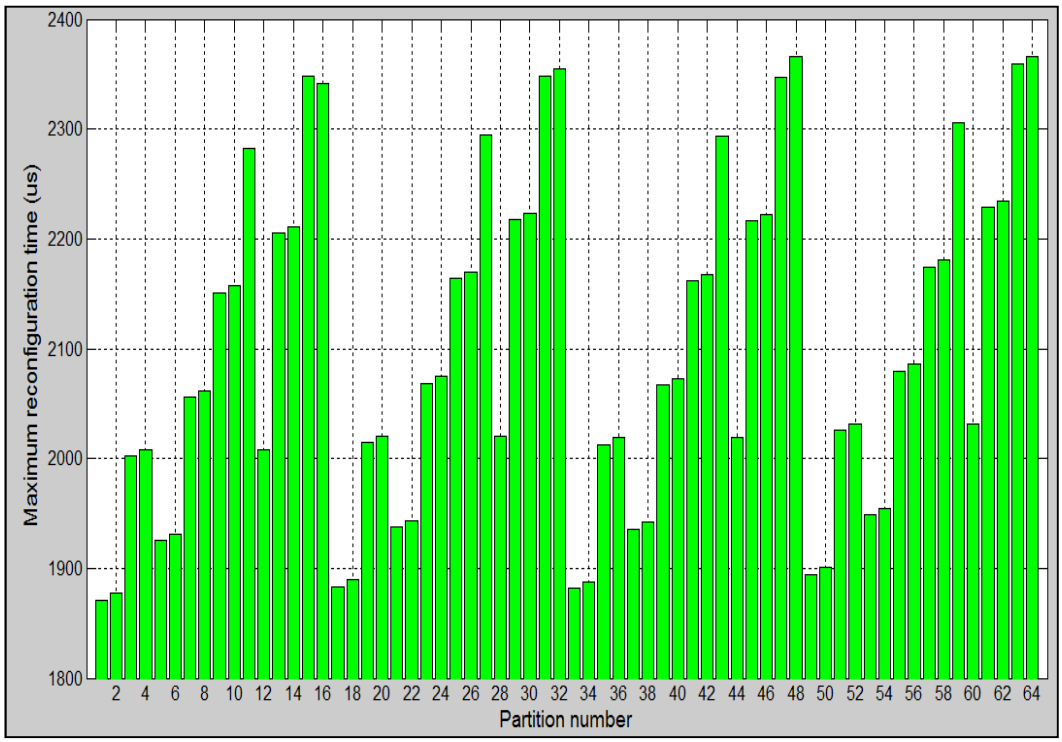


Figure 37: Data on maximum reconfiguration time

In this case the maximum reconfiguration time corresponds, as expected, to partition number 64 as it has all the functions implemented in a reconfigurable way. However, the 2.365 ms that takes to reconfigure this partition (transitioning from UMTS to WiFi as it is the worst case), are still below the 3 ms that have been set as maximum acceptable reconfiguration time. Please note that the saw-tooth like evolution of this factor is also related with the aforementioned procedure for the generation of the different partitions.

I conclusion, all the generated partitions meet the design requirements that have been established; hence the optimal partition can be evaluated and selected.

2.5.2. Optimality analysis

The cost of both the resource consumption and the reconfiguration time has been calculated, normalized, weighted and added in order to carry out the optimality analysis. Figure 38 (individual analysis) and Figure 39 (overall cost) show this information:

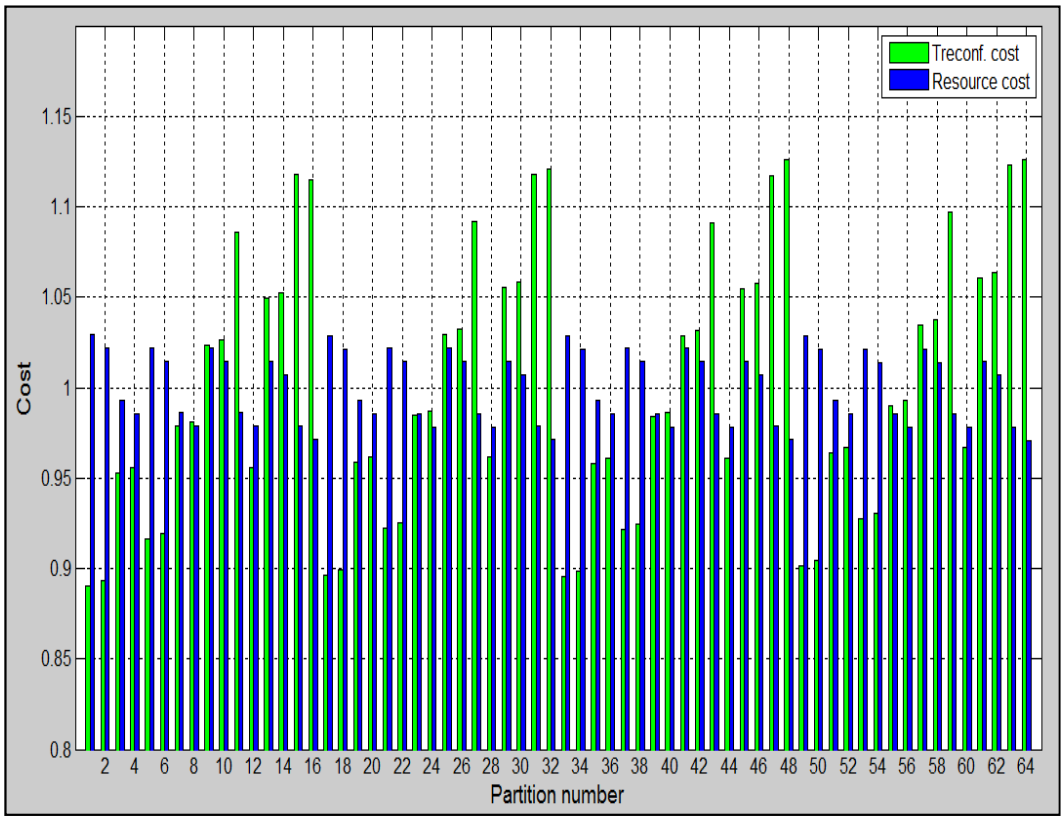


Figure 38: Individual design cost analysis (neutral)

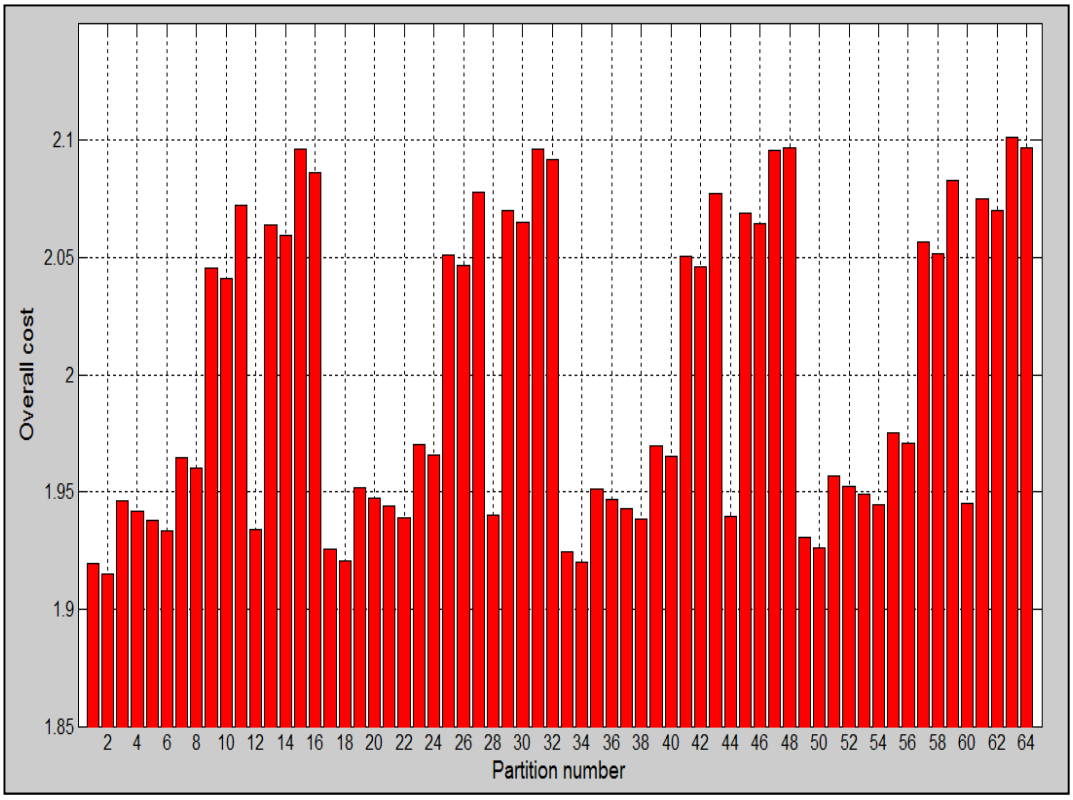


Figure 39: Overall design cost analysis (neutral)

Despite the saw-tooth like evolution of the graphics, it can be observed that in the current implementation the relative cost of the reconfiguration time is higher than the cost of the resource consumption. That is, there are bigger variations, in regard to the average value, in the reconfiguration time that in the number of occupied resources. Consequently, with the weighting parameters set to a neutral optimization goal, the most optimal partitions are those with a minimum reconfiguration time is spite of a bigger design size.

If the optimization goal is changed, e.g. using the weighting parameter set named "Design size (soft)" of Table 3 (with [α, β, γ] = [2, 1, 1]), the cost representation changes as can be seen in Figure 40 and Figure 41. This set looks for a soft optimization of the design size; therefore, the cost of the corresponding factor is penalized, making it more important in the overall cost. Consequently, with this new optimization strategy, those partitions with less resource consumption are considered more optimal.

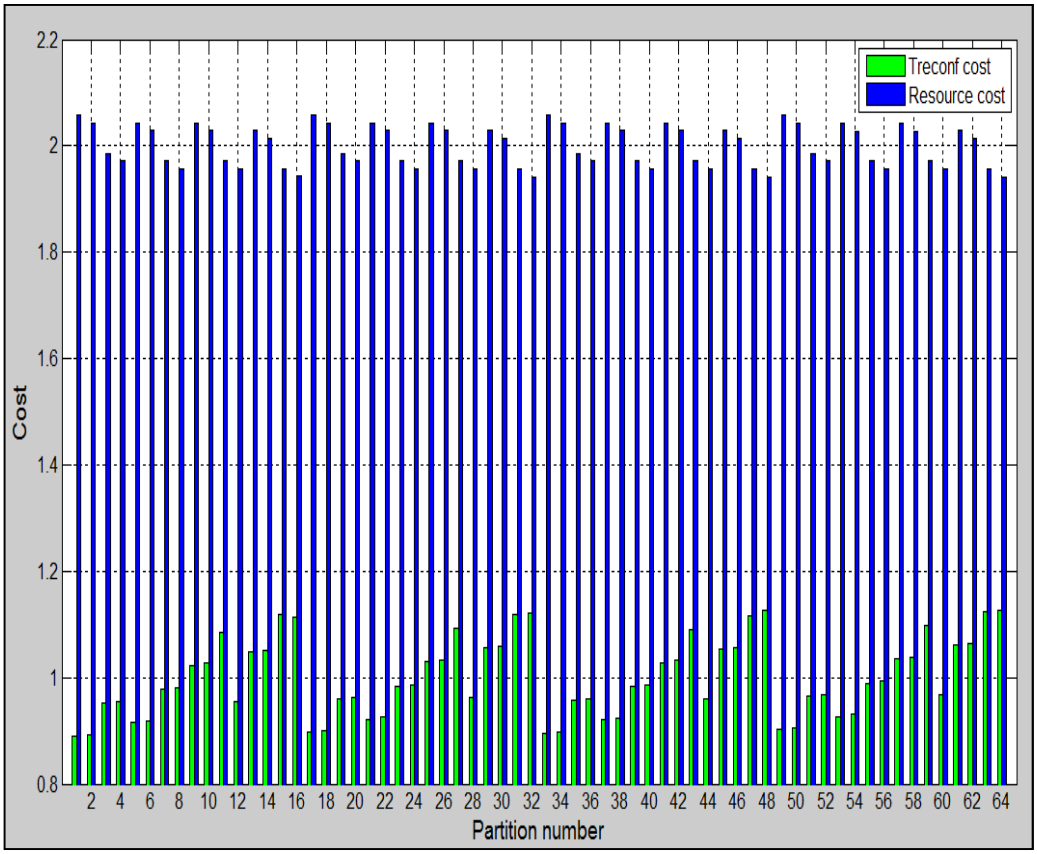


Figure 40: Individual design cost analysis (area optimization)

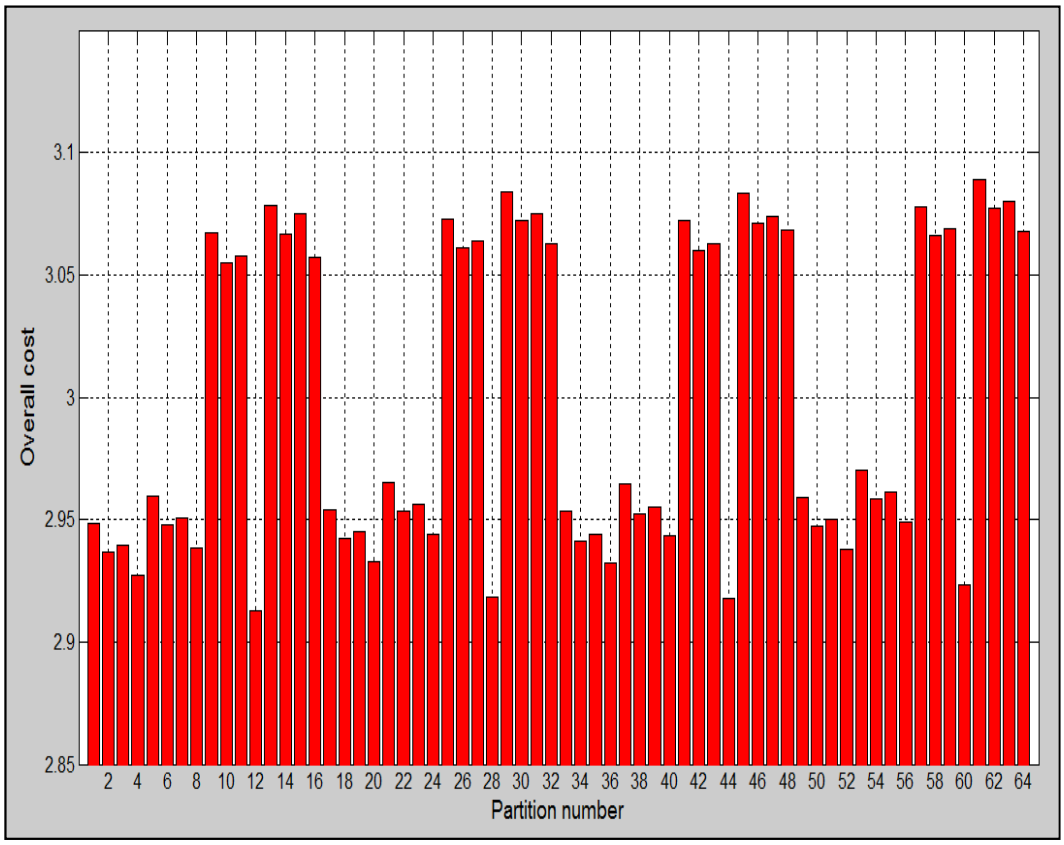


Figure 41: Overall design cost analysis (area optimization)

Considering again the neutral optimization strategy (Figure 38 and Figure 39), (there is no design requirement that recommends the use of another strategy), and based on estimated data, the optimal partition is partition number 2, with an overall cost of 1,915. However, taking into account that several partitions have a similar cost, their final physical implementation has been carried out and the real implementation data has been obtained. With this data, that includes the maximum execution frequency, the real cost of each partition can be calculated and the real optimal partition selected.

2.6. Physical implementation

The five partitions with less estimated cost have been selected for this physical implementation. They are partitions number 1, 2, 18, 33 and 34. Analyzing the implementation status of each function in these partitions, it can be observed that the selected partitions correspond to those in which the functions with higher reconfiguration times (namely: 1st and 2nd interleavers and IFFT) are implemented either statically or in their parameterized version. Taking into account the relative importance that the reconfiguration time has in this implementation, the obtained results are completely coherent.

The obtained data after the physical implementation is shown in Table 9:

Table 9: Partition data obtained from physical implementation

Partition data based on physical implementation							
	Resources (SLICEs)			Avg. Reconf. Time (us)	Fmax. (MHz)		
Partition	WiFi	WiMAX	UMTS	-	WiFi	WiMAX	UMTS
1	9793	8932	7711	1703	113	105,5	115,3
2	9293	8759	7791	1764	112,5	105,5	115,2
18	9261	8801	5811	1813	118,2	101,2	110,9
33	9705	8909	7779	1769	109,1	105,2	110
34	9392	8744	7798	1794	109,9	104,2	110,5

Highlighted values represent the worst-case configurations, that is, the values that have to be taken into account when calculating each partition's cost. Besides, it should be noted that information on both, resource occupation and maximum achievable clock frequency have been obtained from the reports that Plan Ahead generates after carrying out the implementation. In turn, the average reconfiguration time has been calculated using the estimation function (13) that was introduced in Section 3.1 of Chapter 5. This function is initially applied to the size of the generated partial bitstreams obtaining the reconfiguration time for each PRA. Afterwards, this reconfiguration times are appropriately added depending on the waveform transition to be performed, and finally averaged to obtain the presented data.

Once the cost function is applied to data on Table 9 the result below are obtained (Table 10). Please note that the maximum clock frequency present in Table 9 has been replaced by its inverse, the minimum clock period in order fit the cost function.

Table 10: Partition cost based on physical implementation

Partition cost based on physical implementation							
	Max. Resources		Avg. Reconf. Time		Max. Tclk_min		Overall cost
Partition	Absolute (SLICEs)	Cost	Absolute(us)	Cost	Absolute(ns)	Cost	-
1	9793	1,032	1703	0,963	9,48	0,989	2,983
2	9293	0,979	1764	0,998	9,48	0,989	2,966
18	9261	0,976	1813	1,025	9,88	1,031	3,032
33	9705	1,023	1769	1,000	9,51	0,991	3,014
34	9392	0,990	1794	1,014	9,60	1,001	3,005
Average	9488,8	-	1768,622	-	9,59	-	-

Highlighted in yellow the partial costs corresponding to each of the design factors (design area, reconfiguration time and minimum clock period) are presented. Besides, in the column presenting the overall cost, the optimal partition has been highlighted in green. After the final physical implementation, certain changes in the absolute data can be appreciated. This will be later addressed in Section 3. Nevertheless, partition number 2 is still the optimal one.

Regarding the application of the cost function, it should be noted that the average values that have been used correspond only to the five partitions under scope. Consequently, the obtained costs cannot be compared to the ones previously calculated with estimated data. Besides, the neutral optimization strategy ([α, β, γ] = [1, 1, 1]) has been used in order to weight each factor. With this

strategy, it can be seen that the variations in regard to the average values are quite similar in the three factors. Consequently, unlike the analysis carried out with the estimated data where reconfiguration time had a predominant position, in this case none of the factors have a significant higher weight in regard to the other two.

3. ANALYSIS OF THE OBTAINED RESULTS

Apart from the ultimate election of the optimal partition, there are certain aspects related with the presented data that have to be explained in order to obtain a better understanding of the optimization process.

It should be noted that costs obtained by the different generated partitions are relatively small. Attending to the information presented in Figure 38, where the individual design size costs are presented, it can seen that all the costs are between 0,85 and 1,15. That is, the highest variation of a factor from its average value (1) is less than a 15%. This value, which is directly proportional to the optimization capacity achievable by the design methodology, may seem quite small. The main reason for this issue is the election that has been carried out of the waveforms to be implemented. The presence of UMTS, which has a completely different implementation nature in comparison with WiFi or WiMAX, does not permit the existence of more common functions that can be implemented statically. Therefore, the optimization capacity of the methodology gets reduced. The proposed design methodology gets better results when the waveforms to be implemented have similar nature.

In regards to the presented results in Table 9, obtained from the physical implementation of certain partitions, it can be observed that they have suffered some changes in comparison with their estimated data. Table 11 details these results:

Table 11: Comparison between estimated and real data

Estimated data vs. Real data						
	Resources (SLICEs)			Avg. Reconf. Time (us)		
Partition	Estimated	Real	Error	Estimated	Real	Error
1	10441	9793	-6,6%	1871	1703	-9,9%
2	10337	9293	-11,2%	1877	1764	-6,4%
18	10335	9261	-11,6%	1889	1813	-4,2%
33	10439	9705	-7,6%	1882	1769	-6,4%
34	10335	9392	-10,0%	1888	1794	-5,3%

A reduction in both, resource utilization and reconfiguration time can be observed. As it was stated during the presentation of the methodology, implementation tools can carry out certain low grain optimization procedures such as SLICE packaging, equivalent register removal or unused logic trimming that generate these differences. However, they are smaller that a 12%, hence the estimated values are significant and can continue being used.

Finally, it should be noted that quite a small design like the one presented has generated 64 different partitions to be evaluated. Moreover, some simplifications have been necessary to present the generated data appropriately. Consequently, the use of an automation procedure like the one proposed in Section 4 of Chapter 3 is highly advisable.

4. SUMMARY

This chapter has presented a case study of the application of the proposed design methodology and design framework. All the steps within the methodology have been applied and explained in order to find the optimal implementation of a multi-standard modulator. This modulator is made up of the transmission functions of three communication standards, namely: WiFi, WiMAX and UMTS. Once the complete procedure has been carried out an optimal design has been selected. Being the reconfiguration time a predominant factor in this system, the optimal partition is the one in which the functions with higher reconfiguration times are implemented either statically or in their parameterized version. However, due to the particular characteristics of the selected standards, the obtained improvement is not as big as it could be expected. The selection of standards with different implementation natures such as WiFI and WiMAX (based in OFDM) on the one hand, and UMTS (based on spread-spectrum techniques) on the other hand, reduces methodology's efficiency.

Chapter 7

Implementations

7. Implementations

1. INTRODUCTION

In this chapter two additional implementations, carried out with the proposed design framework, are presented. These implementations, unlike the multi-standard modulator, are not completely focused on the design methodology as their reconfiguration possibilities are quite delimited. However, they provide a tangible demonstration of the feasibility of the proposed work with fully functional applications. Besides, the results and the information obtained during the design process have been a key point in the definition and development of both the design framework and the static/reconfigurable partitioning methodology. The first implementation is a small form factor cognitive video transmission system. This system is able to change its Intermediate Frequency (IF) if the transmission channel is occupied, hence achieving a secure communication. The frequency change is carried out via dynamic partial reconfiguration. In the second implementation several data coding functions, usually used in Software Defined Radios, have been designed and executed over R3TOS, a Reliable, Reconfigurable and Real-time hardware Operating System developed by the SLIG group of University of Edinburgh. This operation system harnesses dynamic partial reconfiguration to enable hardware tasks to behave like software ones.

2. SMALL FORM FACTOR COGNITIVE VIDEO TRANSMISSION SYSTEM

2.1. Introduction

The reduction, or at least the no-inclusion, of new wires is a common trend nowadays both in the consumer and in the industrial environments. Particularly in industrial environments, or in places such as airplanes, trains or vertical transport devices, where the weight and the installation ease are appreciated characteristics, the search of technologies that allow a reduction in the number of wires is an ongoing research topic. Software Defined Radios or Cognitive Radios are good candidates to carry out this task as the ability they have to get adapted to their surrounding environment allows them to work into harsh environments and perform secure communications.

On the basis of the above arguments, a cognitive video streaming transmission system has been implemented over the proposed design framework. The signal processing algorithms that take part in the system have been designed using Xilinx's rapid prototyping tool: System Generator. Besides, the frequency change that the system is able to perform is carried out via FPGA dynamic partial reconfiguration. As will be explained later, due to the simplicity of the system, it has not been necessary to apply the partitioning methodology. However, certain tests with different sizes of the reconfigurable area have been carried out in order to evaluate the relation between the size of the reconfigurable area, the size of the partial bitstreams and the reconfiguration time.

The designed system removes the RS232 wire over which a video streaming is transmitted between two computers, and substitutes it with a cognitive wireless link. The system would emulate a real-world application such as closed-circuit television (CCTV) surveillance. With the digital part fully FPGA implemented, the transmission system is made up of a modulator, an RF section and a demodulator. It implements an OQPSK modulation scheme, used in the IEEE 802.15.4 wireless standard on which other industrial wireless standards such as WirelessHART are based [HART'07]. Both the transmitter and the receiver are completed with an embedded MicroBlaze soft-processor in charge of managing the end-point communications via the RS232 wire that has been replaced and carrying out the whole system's control.

2.2. Environment description and system setup

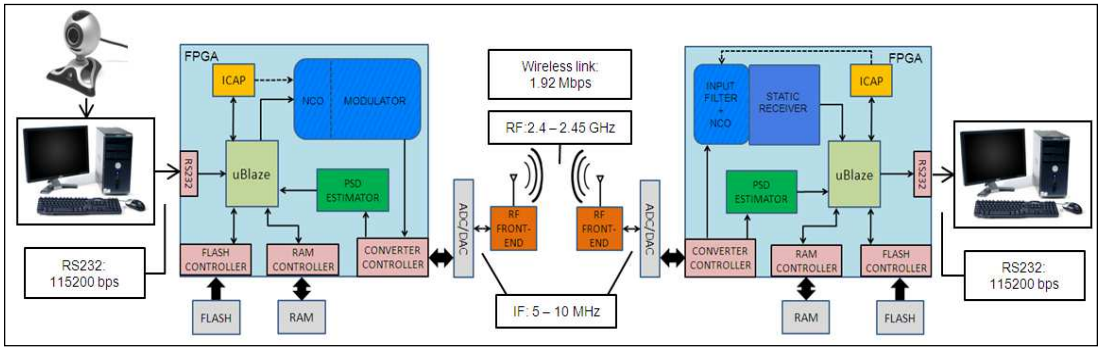


Figure 42: Cognitive video streaming transmission system

Figure 42 gives an overview of the implemented system. It is made up of:

- The end-point PCs that generate the video streaming from a Webcam, transmit and receive it through the RS232 interface and reproduce it on a screen. Further details on section 2.2.1
- The cognitive transmitter. Fully FPGA implemented, receives the video streaming from the source PC, modulates the data with an OQPSK modulation scheme and up-converts it to a 5 or 10 MHz Intermediate Frequency (IF).The choice of this frequency is made based on the availability of the transmission channel. Besides, the implementation of this frequency change is carried out via FPGA dynamic partial reconfiguration. (Section 2.3.1)
- The RF front-end. Up/Down converts the modulated signal in IF to the 2.4 GHz RF band and transmits/receives it to/from the air. (Section 2.2.3)
- The cognitive receiver. Looks for the signal in the predefined frequencies (5 or 10 MHz), reconfigures itself to the target frequency, performs a frequency and phase synchronization, demodulates the input signal and outputs data over the RS232 interface. (Section 2.3.2)

The RS232 standard is neither the typical one, nor the most suitable one for a video transmission. However, it has been chosen for this implementation due to its simplicity. Nevertheless, the proposed

implementation serves as a proof-of-concept for future developments where more complex wired transmission systems such as Ethernet could be replaced.

A description of the hardware platforms and specific software programs used for this implementation is presented below.

2.2.1. Software for streaming video over RS232

Freeware software has been used in order to transmit the video stream generated by the webcam over the RS232 interface. The **AV RS232 Sender** developed by Olds [Olds'12] uses unreal media server, live server and player [UNREAL'12] on this purpose. These programs allow the configuration of certain parameters such as the video resolution and frame rate or RS232 baud rate.

The current configuration generates a raw capture of 208x170 pixels and 3 frames-per-second from the webcam (generating an overall throughput of 828 kbps) that is later compressed, with the VC1 (WMV9) codec to 90 kbps in order to fit the maximum available rate of 115200 bauds in the AV RS232 Sender. It should be noted that both the RS232 controller present in the FPGA (921600 bauds maximum) and the radio link itself (1.92 MBps as will be seen later in section 2.3.1) are able to work at higher data rates. Unfortunately, the used software limits the transmission to the aforementioned 115200 bauds.

2.2.2. Hardware platform

Two SMT8096 boards [SUNDANCE'05] have been used to implement the digital part of the presented cognitive transmission system. This hardware platform is a PCI system, based on three main modules that can be seen on Figure 43. The SMT310Q PCI carrier board holds the SMT368 + SMT350 FPGA module with ADC/DAC converters and the SMT395 DSP module containing a TI C6416T DSP. The current implementation only uses the FPGA module and its associated ADC/DAC converters.

Figure 43: SMT8096

A Virtex 4-SX (XCV4SX35) FPGA is the core of the FPGA module. Complementing it, two 8 Mbyte banks of high speed RAM and a XCF32 Flash PROM are attached to the FPGA. This PROM is in charge of both, configuring the FPGA on start-up and storing the partial bitstreams for the dynamic partial reconfiguration.

Two 14-bit, 125 MHz sampling rate, TI ADS5500 analog-to-digital converters and a dual 16-bit, 500 MHz, DAC5686 digital-to-analog converter make up the SMT350 ADC/DAC expansion board. Additionally, a CDCM7005 programmable clock synchronizer is also present in this board providing low-jitter clock both to the ADC/DAC converters and to the FPGA.

Bearing in mind the IF and data requirements set up for the transmission system, a system clock of 61.44 MHz has been established. This clock is common to the ADC/DAC converters and to the signal processing algorithms implemented in the FPGA. Besides, a 100 MHz clock is also available in the FPGA for auxiliary tasks such as the microblaze system in charge of the RS232 interface and dynamic partial reconfiguration.

2.2.3. RF Front-end

A Radio Frequency (RF) front-end has also been designed and connected to each of the SMT8096 boards so as to achieve an over-the-air transmission. The front-end up/down converts the IF frequency output from the ADC/DAC converters to a 2.4 GHz RF frequency. It is completed with a pair of self-designed antennas that look for the best performance when working into metallic environments. In terms of frequency reconfiguration, the design and use of reconfigurable antennas, and reconfigurable matching networks and front-ends has been widely covered in the literature as it is usually a necessary characteristic. However, it is out of the scope of this work. Due to the small frequency hopping that the proposed transmissions system carries out (5 MHz wide) it can be performed by conventional elements.

2.3. Implementation

The following sections describe the implementation of the transmitter and receiver that has been carried out in System Generator.

2.3.1. Transmitter

The transmitter is divided into three main data processing tasks: the data acquisition task, the OQPSK modulator itself and the Power Spectral Density (PSD) estimator in charge of the analysis of the transmission channel's availability. Figure 44 shows the implementation of these tasks. The system is designed to achieve an overall data rate of 1.92 Mega bits per second, high enough to fulfil with the 115200 bps required by the RS232 link.

a) Data acquisition task

Taking up the bottom line of the graph, the data acquisition task receives the data from the MicroBlaze processor that monitors the whole system and prepares it for the modulation. In this step,

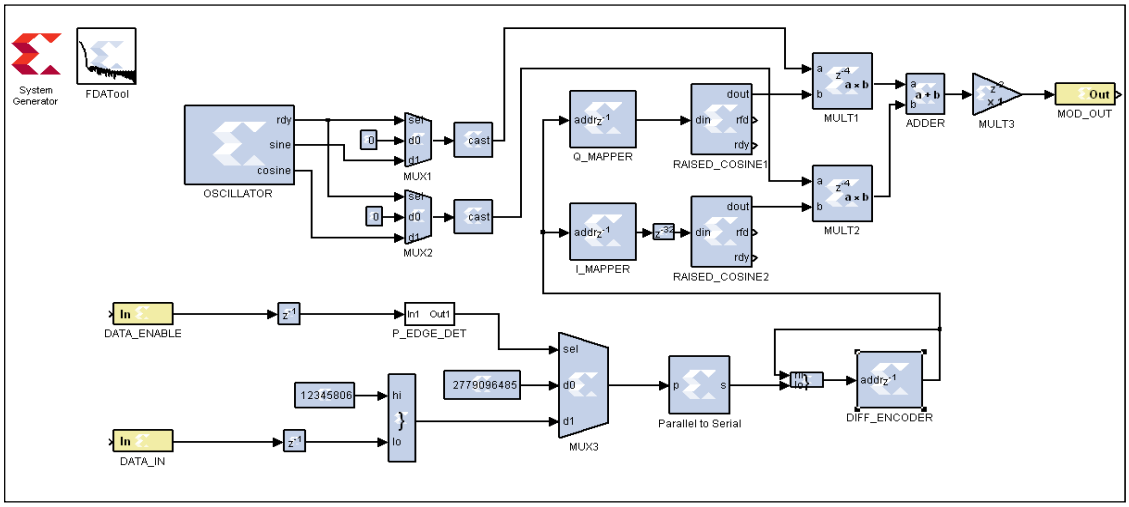


Figure 44: Data acquisition and OQPSK modulator

first of all a 3 byte header is attached to the data in order to help its demodulation in the receiver. Subsequently, data passes through a differential encoder that suppresses the phase ambiguity present in OQPSK modulation schemes.

b) OQPSK modulator

In the top line of the figure the OQPSK modulator is implemented. It is mainly composed of the numerically controlled oscillator (NCO) that generates the sine and cosine signals needed for the modulation, the mapper and the 64-tap, 0.25 roll-off raised-cosine filter that generate the I and Q baseband signals and the final multipliers and adder that perform the frequency up-conversion.

As stated before, the system is able to work in two IF frequencies depending on channel's availability, 5 or 10 MHz. An ideal cognitive design would require the transmitter to work in any IF frequency within the limits of the DAC converter, however, looking for simplicity it has been limited to the aforementioned two frequencies. The frequency change is carried out changing the oscillator's settings. With the aim of reducing the amount of resources used by the design and as a demonstration of the feasibility of this technology, this change of settings is performed using dynamic partial reconfiguration.

c) PSD Estimator

The PSD estimator shown in Figure 45 is responsible of analyzing the transmission channel. It is present both in the transmitter and the receiver and offers them the necessary information about the signal power presence in the target frequencies. It is implemented with a 64-tap Fast Fourier Transform (FFT) attached to a power detector. The FFT carries out a real time analysis of the power presence in the spectrum. It delivers a 64 byte array that represents the spectral power distribution. Bearing in mind that the ADC converters work at 61.44 MHz, the FFT is able to discern a 1 MHz approximate bandwidth. Subsequently, this data, that is received every millisecond, is averaged to reduce the background noise and passed to the power detector. In the current implementation, this

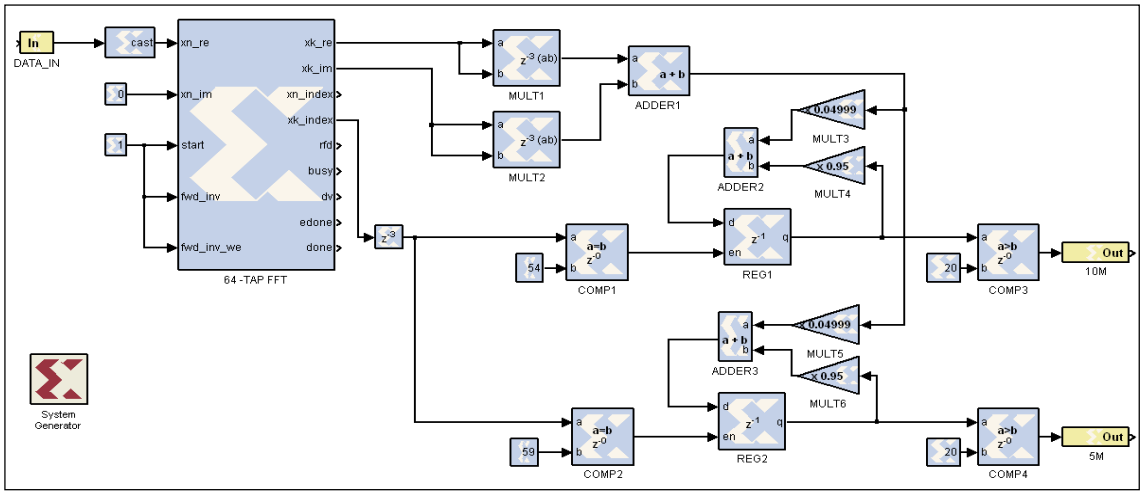


Figure 45: PSD Estimator

power detector only checks the state of the 5 and 10 MHz frequency bands, although the whole span is available. In case the power presence in any of the frequencies is higher than the maximum assumable by the communication, the corresponding signal is asserted.

2.3.2. Receiver

The cognitive OQPSK receiver carries out a power search in the available spectrum and dynamically reconfigures itself to receive the transmitted signal in the corresponding frequency. Afterwards, fine grain frequency and phase synchronization are performed and data is demodulated and submitted to the MicroBlaze that, among other tasks, converts it to the final RS232 interface. The power search is performed with the aforementioned PSD estimator, while the rest of signal processing functions are detailed below. An overview of the receiver is shown in Figure 46.

a) Costas Loop

The core of the receiver is a Costas Loop [Shah'09]. This algorithm is a digital PLL used for carrier phase recovery from suppressed-carrier modulation signals. It is made up of an oscillator, a complex multiplier, a phase detector and a loop filter. It takes up the left half of Figure 46. The algorithm tunes the oscillator until it is phase and frequency locked to the carrier signal. Besides, it also carries out the demodulation of the input signal giving out the baseband In-phase (I) and Quadrature (Q) data components.

Although the Costas Loop is able to track differences between the default frequency of the oscillator and the real incoming carrier frequency, the 5 MHz jump between the two possible IF frequencies is too large. Consequently, in order the Costas Loop to work properly it is necessary to reconfigure the default frequency of the oscillator and adjust certain filtering parameters in the complex multiplier. In the same way as the transmitter, this reconfiguration is carried out using FPGA dynamic partial reconfiguration.

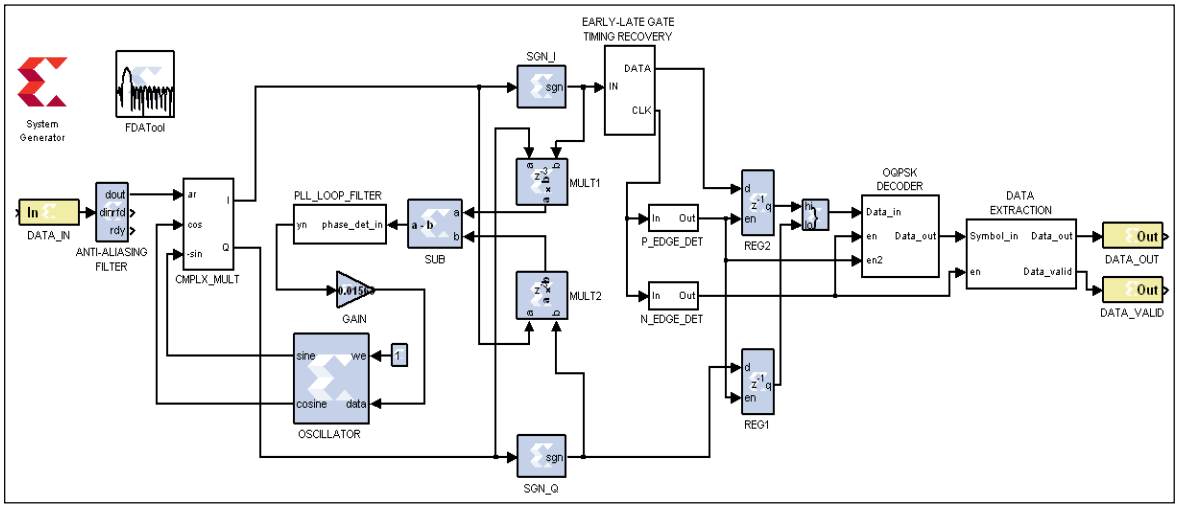


Figure 46: OQPSK demodulator

b) Early-late gate timing recovery

Due to the structure of the receiver and to the relation between the ADC sampling rate (61.44 MHz) and symbol rate (960 kilo-bauds), a 64x oversampling ratio is present in the demodulated data. Therefore, it is necessary to implement a timing recovery algorithm in order to choose the optimal sampling point. The implemented algorithm is the well known Early-late gate [Zicari'08]. This algorithm tunes the clock that synchronizes the data to its optimal point with a loop filter. The error signal that drives this loop filter is generated using samples that are early and late compared to the ideal sampling point. When the sampling point is not optimal the early and late samples are at different amplitudes, hence the error signal is generated. Once the timing recovery loop converges, the early and late samples are at equal amplitudes and the sample to be used for later processing is the sample that lies in the middle of them.

c) Data recovery

Once the received data is optimally sampled, the source data has to be recovered. As stated in the description of the transmitter, in order to solve certain problems that OQPSK modulation introduces (i.e. phase ambiguity), some data transformations have been carried out. Consequently, the receiver has to undo these transformations. The “OQPSK decoder” and “Data extraction” blocks are in charge of this operation. The first block is a differential decoder that regenerates the original data analyzing, not only the current received symbol, but also the previous one. Later, the “Data extraction” block looks for the header bytes introduced in the transmitter, removes them and outputs the original data and a data enable signal.

d) Anti-aliasing filter

Data digitalized by the ADC converter at 61.44 MHz is firstly filtered by a 25-tap anti-aliasing filter when it arrives to the receiver. This filter, which rejects the out of band power in order to improve the performance of the Costas Loop, is quite simple but requites special attention in the proposed

implementation. Due to the cognitive characteristic of the system, the pass and stop-band of the filter have to be reconfigured depending on the used IF frequency. Just in the same way as the rest of the reconfigurable components in the system, the reconfiguration of this filter is also carried out with dynamic partial reconfiguration.

2.3.3. Infrastructure for dynamic partial reconfiguration

Both the transmitter and the receiver include a MicroBlaze processor as key device for the management of dynamic partial reconfiguration. This processor is mainly connected to the ICAP port that enables access to the configuration memory, and to the controllers of the several memories that hold the partial bitstreams. Besides, other peripherals like the RS232 controller, the memory controllers and the aforementioned signal processing algorithms that carry out the radio features are also connected to this processor. Figure 47 offers a detailed view of the implemented infrastructure. It should be noted that the use of a complex microprocessor like MicroBlaze responds to the facility it offers to be programmed and debugged, hence simplifying whole system's set up. As will be discussed in section 2.4 the system could be simplified by substituting the MicroBlaze processor by a simpler architecture.

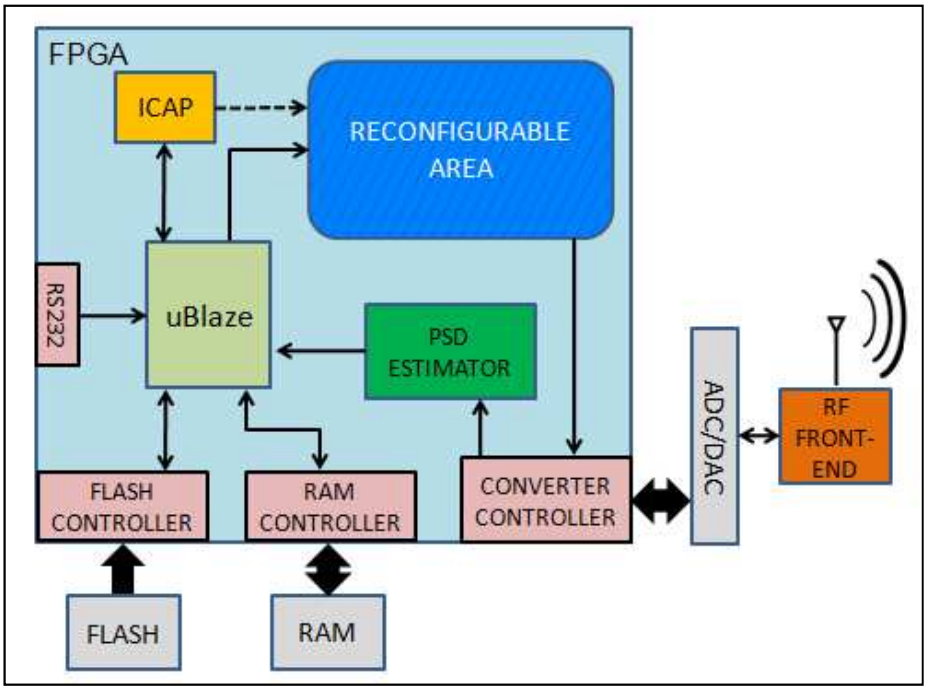


Figure 47: Detailed scheme of the internal organisation of the FPGA

When the algorithm running in the MicroBlaze decides that a frequency change is needed, the new configuration data is downloaded from one of the memories of the system to the configuration memory of the FPGA via the ICAP port. The configuration data source can be ‘any’ of the memories in the system because the storage place of these files depends on their size. When working with small partial bitstream they are stored in the FPGA’s internal BRAM memories. These limited resources provide very high speed data access so that the reconfiguration time is minimum. When

dealing with bigger partial bitstreams they have to be stored in the external RAM memory that offers a poorer performance. Nevertheless, in either case all the bitstreams are stored in the non-volatile FLASH memory on startup and later copied by MicroBlaze to their final location.

2.3.4. Software tasks

Figure 48 shows an execution diagram of the different software tasks present in the MicroBlaze processor of both the transmitter and the receiver.

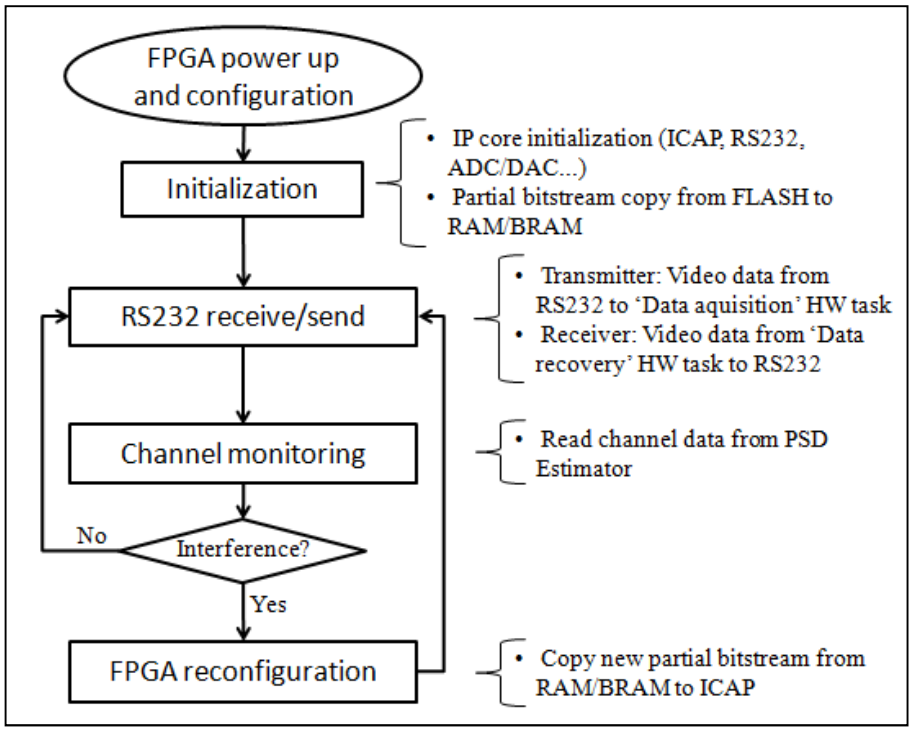


Figure 48: Software tasks' execution flow

Once the FPGA's initial configuration is automatically loaded on power-up and the MicroBlaze wakes up, the code execution starts. The initialization task first initializes all device drivers (ICAP, memory controllers, RS232, ACD/DAC controllers… and so on) and later copies the partial bitstreams from the FLASH memory to their final storage place. Then, the execution loop starts. This loop is made up of two tasks: the RS232 receive/send task (depending on whether it is the transmitter or the receiver) and the channel monitoring task. The first task is in charge of the communication between the RS232 controller and the signal processing algorithms, sending/receiving data to/from them. The channel monitoring task in turn, receives information from the PSD estimator and decides whether an IF change is necessary or not. In case it is necessary, the relevant actions in order to perform the dynamic partial reconfiguration of the device are carried out.

2.4. Measurements

This section will provide an overview of the most relevant measurements made on the proposed implementation. The design partitioning methodology has not been used in this implementation as the resources that need to be reconfigured in order to perform the IF frequency change are highly localized. However, measurements on the three main factors addressed by the methodology (resource utilization, reconfiguration time and maximum clock frequency) have been made. Besides, and in order to obtain an estimated value of the reconfiguration time that a wider functionality change would take (e.g. a constellation change), measurements on a complete reconfiguration of the transmitter have also been carried out.

Regarding the performance of the communication link, a 1.92 Mbps wireless link has been achieved. This data rate is high enough so as to properly transmit the video streaming at 115.2 Kbps generated by the webcam and the AV RS232 sender. Furthermore, avoiding the limitation of the RS232 interface, the achieved wireless bandwidth could be better utilized.

Table 12: Transmitter resource utilization

Transmitter FPGA resource utilization					
	SLICE	Flip-Flop	LUT	BRAM	DSP48
OQPSK Modulator	548 (4%)	644 (2%)	813 (3%)	3 (2%)	70 (36%)
Oscillator	51 (<1%)	61 (<1%)	79 (<1%)	1 (<1%)	0 (0%)
PSD estimator	1547 (10%)	1955 (6%)	2089 (7%)	3 (2%)	32 (17%)
uBlaze system	3074 (20%)	3004 (10%)	3943 (13%)	33 (17%)	3 (2%)
Full design	5432 (35%)	5679 (18%)	6881 (22%)	40 (21%)	105 (55%)
NR design	5531 (36%)	5772 (19%)	6988 (23%)	41 (21%)	105 (55%)

Table 13: Receiver resource utilization

Receiver FPGA resource utilization					
	SLICE	Flip-Flop	LUT	BRAM	DSP48
Filter + oscillator	2183 (14%)	3158 (10%)	1928 (6%)	1 (<1%)	41 (21%)
Static receiver	1037 (7%)	1201 (4%)	982 (3%)	1 (<1%)	3 (2%)
PSD estimator	1547 (10%)	1955 (6%)	2089 (7%)	3 (2%)	32 (17%)
uBlaze system	3074 (20%)	3004 (10%)	3943 (13%)	33 (17%)	3 (2%)
Full design	8091 (53%)	9544 (31%)	9390 (31%)	38 (20%)	79 (41%)
NR design	10324 (67%)	13224 (43%)	11365 (37%)	39 (20%)	120 (63%)

Table 12 and Table 13 sum up the FPGA resource utilization of the implemented transmitter and receiver respectively. At a first glance the transmitter occupies 5432 SLICES, while the receiver occupies 8091. These values correspond to a 35 % and 53% of the XC4VSX35 Virtex-4 FPGAs in which they have been implemented. Taking into account that this FPGA could be considered as “small”, (the biggest FPGA in the Virtex-4 family is nearly six times bigger) the small form factor and simplicity of the presented system is demonstrated. Analyzing the different parts that make up the transmission system, the MicroBlaze constitutes the biggest part of it (20% of the FPGA). This processor, mainly in charge of the management of the dynamic partial reconfiguration, could be considered as an important overhead in relation to the size of the signal processing functions.

Table 14: Reconfiguration times

Reconfiguration time			
	Partial bitstream size	Bitstream storage	
Design	-	RAM	BRAM
Transmitter (oscillator)	24 KBytes	4,5 ms	3 ms
Transmitter (whole)	267 KBytes	60 ms	-
Receiver (Filter and osc.)	763 Kbytes	171,5 ms	-

However, this overhead can be reduced or played down. On the one hand, in a more complex design the presence of a powerful processor like MicroBlaze could be exploited to execute complex software implemented tasks. On the other hand, and targeting small size designs, the management of partial reconfiguration could be ported to a more simple element like a state machine or a PicoBlaze processor hence reducing this overhead.

The bottom rows of the tables present the resource utilization of a Non-Reconfigurable (NR) implementation of the designs. In the case of the transmitter, due to its simplicity and to the only reconfiguration of the oscillator, the use of dynamic partial reconfiguration only reduces the design size by a 2%. In turn, in the receiver, this reduction reaches a 21%, what justifies its use. The above mentioned simplification of the reconfiguration management system would also improve these ratios.

In Table 14 the reconfiguration times of the different designs are shown. As can be observed, two different reconfiguration granularities have been tested for the transmitter in order to evaluate the relation between the size of the reconfigurable area, the size of the partial bitstreams and the reconfiguration time. On the one hand (named, “Transmitter (oscillator)”), only the oscillator is reconfigured in order to carry out the IF frequency change. On the other hand (named “Transmitter (whole)”), although it is not necessary, the whole OQPSK modulator is erased and reconfigured to achieve the IF change. As already explained, the measurements obtained from this configuration provide a reference on how long it would take to perform a bigger reconfiguration (i.e. a constellation change or a full communication standard change). The minimum reconfiguration time (3 ms) is achieved when only the oscillator is reconfigured and when the partial bitstream is stored in the internal BRAMs. As stated before, this bitstream location is only possible for small partial bitstreams. With this bitstream stored in a RAM memory, the reconfiguration time rises to 4.5 ms. The reconfiguration of the whole transmitter takes 60 ms and the reconfiguration of the receiver, the longest one, reaches the 171.5 ms. These values are good enough for a video streaming application but may not be sufficient for many other applications.

Calculating the reconfiguration speed from the above data, 8 MBps are achieved when the partial bitstream is stored in BRAM and 4.5 MBps when it is in RAM. This performance is very poor compared to the theoretical maximum writing speed reachable by the ICAP port in Virtex-4 devices of 400 MBps. The responsible of this performance loss is the standard ICAP controller provided with the MicroBlaze processor, which is not as optimized as possible. An enhanced implementation of this controller like the one presented within the next implementation (SDR over R3TOS - Subsection 3.2.1) would offer reconfiguration times around the millisecond.

Table 15: Achievable maximum frequency

Maximum clock frequency	
Transmitter (oscillator)	62,35 MHz
Transmitter (whole)	65,88 MHz
Receiver	69,06 MHz

Finally, Table 15 sums up the maximum achievable clock frequencies in the designs. Measurements on the receiver and on the two implementation possibilities of the transmitter have been carried out. On the one hand it should be noted that all the designs meet with the 61.44 MHz frequency requirement established for the system. On the other hand, attending to the maximum clock frequencies achieved for the two different implementation possibilities of the transmitter it can be seen that the implementation in which the whole transmitter is reconfigured obtains a higher clock frequency. In this implementation there is only a big reconfigurable area that holds the whole transmitter. Therefore, as explained in previous chapters, the restrictions for the implementation tools are softer than the ones in a design in which a small reconfigurable area inside the design fixes the routing of several signals.

3. SOFTWARE DEFINED RADIO OVER R3TOS

3.1. Introduction

Heretofore, the presented implementations have involved a straightforward, stand-alone use of the design framework formed by the FPGA rapid prototyping tools and dynamic partial reconfiguration. That is, the designed systems carry out a fully parallel, pipelined implementation of all the SDR tasks/functions that make up a concrete waveform, using dynamic partial reconfiguration only when a waveform change or an adjustment of a certain parameter is needed. However, dynamic partial reconfiguration enables the development of other implementation strategies, also compatible with SDR applications, such as proposed by the R3TOS project.

Harnessing dynamic partial reconfiguration, R3TOS, a Reliable, Reconfigurable and Real-Time Operating System [Iturbe'10], exploits FPGA resources efficiently over time, depending on functional requirements and existing resources, using classical software operating system-like support. Tasks are swapped in and out of the FPGA's reconfigurable area depending on computation requirements over the execution time. Furthermore, R3TOS is completed with several scrubbing functions that ensure safe operation even in harsh environments such as aeronautics and space [Iturbe'11a]. Considering all of the above, the use of R3TOS for the implementation of Software Defined Radios is an appealing proposition. Their joint leads to a system in which SDR takes care of a secure communication while R3TOS guarantees efficient and reliable hardware utilization.

As previously mentioned, the use of a hardware operating system in the implementation of a SDR slightly differs from the implementation strategy used up to now. The task-oriented strategy utilized

by R3TOS makes it necessary to redefine the implementation of the SDR waveforms, being necessary to partition them into former tasks, whose execution is scheduled individually. However, in the same way that the fully pipelined implementation strategy, the different granularities in which a waveform can be partitioned also generate different implementation issues that can be optimised. Although due to the nature of the hardware operating system the existence of static functions makes no sense, the use of a modified version of the proposed partitioning methodology to determine the best partitioning granularity can achieve this optimisation. The necessary modifications in the partitioning methodology, explained in sub-section 7.3.3, mainly consist of including the new design factors (e.g. execution time) in the cost function.

Therefore, this implementation is a proof-of-concept of both, the sustainability of using R3TOS in the implementation of Software Defined Radios and the possibility of harnessing the proposed partitioning methodology in the election of the best granularity for the tasks to be executed. In short, the first tasks (the data coding functions in a modulator) of a SDR have been implemented in R3TOS. Apart from the typical function design procedures, several developments have been carried out in order to achieve a fully functional implementation. The adaptation of the communication interface generated by System Generator to R3TOS communication interface and the development of a task context saving and restoration procedure are two of them.

3.2. Overview of R3TOS

The R3TOS project [Iturbe'10] aims to develop a fault-tolerant, real-time operating system for FPGAs. R3TOS puts the versatile resources embedded in a reconfigurable chip at the service of computation in a reliable way. Unlike the classical way of implementing data processing tasks on FPGAs, R3TOS enables hardware tasks to behave like software tasks. Tasks are swapped in and out of the FPGA's reconfigurable area depending on computation requirements over the execution time. This way, FPGA resources can be efficiently shared over time, which allows for the execution of very large applications on small devices, increasing performance per area and reducing power consumption. Other main characteristics of R3TOS are real-time performance thanks to a real time scheduler and fault-tolerance as tasks can be placed around emerging and permanent faults on-chip. Indeed, hardware tasks are scheduled based on their deadlines and placed on non-damaged resources, which are detected through regular scrubbing, keeping the system fault-free at all times. A simplified illustration of R3TOS is presented in Figure 49 [Iturbe'11b] [Hong'11b].

A description of the R3TOS architecture and the several processing modules that take part in it is presented below.

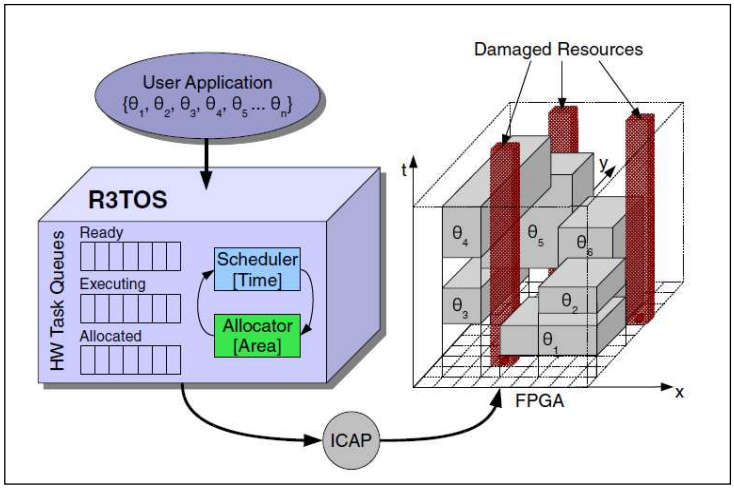


Figure 49: Scheduling, allocating and executing hardware tasks onto a partially damaged FPGA in R3TOS

3.2.1. ICAP controller

Many of the operations and possibilities R3TOS offers lie in the use of dynamic partial reconfiguration of FPGAs. R3TOS makes use of the ICAP port provided by Xilinx FPGAs as it offers the maximum configuration bandwidth (400 MBps) and in order to remove the need for external devices to control this reconfiguration procedure. Nevertheless, an efficient control of this port is necessary to harness its maximum bandwidth.

R3TOS implements a custom-made ICAP controller composed of a finite state machine and a PicoBlaze microcontroller, which offers several high level functions for the ICAP port. The basic functions are: task loading, task blanking, data feeding and collection to/from tasks, and scrubbing. All these functions are executed exploiting the maximum bandwidth of the ICAP port, accessing 32 bit words at 100 MHz. Although the final access to the ICAP port is carried out at 400 MBps, if the overhead and delays introduced by PicoBlaze and finite state machine are taken into account, the effective bandwidth obtained falls to 380 MBps.

3.2.2. R3TOS scheduler

The scheduler is the part of R3TOS in charge of deciding the order in which different hardware tasks have to be executed. It is implemented on a PicoBlaze processor that executes a port of the well-known Earliest Deadline First (EDF) scheduling algorithm. Unlike the classical implementation of this algorithm in software operating systems, where usually only a single task can be executed at a time, the R3TOS approach considers the possibility of executing tasks in a true parallel way. This way the scheduler has a higher degree of freedom to accomplish its work while the placement of the different tasks becomes more complicated. The allocator, discussed in the next sub-section, is in charge of dealing with this issue. Consequently, the scheduler shares information with the allocator as well as the processor that that issues the tasks.

Moreover, as already mentioned in the introduction, the aim of R3TOS is to make hardware tasks behave like software ones. Therefore, tasks in R3TOS meet the typical states in any operating system: *Waiting*, *Ready*, *Preempted* and *Executing,* plus two hardware-centric additional states: *Allocating* and *Allocated* [Hong'11a]. Due to concrete restrictions of partial reconfiguration, task preemption is not straightforward and needs special care as will be seen later.

3.2.3. R3TOS allocator

Once the scheduler has decided that a particular task has to be executed, it is the allocator's job to look for an appropriate site within the reconfigurable area of the FPGA. For this purpose there are three main parameters it has to take into account: the damaged sites where it is not possible to allocate any resource, the sites already under utilization by other tasks and the resources the scheduled task needs for its execution. Due to the non-homogeneous distribution of resources on modern FPGAs, e.g. DSP Blocks or BRAMs, the use of this type or resources by any task may be very restrictive and a bottleneck for the allocator's performance. Therefore this has to be taken into account when dealing with task design for R3TOS. The allocator implementation is based on a third PicoBlaze that executes the Empty Area Compaction (EAC) allocation algorithm [Hong'11b]. The algorithm gets task information from the scheduler and information about the damaged sites from the ICAP controller.

3.2.4. ICAP-based Inter-Task Communication Infrastructure (I2CI)

One of the main peculiarities of R3TOS lies in the data communication of the tasks. In order to obtain the highest area performance and a total freedom for the allocator to avoid hardware failures, the classical inter-task 'wired' communication systems have been replaced with a new inter-task communication mechanism. Indeed, systems such as Network on Chip (NoC) or point-to-point communications between tasks and host, force tasks to be placed into certain places. Furthermore, this kind of solutions requires the use of static lines that make the management of partial reconfiguration more difficult.

For this reason, R3TOS takes advantage of the capabilities of ICAP and implements a novel ICAP-based Inter-task Communication Infrastructure (I2CI) mechanism [Iturbe'11b]. In it, data stored in diverse resources across the FPGA like BRAMs or LUTs configured as RAM, is accessible for tasks that need it via ICAP. Considering this, I2CI uses ICAP to move data to/from tasks from/to other tasks or the host. In the current implementation, each task has an input and an output buffer implemented by a BRAM, and so does the MicroBlaze host interface processor. The ICAP controller is responsible for copying input data to the task from the MicroBlaze processor or other tasks, and once data is processed, copying it back. I2CI also adds some synchronization features for controlling the execution as can been seen in Figure 50. These features are based on a *Hardware-oriented Semaphore* (HWS), implemented on a LUT. This HWS acts as task reset signal before execution and as 'DONE' signal when it has finished computation.

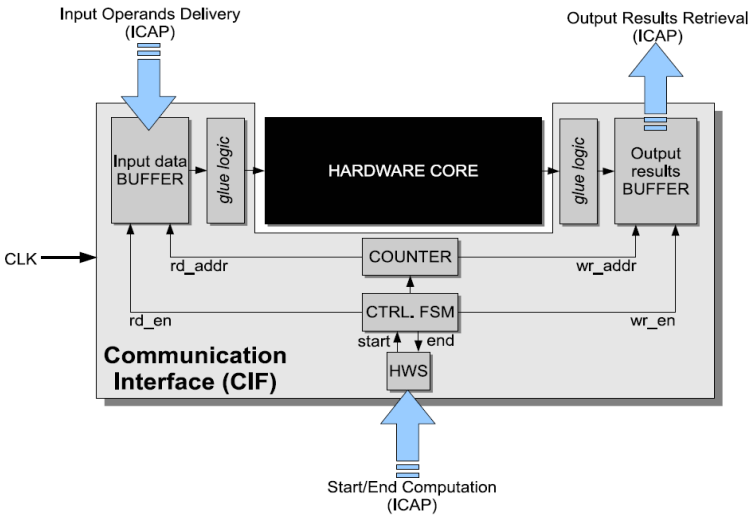


Figure 50: ICAP-based inter-task communication infrastructure detail

3.2.5. Miscellaneous

The full version of R3TOS completes with a MicroBlaze processor which is in charge of several control tasks of the whole system such as task bitstream management on start-up, and generation of the list of tasks to be executed. It also eases the connection with the external world via standard communication IPs such as Ethernet or RS232, carries out the control of the diverse memories in the system and is also available for the execution of any software task if and when needed.

Other IPs within R3TOS are: a recovery unit that can reconfigure any of the aforementioned parts of R3TOS in case of failure; a Configuration Guardian (CG) that is independent from the allocator ensuring that a task does not accidentally overwrite any existing part of the system; and a communication monitor that guards the communications between the different processors in the system and reports any malfunction.

Figure 51 shows a schematic diagram of the implementation of R3TOS in an FPGA.

The tasks present in the execution area have been drawn faded to represent their reconfigurability, that is, neither their execution, nor their physical location are fixed features. Besides, in order to ease the representation, the reconfigurable area has been drawn smaller than it really is. Finally, a simplified representation of the BRAM based I2CI communication interface can be observed both in the tasks and in the host MicroBlaze processor.

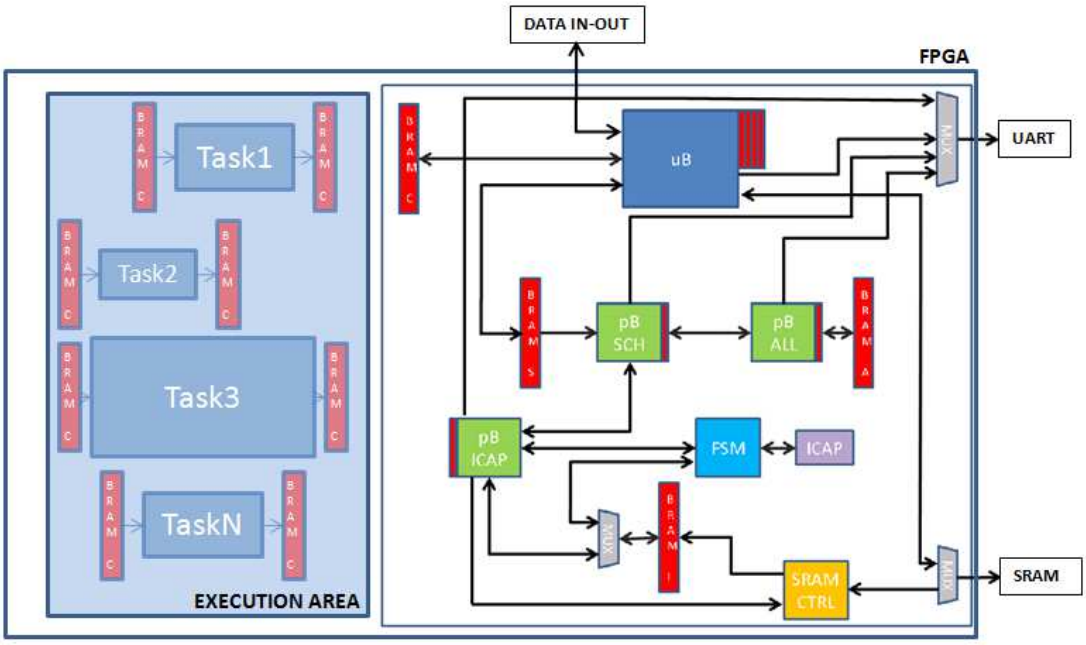


Figure 51: R3TOS implementation

3.3. Design issues in R3TOS and modified cost function

The software-like task execution proposed by R3TOS leads to the appearance of new design implications that need to be analyzed and solved if necessary. Up to this point the designs implemented in a fully pipelined and parallel way have been characterized by three main factors: reconfiguration time, FPGA resource consumption and maximum clock frequency. These factors are still important for the tasks executed in R3TOS; however, a new factor needs to be addressed: the execution time.

Parallel, pipelined architectures process data in a sequential way. In these architectures data is fed serially, and after some time, determined by system's latency, output data is generated and delivered also serially. Therefore the "execution time" concept is quite vague and is usually replaced by the measurement of system's latency and throughput. In contrast, task execution time is an important factor that highly affects real-time operating systems such as R3TOS. Consequently, although each hardware task for R3TOS is a standalone parallel, pipelined system, the execution time has to be measured and taken into account.

Due to R3TOS' particular architecture, task execution time is composed of function programming, data feeding, data processing, data collect and function unloading:

- Function loading time: This time is directly related with the reconfiguration time used up to now. It comprises the necessary time to download via ICAP the partial bitstream that contains the target task. As always, this time is dependent on task size and ICAP access speed. In case the task context has to be recovered from a previous execution (feature

explained in section 3.4.3), the time that this process takes has to be also included in function loading time.

- Data feeding time: Regarding the task communication interface proposed by R3TOS, data needs to be fed into the input buffers of I2CI prior to task execution. This operation is also carried out via ICAP so its access speed and the amount of data to be processed are the factors that determine its duration. In regards to this last factor some considerations related to the type of buffer to be used need to be explained. Due to FPGA restrictions, the use of BRAM based buffers requires the programming of the 4 BRAMs present per clock region and column. Taking into account that 64 frames (1312 bits each) are needed in order to program their content, 10.25 KBytes (1312 bits x 64 frames) have to be transferred, no matter which the size of the needed real data is. In turn, RAM LUT based buffers can be frame addressed, hence achieving a minimum data transfer of the 1312 bits (164 Bytes) that make up a frame. Besides, it should be noted that not all the transferred data is effective data. Just 4x2KBytes in the BRAMs and 64x2Bytes in the RAM LUTs can be used for processing, what leads to 78% efficiency. Figure 57 and Figure 58 in Annex 1, which have been extracted from [Iturbe'12a], show the configuration memory organization of BRAM and RAM LUTs in order to support the data above.

- Data processing time: This time is the period between the activation of the Hardware Semaphore that enables task execution and the pull in of the "DONE" indication in this same resource. This time comprises the data read from the input buffer, the pipelined data processing and the writing into the output buffer. Consequently, it depends on the clock frequency used by the task, in its achievable throughput, in the amount of data to be processed and in the latency.

- Data collecting time: In a similar way as the data feeding process, once a task execution has finished it is necessary to collect the produced data. Therefore, the factors that need to be taken into account in order to measure or estimate this time are the same as in the data feeding time. However, it should be noted that both times need not necessarily be equal as some functions may modify the data input/output ratio (e.g. a data puncturer).

- Function unloading time: If task context needs to be saved for a future execution (feature explained in section 3.4.3), the partial bitstream that describes the task has to be updated with the current state and stored in RAM. Consequently the corresponding time has to be also taken into account.

In order to harness the proposed design methodology in the election of the best task size, it is necessary to add the abovementioned execution time to the cost function. Besides, certain considerations must be taken into account in regards to the particularities of the way R3TOS executes tasks. R3TOS, being a hybrid between a hardware and a software operating system, has characteristics of both of them, namely: data dependencies among tasks and task parallelization. Consequently, at the time of calculating overall task execution time it is necessary to consider that

certain operations can be parallelized (e.g. task processing) while others are not (e.g., function programming, task data feeding/collecting or any operation in which the ICAP is involved). Moreover, dead times waiting for a certain data to arrive have to be also measured.

Consequently, the modified version of the cost function for evaluating a certain task partition "p" is described as follows:

$$Cost(p) = \alpha \cdot \frac{Resources_{Max.}(p)}{\overline{Resources_{Max.}}} + \delta \cdot \frac{T_{EXEC.}(p)}{\overline{T_{EXEC.}}} \tag{16}$$

At first sight it can be observed that the main difference with the former cost function (3) is the presence of only two terms in this new function: the first term evaluating the design size (which has been maintained from the original cost function) and a second term evaluating execution time. The above presented definition of execution time has shown that both, task reconfiguration time and minimum clock period, that is, the two terms that are not present in R3TOS' cost function, have influence on task execution time. Consequently, these two terms have been replaced by a new one just evaluating overall execution time. Please note that this overall execution time refers to the required time to execute all the tasks of a certain application. Similarly to any other term present in the cost function, the execution time is normalized so that it can be properly added to the rest of the terms. Besides, it also has the weighting parameter δ in order to weight the term if necessary.

Sub-function (17) details the computation of the execution time:

$$T_{EXEC.}(p) = \sum_{t=1}^{T(p)} [T_{LOAD}(p,t) + T_{FEED}(p,t) + T_{COLL.}(p,t) + T_{UNLOAD}(p,t)] + T_{PROC.}(p) \tag{17}$$

"T(p)" is defined as the number of different tasks that exist in a certain partition "p". Please note that the finer grain the task partition granularity is, the higher the number of tasks will be. In turn "t" is the way of addressing each individual task.

It can be observed that the overall execution time is divided into two terms. On the one hand the summation, for all the individual tasks "T(p)", of the times of those procedures related with the ICAP port is carried out. These procedures cannot be parallelized as they require an individual resource such as ICAP, hence their addition is straightforward. On the other hand, a second term representing the overall data processing time is present. Data processing can be parallelized between different tasks, depending on the available resources in the FPGA and on data dependencies between tasks; therefore, it has to be represented with a generic term. Unfortunately, the measurement or estimation of the dead times and the possibility of applying parallelization is not straightforward. Therefore, a deep analysis and the use of timing diagrams are usually necessary in order to correctly measure data processing time.

Furthermore, as has been explained before, data processing time is dependent on the maximum clock frequency achievable by each of the tasks. Consequently, it is necessary to carry out the complete implementation flow in order to obtain this information. As well as in the original cost function presentation, maximum clock frequency may be unavailable in certain design steps and cannot be estimated. In the original cost function this factor was omitted to solve this issue, however, this cannot be done for R3TOS' cost function as data processing time is a representative factor. In this case the data processing time should be estimated using the predefined clock frequency as the maximum clock frequency. That is, using the default clock frequency selected when the definition of the application was carried out to perform the estimation of the required time for a task to process all its corresponding data. This way the over-clocking considered by the design methodology is not harnessed but the cost function can be applied before the complete physical implementation of the whole design.

Finally, sub-function (18) calculates the normalization term for the execution time. Please note that it carries out the arithmetic mean of the execution times of all the different possible partitions:

$$\overline{T_{EXEC.}} = \frac{1}{P}\sum_{p=1}^{P} T_{EXEC.}(p) \tag{18}$$

In addition to the election of the best task partition, R3TOS considers several implementation strategies that look for the reduction of the execution time:

- Task preservation: In order to reduce function loading/unloading times, a certain task is not unloaded until the area or resources that it occupies are needed. In this way, if the task is to be executed in a short period of time, function loading/unloading is not needed and that time is saved.

- Task parameterization: This technique proposed in the design methodology can also be harnessed in R3TOS if linked to the aforementioned task preservation technique. Task parameterization reduces task programming time both when a standard change is needed (i.e. continuing with the use proposed up to now) and when a certain function/task is executed more than once in a certain waveform (e.g. the two different interleavers present in WiFi or UMTS). That way, just the interleaver pattern needs to be reconfigured prior to the new task execution (provided task area has not been required).

- Snake strategy: Focusing on data feeding and data collecting times, the *Snake* BRAM reuse strategy aims to reduce them. Provided that output data from a function becomes directly the input of the next task, it is possible to improve ICAP bandwidth utilization by reusing BRAM content. In the *Snake* strategy, the output buffer BRAM of a task is not blanked after execution and it is reused as input buffer for the next task. This way, there is no need for data transfer through the ICAP port.

- BRAM size optimization: This feature is more a design recommendation that a design strategy. Taking into account that it is not possible to individually program data into a single BRAM (they are arranged in a vertical 4 BRAM structure), it is advisable to use data slots with a size as similar to the 4x2048 bits available in the BRAMs as possible. In this way ICAP transfer bandwidth is harnessed more efficiently.

3.4. Implementation

As stated in the introduction, the present implementation considers the design and execution of several data coding functions on R3TOS. Besides, it also presents the adaptation of System Generator's communication interface to I2CI and the development of a task context saving and restoration procedure. The whole system has been implemented on a Virtex 4 XC4VLX160 FPGA from Xilinx.

3.4.1. Data coding functions

The data coding functions from the communication standards implemented in the aforementioned multi-standard modulator (WiMAX, WiFi and UMTS) have been used. These functions are: data randomizer, data puncturer, Reed-Solomon encoder, convolutional encoder and data interleaver. As usual, the design of these functions has been carried out with Xilinx's rapid prototyping tool: System Generator.

The use of a modified version of the partitioning methodology has been presented as a possibility for electing the best waveform partition granularity. Unfortunately, due to the necessity of delimiting the duration of this research work, it has not been possible to fully deploy the application of the methodology. Consequently, only certain measurements on the main design factors and some tests involving several function granularities have been carried out. Regarding function granularity, two different approaches have been tested. On the one hand an individual partition of the functions has been carried out. This motivates a function-by-function execution whereby every coding function is scheduled and executed in an individual task in R3TOS. On the other hand, a coarse grained

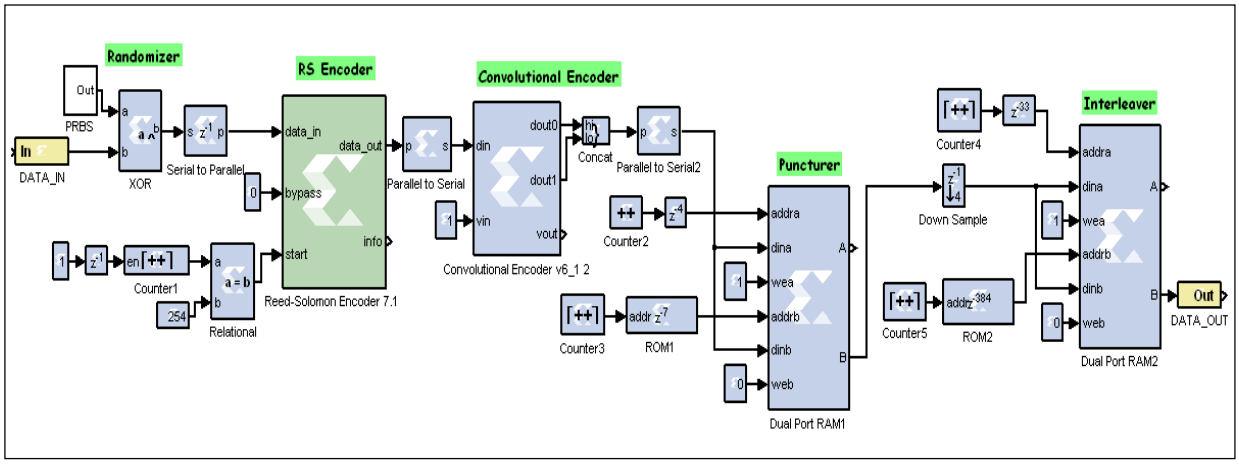


Figure 52: “Full-standard” task implementation for WiMAX

partition has also been tested. In this partition, named "full-standard", every single task embraces the necessary functions for a concrete communication standard. This way the task scheduling only happens when a change of standard is required. Figure 52 shows the implementation in System Generator of the "full-standard" WiMAX task. It can be observed that the task contains all the aforementioned coding tasks.

3.4.2. I2CI connection to System Generator interface

System Generator automatically generates a predefined communication interface for its designs. It is made up of the data input and data output ports (which are directly inferred from the *Gateway_in* and *Gateway_out* blocks placed by the user in the design), plus the clock management signals. These signals are: the clock input itself (*clk*), and the 'clock enable' input (ce) and 'clock enable clear' (ce_clr) input signals in charge of controlling tasks' internal clocking and multi-rate synchronization. In order to make this interface suitable for R3TOS, a joint infrastructure including the I2CI and the aforementioned communication interface has been implemented. This infrastructure extracts data stored in the BRAMs of I2CI and delivers it serially to the input port of task's interface with the appropriate format and data-rate. Similarly, data obtained from the task is adapted and written into I2CI's output BRAM. Regarding task's synchronization, the developed infrastructure generates the ce and ce_clr clear signal from I2CI's HWS resource. To sum up, the developed infrastructure can be observed in Figure 50. It is made up of the glue logic, the address generating counters and control FSM.

3.4.3. Task context saving and restoration procedure

Functions like Data randomizer are not time invariant. The Pseudo Random Binary Sequence (PRBS) block that generates the randomization data has to start from the point of the last execution in order to ensure coherent data recovery in the receiver. The same happens with other signal processing functions like Infinite Impulse Response (IIR) filters or with any task that has been pre-empted. This feature, similar to context saving and restoration performed in software processors, has been extensively covered in [Simmler'00], [Kalte'05] and [Jozwik'10]. However, some issues related with the behaviour of the Flip-Flops have been discovered, which, at the best of the authors' knowledge, are not covered by these previous works, at least not for the Virtex 4/Virtex 5 FPGAs considered in our approach.

When dealing with context saving, it has to be noted that the information that can be read-back via ICAP, if no previous action is performed, contains the initial values of the Flip-Flops, not the current ones. It is precisely the current state of the Flip-Flops the one that holds the context of the task. Moreover, programming a partially reconfigurable area configures everything but the state of the Flip-Flops that maintain their previous state. Therefore, in case another task is placed in the same area and updates the state of these Flip Flops, the context recovery would fail. This way the straight procedure for context saving and recovery, would fail. A first approach to solve this issue could consist in designing tasks so that no Flip-Flops are used. Instead, SRL16 shift registers or LUTs configured as RAM are used. This approach, although functional, can lead to very poor performance in terms of resource utilization and is unfeasible in case the designer does not have control over task

implementation (i.e. third party tasks or tasks designed with rapid prototyping tools in which code is generated automatically). Consequently, another approach should be considered.

Further research has found that Flip-Flop state saving and restoration is responsibility of two ICAP commands: GCAPTURE and GRESTORE [XILINX'09, XILINX'11a]. GCAPTURE copies the current value of the Flip-Flops into the configuration memory so that they can be read back. More precisely, the current Flip-Flop states are stored in the INIT/VALUE bits of the frame that stores the Flip-Flops' configuration of each CLB column (frame with minor address 20 of a certain CLB column). Similarly, the GRESTORE command performs the inverse operation. It initializes the value of the Flip-Flop with the value stored in the INIT/VALUE bits of the configuration memory. GRESTORE is precisely one of the last commands present in the "normal" bitstreams (i.e. non reconfigurable bitstreams), leaving the FPGA initialized and ready for running.

These two commands seem to solve the context saving and restoration problem. Unfortunately, their application range makes them unsuitable for this purpose, at least if no further action is carried out. Both commands are applied to the whole FPGA, that is, when any of them is sent to the ICAP all the Flip-Flops in the FPGA are affected. This issue could be assumable in the case of the GCAPTURE command as normal operation is not affected although the initial state of the system is lost. However, in the case of the GRESTORE command the effect is determinant. When the command is executed looking for the initialization of a certain task, the Flip-Flops of the whole FPGA are initialized. This would lead to a corruption of the R3TOS system and of other tasks being executed at the same time. For this reason the GRESTORE command is not present in the partial bitstreams and therefore the Flip-Flops in the reconfigurable area are not initialized with the values coded in VHDL.

In order to solve these problems two approaches for achieving a correct task context saving and restoration have been developed.

a) Local reset approach

The local reset approach harnesses GCAPTURE in order to perform the context save and develops a local, direct-access, Flip-Flop initialization procedure for context restoration. As already explained, the use of GCAPTURE in this approach has to deal with the loss of the initial configuration of the system. In the current task execution flow implemented by R3TOS this issue has not further influence, however, it should be taken into account in future developments. In order to restore the state of Flip-Flops within just one task, it is necessary to perform a local initialization of those Flip-Flops via a local reset. The key points for this reset are the SR pin of each Flip-Flop and the SRMODE bits in the configuration memory. Driving the SR pin high, the Flip-Flop is initialized with the value stored in the SRMODE bit. Therefore, a manual update of SRMODE bits via ICAP, followed by a reset, enables the load of user-defined values into the Flip-Flops. Access to the SR pin is granted with a proper coding in VHDL or Verilog, or through certain parameters in System Generator or other rapid prototyping tools. In the particular case of this implementation, all the SR signals obtained are connected together to the Hardware-Oriented Semaphore so that an automatic reset is performed prior to the execution of a task. A detailed view of Flip-Flop connections is given in

Figure 53. There, both the aforementioned Flip-Flop physical ports and configuration memory bits can be observed.

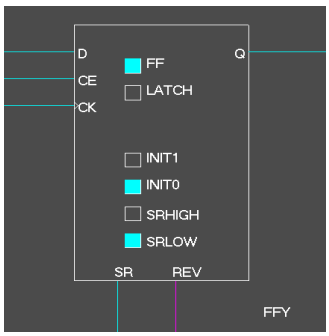


Figure 53: SLICE Flip Flop representation in FPGA editor

Overall, the procedure for context saving and restoration includes the following steps:

1. Execute the GCAPTURE command
2. Read-back the frames that make up the task. The INIT/VALUE pins are already updated with the current values of the Flip-Flops in the partial area.
3. Store the partial bitstream in R3TOS' RAM memory.

When the task needs to be restored:

1. Recover the task's partial bitstream from RAM.
2. Copy the values of INIT/VALUE bits to SRMODE bits.
3. Program-back the task into the reconfigurable area using the modified partial bitstream.
4. Force a local reset prior to task execution

b) Area protection approach

The area protection approach aims to use the GCAPTURE and GRESTORE commands in order to achieve the task context saving and restoration. On this purpose it has been necessary to develop a protection procedure that makes a certain Flip-Flop or group of Flip-Flops immune to these commands. This way a localized application of the commands can be carried out without disturbing

	Top_Bottom	Block Type	Row Address	Column Address	Minor Address
Index	22	21:19	18:14	13:6	5:0

Figure 54: Frame addressing (FAR address)

the normal operation of both the R3TOS kernel and the rest of the tasks under execution.

This protection can be achieved by setting up bit number 13 in the so called "Block Special Frame" of every CLB column within the FPGA. This special frame is accessed setting up a FAR address with "minor address" 0 and a "block type" 3 for Virtex 4 FPGAs or 2 for Virtex 5 FPGAs. In order to clarify the generation of this address, Figure 54 shows the organization of the FAR register. "Top_Bottom", "Row Address" and "Columnn_Address" select the target CLB group (1 clock region height, 1 CLB width), while "Block Type" and "Minor Address" select the type of frame to be accessed.

Focusing on the "Block Special Frame", Figure 55, extracted from the Virtex 5 FPGA configuration user guide [XILINX'11a], shows the use of some of its bits.

Bit Numbers	Use
<15>	Unused
<14>	Gates GTS_CFG_B in IOB column only
<13>	Gates GCAP and GRESTORE in all columns
<12>	Gates GHIGH_B and GWE in all columns
<11:0>	12 ECC bits (assumes that all other bits are 0)

Figure 55: Block Special Frame bits

It can be observed that bit 13 is in charge of the control of the GCAPTURE and GRESTORE commands. Setting up this bit, the Flip-Flops within the CLB group controlled by this frame are not affected by the execution of neither the GCAPTURE nor the GRESTORE commands. Consequently, setting up this protection bit in all the frames of the FPGA and clearing it just for the area of the task to be saved/restored enables the direct use of the GCAPTURE and GRESTORE commands.

To sum up, the area protection approach completes as follows:

1. Access all the CLB groups in the FPGA and protect them by setting up bit number 13 in the Block Special Frame.
2. Unprotect the area occupied by the task to be saved.
3. Execute the GCAPTURE command.
4. Read-back the frames that make up the task. The INIT/VALUE pins are already updated with the current values of the Flip-Flops in the partial area.
5. Store the partial bitstream in R3TOS' RAM memory.
6. Protect again the area occupied by the task.

At this point the aforementioned area can be blanked and used for the execution of another task. Whenever the saved configuration has to be recovered, the procedure continues as follows:

1. Recover the task's partial bitstream from RAM.
2. Program-back the task into the reconfigurable area.
3. Unprotect the area occupied by the task.
4. Execute the GRESTORE command.
5. Protect again the area occupied by the task.
6. Enable task execution.

Figure 56 shows graphically this concept.

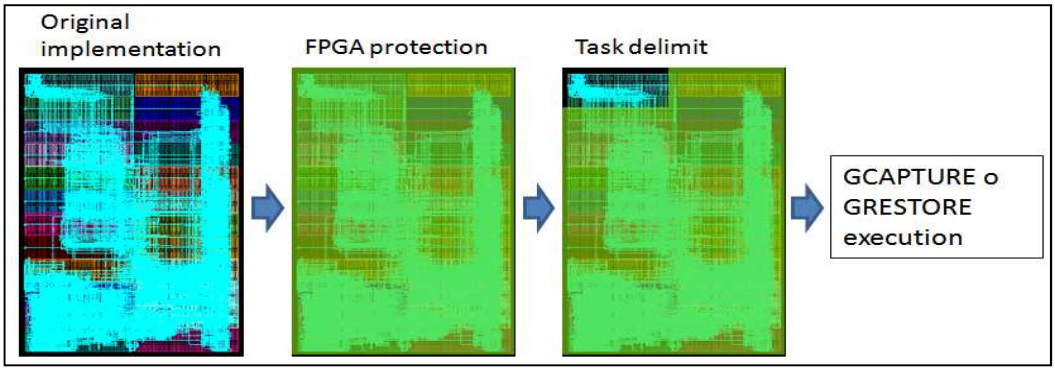


Figure 56: Task context saving and restoration procedure

The proposed procedure addresses both the task context saving and restoration as it is a characteristic needed by R3TOS. However, just the FPGA protection and the local use of the GRESTORE command can and should be used in order to carry out Flip-Flop initialization whenever dynamic partial reconfiguration is used.

Besides, it should be noted that due to technical limitations, the granularity achieved by the protection procedure is restricted to a height of a single clock region (16 CLBs in Virtex 4 and 20 CLBs in Virtex 5) and to a width of a single CLB. Therefore, in case smaller tasks were to be targeted (i.e. placing vertically two tasks within a single clock region) the local reset approach needs to be used in order to achieve a proper initialization.

3.5. Measurements

This section sums up the results obtained from the implementation of the aforementioned data coding function on R3TOS. As already mentioned two implementation strategies have been tested: a "function-by-function" implementation whereby every coding function is scheduled and executed in

an individual task and a "full-standard" implementation where every single task embraces the necessary functions for a concrete communication standard.

Table 16: Task resource utilization

	Resource utilization			
Function	SLICEs	LUTs	FLIP FLOPs	BRAMs
Randomization	58	71	45	2
RS encoder	176	311	202	2
Conv. Encoder	80	74	55	2
Puncturing	39	47	41	4
Interleaver	483	503	425	4
Full-standard	798	831	724	6
R3TOS (full)	5571	7383	4157	16
R3TOS (stand-alone)	1793	2778	1157	6

Table 16 shows the FPGA resource utilization of the implemented functions as well as R3TOS' overhead. It can be observed that a minimal of 2 BRAMs per task corresponding to the input/output buffers has been used. According to the BRAM size optimization strategy introduced before, the use of 4 BRAMs per buffer increases the efficiency of the ICAP transfers, however, in this case and looking just for simplicity a single BRAM per buffer is used. It is also noticeable that the Full-standard design occupies fewer resources than the sum of the individual functions. This is due to the removal of the glue logic that connects the input/output buffers to each task. In the full-standard implementation this glue logic is only used once. The size of this glue logic is dependent on the size of the buffer and on the data width to be accessed. In the presented implementation about 10 SLICES of the number shown in Table 16 for each function correspond to the glue logic. Consequently, the Full-standard implementation achieves a 5% size reduction compared to the sum of the individual functions.

Concerning R3TOS' overhead two different data sets are presented, one with a full version of R3TOS with all the components mentioned in Section 3.2 implemented and another, named 'Stand-alone', in which just the minimum components needed by the SDR implementation are present (i.e. the MicroBlaze processor is not preset). Taking into account the small size of the SDR data coding tasks, the overhead of the full version of R3TOS is quite high in this case. However, it has to be stated that this version includes a MicroBlaze processor that cannot be considered as an overhead as it offers the possibility of executing complex software tasks on it if needed. The overhead of the stand-alone version, on its side, is more acceptable in relation with the functions it offers.

Table 17: Task execution time comparison

	Execution time of a single iteration					
Implementation	Task loading	Data feeding	Processing	Data recovery	Task unloading	Total
Randomization	90 us	58,4 us	164,8 us	58,4 us	90 us	461,6 us
RS encoder	207,7 us	58,4 us	164,8 us	58,4 us	207,7 us	697 us
Conv. Encoder	104,2 us	58,4 us	164,8 us	58,4 us	104,2 us	490 us
Puncturing	90 us	58,4 us	164,8 us	58,4 us	90 us	461,6 us
Interleaver	536,4 us	58,4 us	168,3 us	58,4 us	536,4 us	1357,9 us
Total funct. by funct.	1028,3 us	350,4 us	827,5 us	350,4 us	1028,3 us	3,58 ms
Total funct. by funct. (Snake)	1028,3 us	116,8 us	827,5 us	116,8 us	1028,3 us	3,12 ms
Full-standard design	879,2 us	116,8 us	227 us	116,8 us	879,2 us	2,22 ms

Table 17 compares the total execution time of the different approaches that have been used: the function-by-function execution (in both the default version and using the Snake strategy) and the full-standard one, in which all the coding functions are implemented in parallel. The total execution time is divided into task loading time (made up of task transfer time from RAM to ICAP and task context restoration time), data feeding and recovery times, processing time and task unloading time (made up of task context saving time and task transfer from ICAP to RAM)

We note that these timing measurements have been carried out with a timer/counter within MicroBlaze. A 100 MHz clock is connected to the processor; therefore, the theoretical resolution of this timer is 10 ns. However, due to the delay introduced by processor's bus the effective resolution rises to 0.1 us. Consequently, the obtained measurements have been rounded to this resolution.

At first sight it can be observed that the full-standard execution is the fastest one (2.22 ms). This kind of implementation, closer to the traditional way of programming FPGAs, takes advantage of the parallelization potential that the device offers to execute tasks in a pipelined way. However, it has to be noted that the measured times for the function-by-function execution correspond to the worst scenario possible. The scheduler has been restricted to execute a single task at a time (although the FPGA has enough free area for executing more tasks in parallel). Besides, only a single iteration has been measured, that is, once the five tasks have been executed, the system stops. In a cyclic execution where the functions have to be implemented once again, and where more than one task is executed in parallel, a virtual pipeline would be obtained leading to a further reduction in the execution time (i.e. a recursive execution with resources for the execution of two tasks in parallel leads to an execution time per cycle of 2.9 ms). When the scheduler is free to optimize the execution, it harnesses the free times of the ICAP port, for example during the processing of a task, in order to prepare (i.e. download and initialize) the next tasks to be executed. This way the task loading time overlaps with the processing time and the next execution is instantaneous. This is known as "Configuration prefetching".

Comparing the function-by-function implementations, the effect of the Snake strategy can be seen. Removing the data copy from task to task, a 15% reduction in task execution time is achieved (3.12

Table 18: Task loading/unloading times

	Task loading/unloading time	
Function	Protection/unprotection time	Task transfer time
Randomization	7,5 us	75 us
RS encoder	22,4 us	162,9 us
Conv. Encoder	9,4 us	85,4 us
Puncturing	7,5 us	75 us
Interleaver	61,7 us	413 us
Full-standard	102,8 us	673,6 us
Initial protection	2,15 ms	-

ms with the Snake strategy vs. 3.58 ms with data copy).The concatenation of longer task sets would make this reduction more pronounced.

Finally, in order to better appreciate the task context saving and restoration procedure, some measurements on this process have been carried out. It is made up the task transfer process (both from RAM to ICAP or from ICAP to RAM), the task protection/unprotection process and the execution of the GCAPTURE/GRESTORE commands. The execution of these commands takes a single clock cycle and therefore it can be considered a negligible amount. In turn Table 18 shows the duration of task protection/unprotection process and task transfer process.

On the one hand the initial time needed to carry out the complete protection of the whole FPGA can be observed. These 2.15 milliseconds, in which the Flip-Flop protection bit of all the special frames within the FPGA is activated, are spent only once in the start-up of the system. Consequently, they are not included in the cost function and can be considered just as an initial delay. Besides, it should be noted that this time is completely proportional to size of FPGA to be used. In this case, being the XC4VLX160 FPGA one of the biggest in its family, offers quite a big protection time. On the other hand Table 18 also shows the protection/unprotection time of the different tasks (time for both processes is equal). The number of special frames to be modified depends on the number of CLB columns used by each of the tasks. Consequently, assuming a 70% area utilization, the obtained times are proportional to the task resource utilization presented in Table 16. As an exception, Randomization and Puncturing tasks have the same protection time. This is because a minimum size of 4 CLB columns has been established in order to maintain a task structure in which BRAM buffers are placed on the sides of the CLB-based area. In regard to the relation with the task transfer time it can be observed that task protection/unprotection adds a 10-15% to task transfer time. Taking into account that both protection and unprotection processes are required for saving or restoring a task's context, the total overhead added by this process represents a 20-30% of the overall task loading unloading time. Although it is a high overhead, the task context saving and restoration procedure is compulsory for certain tasks hence it cannot be suppressed.

4. SUMMARY

The current chapter has presented two implementations that serve as proof-of-concept of the use of FPGA dynamic partial reconfiguration and rapid prototyping tools in the implementation of Software Defined Radios. On the one hand a simple cognitive radio has been used for replacing a wired video transmission system. The use of this type of radios into industrial environments is highly interesting as they are able to adapt to the harsh conditions present in them. The implemented possibility of changing the transmission frequency if an interference is detected, is a first step in towards demonstrating this sustainability. On the other hand, the feasibility of using this design framework together with the R3TOS hardware operating system has been proved; leading to a system in which SDR takes care of a secure communication while R3TOS guarantees efficient and reliable hardware utilization. The measurements that have been carried out have shown that, as expected, the reconfiguration time (or task execution time in R3TOS) is one of the main disadvantages of dynamic partial reconfiguration when used in communication systems. Nonetheless, the diverse general strategies presented during this research work and the particular solutions proposed for R3TOS are able to reduce it.

Chapter 8

Summary and conclusions

8. Summary and conclusions

1. INTRODUCTION

This chapter concludes this thesis. It summarizes the work developed during this PhD thesis, along with its main contributions, providing also some possible directions for the future work.

2. SUMMARY OF THESIS AND FINAL CONCLUSIONS

In this PhD dissertation, the optimization of Software Defined Radio (SDR) implementation over a novel design framework, made up of FPGA dynamic partial reconfiguration and rapid prototyping tools, has been addressed. This optimization intends to be carried out by the developed design methodology, in charge of the static/reconfigurable partitioning of the SDR. Besides, the feasibility demonstration of both the proposed design framework and design methodology has been demonstrated via their use in three real and functional implementations. Namely: A multi-standard modulator implementing WiFi, WiMAX and UMTS, a small-form-factor cognitive video transmission system and the implementation of several data coding functions over R3TOS, a hardware operating system developed by the University of Edinburgh.

Chapter 2 has provided some insight into the background and existing literature on the three main aspects that take part in this research work: Software Defined Radios, FPGA dynamic partial reconfiguration and rapid prototyping tools. The benefits and drawbacks of this design framework have been analyzed and the key points obtained. Reconfiguration time, design size increment, timing degradation and some issues related with Flip-Flop initialization have been established as the most important factors that need to be addressed in order to achieve a successful implementation. On this basis, the need of a design methodology has been determined and its fundaments set up.

Chapters 3, 4 and 5 have thoroughly described the proposed design methodology. Once the degree of freedom over which the methodology has to act has been described, all the steps involved have been widely explained. Special focus has been set into the two most relevant steps: the establishment of the common parts of the SDR and the design of the cost function used to quantitatively evaluate the performance of each partition.

The defined design methodology has demonstrated to be suitable for optimizing the implementation of SDRs in the proposed design framework. The guidelines considered by the methodology ensure a proper design flow that leads to the election of the optimal design partition. Besides, the use of rapid prototyping tools during the design process permits the reuse of implementations carried out previously, hence easing and reducing design time. Unfortunately, certain limitations of this design methodology have also been detected. On the one hand the optimization capacity achievable by the methodology is dependent on the characteristics of the waveforms to be implemented in the SDR. If the nature of the waveforms is completely different, there will not be many common functions and therefore the optimization possibility will be reduced. On the other hand, when dealing with complex

designs, the amount of data generated by the design methodology grows enormously, making it unfeasible to be applied by hand. In order to solve this issue, the basics of an automation procedure have also been proposed. This way, data management would be carried out automatically, which would also lead to a reduction in the design time.

It should also be noted that although the proposed design methodology and design framework have been oriented towards its use with SDRs, they are not limited to this field. The use of FPGA dynamic partial reconfiguration and rapid prototyping tool can be applied to any other field that requires data processing or on-the-fly change of parameters. Their use in control electronics could be a good example. In turn, the design methodology can also be used in other fields regardless the fact that commonalities are needed between the parts to be reconfigured in order to achieve a significant optimization capacity. Nevertheless, certain parts of the methodology, such as the proposed parameterization techniques can be directly harnessed in many designs.

Chapters 6 and 7 have presented three real and functional implementations that aim to demonstrate that the proposed design framework and design methodology are valid for the implementation and optimization of SDRs. The design process of the multi-standard modulator presented in Chapter 6, has directly applied all the steps considered in the design methodology, hence validating it. In turn, the small-form-factor cognitive video transmission system and the implementation of data coding functions over R3TOS, do not strictly apply the design methodology but aim to explore other design possibilities. In that regard, the use of SDRs into industrial environments has demonstrated to be highly interesting as they are able to adapt to the harsh conditions present in them, hence securing reliable communications. Besides, their joint with a hardware operating system such as R3TOS reinforces the reliability of the communication, leading to a system in which SDR takes care of a secure communication while R3TOS guarantees efficient and reliable hardware utilization.

Finally, the encountered issue around Flip-Flop initialization and that has motivated the design of a task context saving and restoration procedure, appears to be an important problem that, at the best of author's knowledge, is not sufficiently covered in the literature.

3. SUMMARY OF ACHIEVEMENTS

The main achievements of this research work and the associated scientific publications are the following:

1. A novel design framework for the implementation of Software Defined Radios has been proposed and tested. This design framework is made up of FPGA dynamic partial reconfiguration and rapid prototyping tools. The drawbacks and design implications of this framework have been analyzed and evaluated, coming to the conclusion that, despite the multiple benefits that it offers, a design methodology is needed in order to optimize the obtained implementations. [Torrego'09] and [Torrego'10]

2. A complete design methodology that addresses static/reconfigurable partitioning for optimizing the SDR implementation through the aforementioned design framework has been designed. The methodology selects which parts of the SDR are to be implemented statically and which in a reconfigurable way in order to minimize system's cost. [Torrego'12c]

3. A cost function has been developed in order to quantitatively evaluate a certain partition's optimality. The function evaluates those design factors that have been determined as most important in the performance of a design. Namely: design size, reconfiguration time and maximum applicable clock frequency.

4. A multi-standard modulator implementing three of the most used communication standards nowadays (WiFi, WiMAX and UMT) has been designed and implemented using the proposed design framework and design methodology. This implementation serves as a proof-of-concept and as a validation of the first and second contributions. [Torrego'12c]

5. A small-form-factor cognitive video transmission system has been implemented using the proposed design framework. This system is able to change its Intermediate Frequency (IF) if the transmission channel is occupied, hence achieving a secure communication. The frequency change is carried out via dynamic partial reconfiguration. It provides a tangible demonstration of the feasibility of the proposed framework with a fully functional application.[Torrego'11] and [Torrego'12a]

6. The implementation of several data coding functions used in SDRs has been carried out over R3TOS, a Reliable, Reconfigurable and Real-Time hardware Operating System developed by the University of Edinburgh. The design of the functions has been carried out using rapid prototyping tools and applying some of the guidelines proposed by the design methodology. Therefore the possibility of using this methodology in other reconfigurable architectures is demonstrated. [Torrego'12b]

7. A novel task context saving and restoration procedure has been designed for its use with R3TOS. This feature, similar to context saving and restoration performed in software processors, requires special care when dealing with FPGA dynamic partial reconfiguration as some issues related with Flip-Flop initialization appear. [Torrego'12b] and [Torrego'12c]

4. FUTURE WORK

A number of additional issues, between the ones presented in this PhD dissertation, can be explored in the future as an improvement and extension of the current work. These are some of the directions for a possible extension of the presented results:

- Complete development of the basics of the automation procedure for the design methodology that have been introduced in Section 4 of Chapter 3. As has been shown, the manual

management of the design methodology is unfeasible when dealing with complex designs. Therefore, the automation of the methodology would add a great value to it.

- Inclusion in the design methodology of optimization algorithms. In regards to the simplification of the design methodology, the inclusion of certain optimization algorithms that could reduce the number of partitions to be evaluated, would further speed up and ease the design flow.

- Inclusion of the power consumption factor in the cost function. Power consumption is an important characteristic to be considered in any application. It has not been directly addressed in this research work as both its estimation and real measurement are complex tasks so as to be carried out for every partition. Nonetheless, provided the power estimation tools speed up and ease their operation, the inclusion of this factor would be really interesting.

- Generation of estimating functions to be used with R3TOS. Section 3.3 of Chapter 7 has presented a custom cost function for R3TOS. This function includes a new term for the evaluation of the execution time. Although part of this time is related with the maximum clock frequency, hence not being possible to estimate it, the rest of factors that make up this execution time can be estimated. Therefore, the generation of the corresponding estimating function would enable a better evaluation of the system at early design steps.

- Re-implementation of the multi-standard modulator harnessing direct ICAP parameterization. This parameterization technique presented in Section 4.1 of Chapter 4 uses direct ICAP access to certain parts of the parameterized version in order to change its functionality. This parameterization technique reduces the typical design size increment that happens with parameterization at the expense of a small reconfiguration time. The evaluation of the effect that this technique has in the design methodology would be highly valuable.

LIST OF PUBLICATIONS

1. Torrego, Val, I., Muxika, E., Berrizbeitia, A.: 'Partial reconfiguration in FPGA rapid prototyping tools'. Proc. IP-Embedded Systems Conference (IP09), Grenoble (France), Year 2009

2. Torrego, R., Val, I., Muxika, E., Berrizbeitia, A.: 'A step by step methodological approach for merging FPGA dynamic reconfiguration and algorithm rapid designing tools'. Proc. Smart Systems Integration European Conference & Exhibition (SSI2010), Como (Italy), Year 2010

3. Torrego, R., Val, I., Muxika, E.: 'OQPSK cognitive modulator fully FPGA implemented via dynamic partial reconfiguration and rapid prototyping tools'. Proc. Wireless Innovation Forum European Conference on Communication Technologies and Software Defined Radio (SDR'11 – WInnComm – Europe), Brussels (Belgium), Year 2011

4. Torrego, R., Val, I., Muxika, E., Iturbe, X., Benkrid, K.: 'Data Coding Functions for Software Defined Radios implemented on R3TOS'. Proc. Field Programmable Logic and Applications (FPL'12), International Conference on, pp. 33-40, Oslo (Norway), Year 2012

5. Torrego, R., Val, I., Muxika, E., Iturbe, X., Benkrid, K.: 'Implicaciones del uso de la reconfiguración parcial dinámica de las FPGAs en la implementación de Radios Definidas por Software'. Proc. III Jornadas de Computación Empotrada (JCE2012), Elche (Spain), Year 2012

6. Torrego, R., Val, I., Muxika, E.: 'Small form factor Cognitive Radio implemented via FPGA partial reconfiguration replacing a wired video transmission systems'. Proc. Wireless Innovation Forum Conference on Communications Technologies and Software Defined Radio (SDR-WInnComm'12) , Washington (US), Year 2012

7. Iturbe, X., Benkrid, K., Arslan, T., Torrego, R., and Martinez, I.: 'Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs'. Proc. Field Programmable Logic and Applications (FPL'11), International Conference on, pp. 295-300, Milano (Italy), Year 2011

8. Iturbe, X., Benkrid, K., Torrego, R., Ebrahim, A., Arslan, T.: 'Online Clock Routing in Xilinx FPGAs for High-Performance and Reliability'. NASA/ESA Conference on Adaptive Hardware and Systems (AHS'12), Erlangen (Germany), Year 2012.

9. Iturbe, X., Benkrid, K., Torrego, R., Hong, C., Ebrahim, A., Martinez, I., Arslan, T., Perez, J.: "R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient and Dependable Computing on FPGAs". Submitted to IEEE Transactions on Computers (under review)

REFERENCES

[3GPP'08] '3GPP TS 25.212 version 7.8.0 Release 7', 2008

[3L'05] 3L: 'Diamond User Guide Sundance Edition', 2005, 3.0

[ACCELLERA'12] http://www.accellera.org/downloads/standards/systemc/about_systemc/, accessed 2012

[ACTEL'04] ACTEL: 'AC202 - ProASICPLUS PLL Dynamic Reconfiguration Using JTAG', 2004, pp. 12

[ACTEL'12] http://www.actel.com/company/about/default.aspx, accessed 2012

[AGILENT'12a] http://www.home.agilent.com/agilent/product.jspx?nid=-34264.0&cc=US&lc=eng&pageMode=OV, accessed 2012

[AGILENT'12b] http://www.home.agilent.com/agilent/product.jspx?nid=-34264.870751.00&cc=ES&lc=spa, accessed 2012

[Alaus'09] Alaus, L., Palicot, J., Roland, C., Louët, Y., and Noguet, D.: 'Promising Technique of Parameterization For Reconfigurable Radio, the Common Operators Technique. Fundamentals and Examples', Journal of Signal Processing Systems, 2009

[ALTERA'08] ALTERA: 'FPGA Run-Time Reconfiguration: Two Approaches', 2008

[ALTERA'09] ALTERA: 'DSP Builder user guide', 2009, 9 edn.

[ALTERA'10] ALTERA: 'Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs', 2010

[ALTERA'12] http://www.altera.com/corporate/crp-index.html, accessed 2012

[Astarloa'05] Astarloa, A.: 'Reconfiguracion dinamica de sistemas modulares multi-procesador en dispositivos SoPC', Universidad del Pais Vasco, 2005

[Athalye'05] Athalye, A., and Hong, S.: 'Mapping of partial reconfigurable data flows to Xilinx FPGAs'. Proc. SOC Conference, IEEE International, 2005, pp. 111-112

[ATMEL] http://www.atmel.com/about/corporate/default.aspx, accessed 2012

[ATMEL'02] ATMEL: 'FPSLIC on-chip Partial Reconfiguration of the Embedded AT40K FPGA', 2002

[Bayar'08] Bayar, S., and Yurdakul, A.: 'Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP)'. Proc. 2nd HiPEAC Workshop on Reconfigurable Computing, 2008

[Berthelot'08] Berthelot, F., Nouvel, F., and Houzet, D.: 'A flexible system level design methodology targeting run-time reconfigurable FPGAs', Eurasip Journal on Embedded Systems, 2008

[Bobda'07] Bobda, C.: 'Introduction to Reconfigurable Computing' (Springer, 2007)

[Boch'06] Boch, A., Feletti, L.C., Laddomada, M., Mesiti, F., Mondin, M., Seoane, J., Borio, D., Calafato, A., Daneshgaran, F., Presti, L.L., Smolnikar, M., Javornik, T., and Mohorcic, M.: 'Report on signal processing implementation on a DSP board', 2006

[Bonamy'12] Bonamy, R., Hung-Manh, P., Pillement, S., and Chillet, D.: 'UPaRC—Ultra-fast power-aware reconfiguration controller'. Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, pp. 1373-1378

[Cancare'07] Cancare, F.: 'Modeling Methodologies for Dynamic Reconfigurable Systems', University of Illinois, 2007

[Carmichael'99] Carmichael, C., Fuller, E., Blain, P., and Caffrey, M.: 'SEU Mitigation Techniques for Virtex FPGAs in Space Applications'. Proc. Military and Aerospace Programmable Logic Devices International Conference(MAPLD) 1999

[Cofer'05] Cofer, R.C., and Harding, B.F.: ‘Rapid system prototyping with FPGAs’ (Elsevier, 2005)

[Custodio'07] Custodio, E., and Marsland, B.: ‘Self-Healing Partial Reconfiguration of an FPGA’, 2007

[Chechi'11] Chechi, R., and Khanna, R.: ‘QoS Support in Wi-Fi, WiMAX & UMTS Technologies’, International Journal on Electronics & Communication Technology (IJECT), 2011, 2, (3), pp. 176-179

[Delahaye'07] Delahaye, J.-P., Palicot, J., Moy, C., and Leray, P.: ‘Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform’, 2007

[Donato'05] Donato, A., Ferrandi, F., Redaelli, M., Santambrogio, M.D., and Sciuto, D.: ‘Caronte: a complete methodology for the implementation of partially dynamically self-reconfiguring systems on FPGA platforms’. Proc. Field-Programmable Custom Computing Machines (FCCM), 2005, pp. 321-322

[Donthi'03] Donthi, S., and Haggard, R.L.: ‘A survey of dynamically reconfigurable FPGA devices’. Proc. 35 th Southeastern Symposium on System Theory, 2003, pp. 422-426

[Godard'07] Godard, L., Wang, H., Moy, C., and Leray, P.: ‘COMMON OPERATORS DESIGN ON DYNAMICALLY RECONFIGURABLE HARDWARE FOR SDR SYSTEMS’. Proc. SDR Forum Technical Conference 2007 pp. Pages

[Göhringer'10] Göhringer, D., Hübner, M., Benz, M., and Becker, J.: ‘A Design Methodology for Application Partitioning and Architecture Development of Reconfigurable Multiprocessor Systems-on-Chip’. Proc. 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2010, pp. 259-262

[Haessig'05] Haessig, D., Hwang, J., Gallagher, S., and Uhm, M.: ‘A case study of Xilinx System Generator design flow for rapid development of SDR waveforms’. Proc. SDR Forum Technical Conference, Orange County, 2005

[Haessig'06] Haessig, D., Regis, R., and Hermeling, M.: ‘A CASE STUDY COMPARING TRADITION TO MODEL-BASED RAPID DEVELOPMENT OF SDR WAVEFORMS – PART II’. Proc. SDR 06 Technical Conference and Product Exposition, 2006

[Hansen] Hansen, S.G., Koch, D., and Torresen, J.: ‘High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro’. Proc. Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pp. 174-180

[HART'07] HART: ‘HART Field Communication Protocol Specification, Revision 7.0’, 2007

[He'12] He, K., Crockett, L., and Stewart, R.: ‘Dynamic reconfiguration technologies based on FPGA in software defined radio system’, Journal of Signal Processing Systems, 2012, 69, (1), pp. 75-85

[Hong'11a] Hong, C., Benkrid, K., Iturbe, X., Ebrahim, A., and Arslan, T.: ‘Efficient On-Chip Task Scheduler and Allocator for Reconfigurable Operating Systems’, IEEE Embedded Systems Letters, 2011, 3, (3), pp. 85-88

[Hong'11b] Hong, C., Benkrid, K., Iturbe, X., Erdogan, A.T., and Arslan, T.: ‘An FPGA task allocator with preliminary First-Fit 2D packing algorithms’. Proc. Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on, 2011, pp. 264-270

[IEEE'97] ‘IEEE 802.11-1997: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.’, 1997

[IEEE'04] ‘IEEE 802.16-2004: IEEE Standard for Local and Metropolitan Area Networks’, 2004

[Iturbe'12a] Iturbe, X.: ‘Design and Implementation of a Reliable Reconfigurable Real-Time Operating System’, University of Edinburgh, 2012

[Iturbe'11a] Iturbe, X., Benkrid, K., Arslan, T., Chuan, H., Erdogan, A.T., and Martinez, I.: 'Enabling FPGAs for future deep space exploration missions: Improving fault-tolerance and computation density with R3TOS'. Proc. Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on, 2011, pp. 104-112

[Iturbe'11b] Iturbe, X., Benkrid, K., Arslan, T., Torrego, R., and Martinez, I.: 'Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs'. Proc. Field Programmable Logic and Applications (FPL'11), International Conference on, Milano (Italy), 2011, pp. 295-300

[Iturbe'10] Iturbe, X., Benkrid, K., Erdogan, A.T., Arslan, T., Azkarate, M., Martinez, I., and Perez, A.: 'R3TOS: A reliable reconfigurable real-time operating system'. Proc. Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on, 2010, pp. 99-104

[Iturbe'12b] Iturbe, X., Benkrid, K., Torrego, R., Ebrahim, A., and Arslan, T.: 'Online Clock Routing in Xilinx FPGAs for High-Performance and Reliability'. Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS'12), Erlangen (Germany), 2012

[Jozwik'10] Jozwik, K., Tomiyama, H., Honda, S., and Takada, H.: 'A Novel Mechanism for Effective Hardware Task Preemption in Dynamically Reconfigurable Systems'. Proc. Field Programmable Logic and Applications (FPL), 2010 International Conference on, 2010, pp. 352-355

[Kalte'05] Kalte, H., and Porrmann, M.: 'Context saving and restoring for multitasking in reconfigurable systems'. Proc. Field Programmable Logic and Applications, 2005. International Conference on, 2005, pp. 223-228

[Khawam'08] Khawam, S., Nousias, I., Milward, M., Ying, Y., Muir, M., and Arslan, T.: 'The Reconfigurable Instruction Cell Array', Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2008, 16, (1), pp. 75-85

[Kirkpatrick'83] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.: 'Optimization by Simulated Annealing', Science, 1983, 220, (4598), pp. 671-680

[Kuo'12] Kuo, Y.-s., Schmid, T., and Dutta, P.: 'A compact, inexpensive, and battery-powered software-defined radio platform'. Proc. 11th international conference on Information Processing in Sensor Networks (IPSN'12), Beijing (China), 2012, pp. 137-138

[LATTICE'12] http://www.latticesemi.com/corporate/index.cfm?source=topnav, accessed 2012

[Lew'04] Lew, J.K.: 'Low Power System Design Techniques Using FPGAs', 2004

[Lin'07] Lin, Y., Lee, H., Woh, M., Harel, Y., Mahlke, S., Mudge, T., Chakrabarti, C., and Flautner, K.: 'SODA: A High-Performance DSP Architecture for Software-Defined Radio', Micro, IEEE, 2007, 27, (1), pp. 114-123

[Liu'09] Liu, M., Kuehn, W., Lu, Z., and Jantsch, A.: 'Run-time partial reconfiguration speed investigation and architectural design space exploration'. Proc. FPL 09: 19th International Conference on Field Programmable Logic and Applications, Prague, 2009, pp. 498-502

[LYRTECH'12] http://lyrtechrd.com/en/products/view/+model-based-design-kits, accessed 2012

[Maxfield'10] Maxfield, C.: 'FPGAs: System gates or logic cells/elements?', 2010.

[Mecwan'11] Mecwan, A.I., and Gajjar, N.P.: 'Implementation of Software Defined Radio on FPGA'. Proc. Engineering (NUiCONE), 2011 Nirma University International Conference on, 2011

[MENTOR'12] http://www.mentor.com/esl/catapult/overview//, accessed 2012

[Mernik'05] Mernik, M., Heering, J., and Sloane, A.M.: 'When and how to develop domain-specific languages', ACM Computing Surveys (CSUR) 2005, 37, (4), pp. 316 - 344

[MICROSOFT'12] http://msdn.microsoft.com/library/xk24xdbe.aspx, accessed 2012

[Mitola'92] Mitola, J.: ‘Software radios-survey, critical evaluation and future directions’. Proc. National Telesystems Conference, 1992, pp. 13/15-13/23

[Mitola'95] Mitola, J.: ‘The software radio architecture’, Communications Magazine, IEEE, 1995, 33, (5), pp. 26-38

[Moy'06] Moy, C., Palicot, J., Rodriguez, V., and Giri, D.: ‘Optimal determination of common operators for multi-standards software defined radio’. Proc. 4th Karlsruhe Workshop on Software Radios, Karlsruhe (Germany), 2006

[Nicollet'03] Nicollet, E., and Demeure, C.: ‘DSP software architecture for Software Defined Radio’. Proc. DSP enabled Radio, 2003 IEE Colloquium on, 2003

[Noguera'01] Noguera, J., and Badia, R.M.: ‘A HW/SW partitioning algorithm for dynamically reconfigurable architectures’. Proc. Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings, 2001, pp. 729-734

[Olds'12] http://homepages.paradise.net.nz/peterfr2/avrs232sender.htm, accessed 2012

[PENTEK'10] PENTEK: ‘Software Defined Radio Handbook’, 2010

[Rappaport'01] Rappaport, T.: ‘Wireless Communications: Principles and Practice’ (2001, 2nd edn. 2001)

[Rhiemeier'02] Rhiemeier, A.: ‘Benefits and limits of parameterized channel coding for software radio’. Proc. 2nd Karlsruhe Workshop on Software Radios, Karlsruhe (Germany) 2002

[Saha'09] Saha, A., and Sinha, A.: ‘An FPGA Based Architecture of a Novel Reconfigurable Radio Processor for Software Defined Radio’. Proc. Education Technology and Computer, 2009. ICETC '09. International Conference on, 2009, pp. 45-49

[Schoenes'03] Schoenes, M., Eberli, S., Burg, A., Perels, D., Haene, S., Felber, N., and Fichtner, W.: ‘A novel SIMD DSP architecture for software defined radio’. Proc. Circuits and Systems, 2003 IEEE 46th Midwest Symposium on, 2003, pp. 1443-1446 Vol. 1443

[Shah'09] Shah, S., and Sinha, V.: ‘GMSK Demodulator Using Costas Loop for Software-Defined Radio’. Proc. ICACC '09. International Conference on Advanced Computer Control., 2009, pp. 757-761

[Simmler'00] Simmler, H., Levinson, L., and Manner, R.: ‘Multitasking on FPGA Coprocessors’. Proc. Field Programmable Logic and Applications (FPL), 2000 International Conference on, 2000.

[SUNDANCE'05] SUNDANCE: ‘SMT8096 User Manual Version 1.2’, 2005.

[SUNDANCE'09] SUNDANCE: ‘SMT6040 “Sundance Simulink Toolbox”’, 2009

[Torrego'11] Torrego, R., Val, I., and Muxika, E.: ‘OQPSK cognitive modulator fully FPGA-implemented via dynamic partial reconfiguration and rapid prototyping tools’. Proc. Wireless Innovation Forum European Conference on Communications Technologies and Software Defined Radio (SDR’11 – WInnComm – Europe), Brussels (Belgium), 2011, pp. 142-147

[Torrego'12a] Torrego, R., Val, I., and Muxika, E.: ‘Small-form-factor cognitive radio, implemented via FPGA partial reconfiguration, replacing a wired video transmission system’. Proc. Wireless Innovation Forum European Conference on Communications Technologies and Software Defined Radio (SDR’12 – WInnComm), Washington, 2012.

[Torrego'09] Torrego, R., Val, I., Muxika, E., and Berrizbeitia, A.: ‘Partial reconfiguration in FPGA rapid prototyping tools’. Proc. IP-Embedded Systems Conference (IP09), Grenoble (France), 2009.

[Torrego'10] Torrego, R., Val, I., Muxika, E., and Berrizbeitia, A.: 'A step by step methodological approach for merging FPGA dynamic reconfiguration and algorithm rapid designing tools'. Proc. Smart Systems Integration European Conference & Exhibition (SSI2010), Como (Italy), 2010

[Torrego'12b] Torrego, R., Val, I., Muxika, E., Iturbe, X., and Benkrid, K.: 'Data Coding Functions for Software Defined Radios implemented on R3TOS'. Proc. Field Programmable Logic and Applications (FPL'12), International Conference on Oslo (Norway), 2012.

[Torrego'12c] Torrego, R., Val, I., Muxika, E., Iturbe, X., and Benkrid, K.: 'Implicaciones del uso de la reconfiguración parcial dinámica de las FPGAs en la implementación de Radios Definidas por Software'. Proc. III Jornadas de Computación Empotrada (JCE2012), Elche (Spain), 2012.

[UNREAL'12] http://www.umediaserver.net/umediaserver/download.html, accessed 2012

[Wang'02] Wang, J.J., Cronquist, B., McCollum, J., Katz, R., Kleyner, I., and Koga, R.: 'Single Event Effects of a FLASH-based FPGA', 2002

[Wenzel'99] Wenzel, W., and Hamacher, K.: 'A Stochastic tunneling approach for global minimization', Physical Review Letters, 1999, 82, pp. 3003–3007

[Wichman'06] Wichman, S., Adyha, S., Ahrens, S., Ambli, R., Alcorn, B., Connors, D.D., and Fay, D.: 'Partial Reconfiguration Across FPGAs'. Proc. MAPLD International Conference 2006

[WIINN-FORUM'12] http://www.wirelessinnovation.org/page/Defining_CR_and_DSA, accessed 2012

[Wolf'04] Wolf, W.: 'Architectures of FPGAs.', in: 'FPGA-Based System Design' (2004), pp. 105-164

[XILINX'00a] XILINX: 'Core Generator Guide', 2000a, 3.1i edn.

[XILINX'00b] XILINX: 'FPGA Editor Guide 3.1i', 2000b

[XILINX'01] XILINX: 'XAPP404 - Xilinx Alliance 3.1i Modular Design', 2001

[XILINX'04] XILINX: 'XAPP 290 - Two Flows for Partial Reconfiguration: Module Based or Difference Based', 2004

[XILINX'06] XILINX: 'AccelDSP Synthesis Tool Supported MATLAB Constructs and Functions', 2006, edn.

[XILINX'08a] XILINX: 'AccelDSP Synthesis Tool User Guide', 2008, 10.1 edn.

[XILINX'08b] XILINX: 'System Generator for DSP User Guide', 2008, 10.1.1 edn.

[XILINX'08c] XILINX: 'XAPP988-Correcting Single-Event Upsets in Virtex-4 Platform FPGA Configuration Memory', 2008

[XILINX'09] XILINX: 'UG71 Virtex-4 FPGA Configuration User Guide', 2009

[XILINX'10] XILINX: 'SEU Strategies for Virtex-5 Devices', 2010

[XILINX'11a] XILINX: 'UG191 - Virtex-5 FPGA Configuration User Guide', 2011, V3.10 edn

[XILINX'11b] XILINX: 'UG628 - Command Line Tools User Guide', 2011, 12.1 edn., pp. 219-248

[XILINX'11c] XILINX: 'UG702 - Partial Reconfiguration User Guide', 2011, 13.3 edn

[XILINX'11d] XILINX: 'UG747 - Partial Reconfiguration Tutorial, PlanAhead software ', 2011

[XILINX'12a] http://www.xilinx.com/tools/designpreservation.htm, accessed 2012

[XILINX'12b] http://www.xilinx.com/tools/partial-reconfiguration.htm, accessed 2012

[XILINX'12c] XILINX: 'UG640 - System Generator for DSP User Guide', 2012c, 14.3 edn.

[XILINX'12d] http://www.xilinx.com/about/company-overview/index.htm, accessed 2012

[XILINX'12e] http://www.xilinx.com/products/silicon-devices/epp/zynq-7000/index.htm, accessed 2012

[Young'03] Young, S.A., P. Fewer, C. McMillan, S. Blodget, B. Levi, D. : 'A high I/O reconfigurable crossbar switch'. Proc. Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on, 2003, pp. 3-10

[Zicari'08] Zicari, P., Corsonello, P., and Perri, S.: 'A high flexible Early-Late Gate bit synchronizer in FPGA-based software defined radios'. Proc. Circuits and Systems for Communications, 2008. ECCSC 2008. 4th European Conference on, 2008, pp. 252-255

[Zong'08] Zong, W., and Arslan, T.: 'A low power reconfigurable heterogeneous architecture for a mobile SDR system'. Proc. ICECE Technology, 2008. FPT 2008. International Conference on, 2008, pp. 313-316

ANNEX 1

1. XILINX FPGA PROGRAMMING FRAME FORMAT

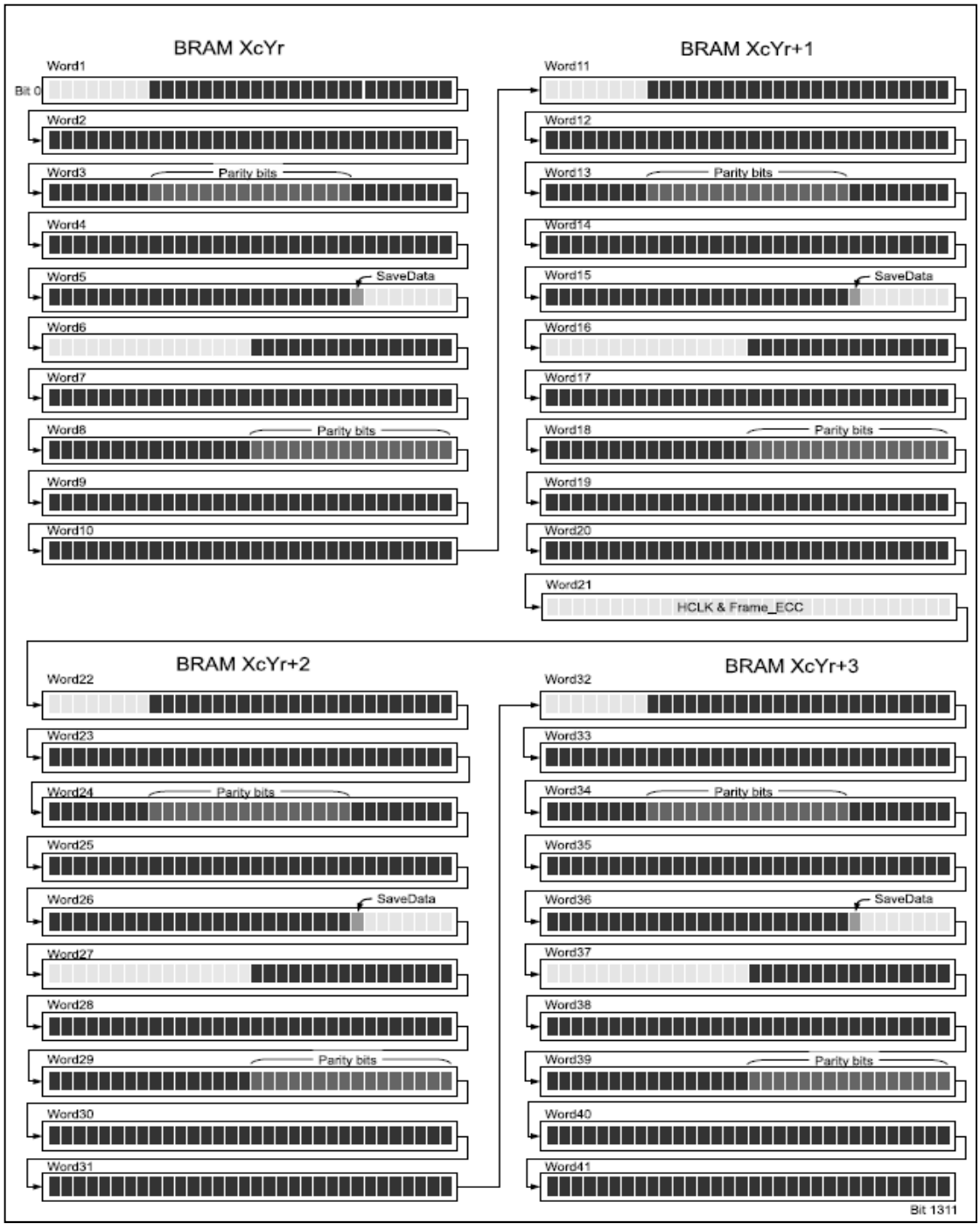


Figure 57: BRAM frame organization

The above image shows the internal structure of the bits present in a single BRAM content configuration frame. Please note that 64 complete frames are needed in order to configure the whole data in the BRAM. The image shows that each frame configures data into 4 BRAMS ("r" and "c"

represent the row and column of the BRAM in the FPGA). Each BRAM in a Virtex 4 FPGA contains 2 KB of user data storage, therefore, once the 64 frames have been downloaded, 8 KB (4 BRAMs x 2 KB each) of data are available. The difference between the downloaded data: 10.25 KB (64 frames x 1312 bits each) and the available user data (8 KB) resides in the extra configuration bits in the frame. To sum up the frame is comprised of: Real Data (bits coloured in black), Parity Bits (dark grey), SaveData bit, HCLK and Frame_ECC bits and Reserved bits (light grey).

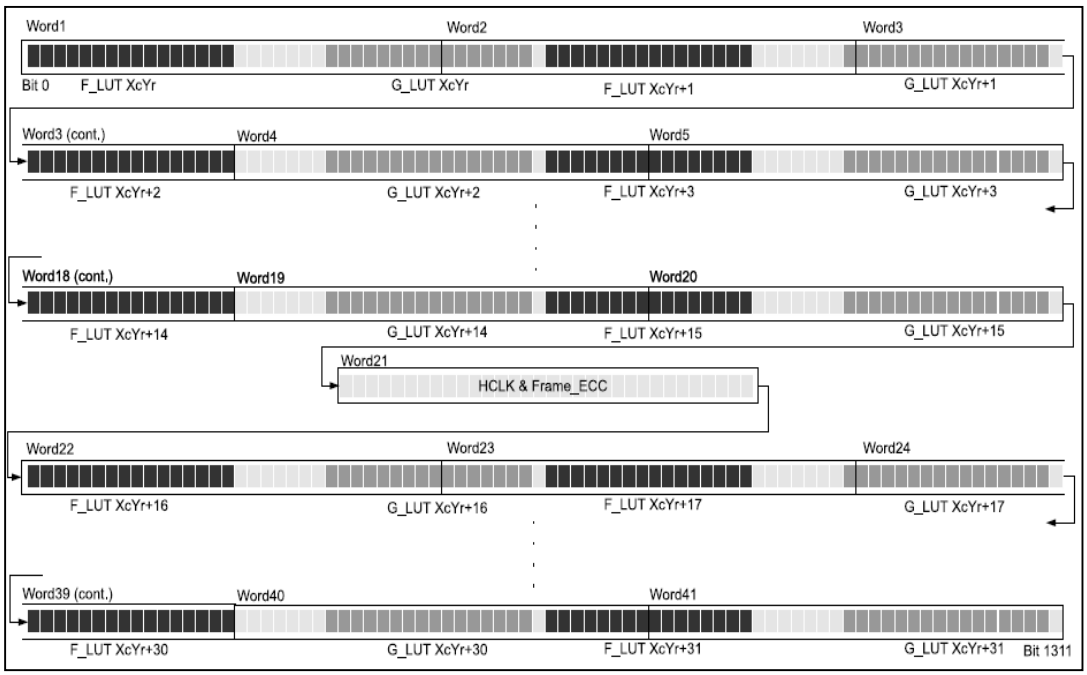


Figure 58: RAM LUT organization

Figure 58 shows the internal frame organization of a RAM LUT. In this case each frame configures a complete number of LUTs so it is possible to write just a single frame. Similarly to the BRAM frames, "r" and "c" represent the row and column that indicates the placement of the CLB that contains the LUTs within the FPGA. Taking into account that each CLB contains two LUTs they are noted "F" and "G". Each LUT is able to store 16 bits of information and each frame configures 64 LUTs. The frame is comprised of Real Data (bits coloured in black for F_LUT and dark grey for G_LUT), HCLK and Frame_ECC bits and Reserved bits (light grey).

2. BITSTREAM SIZE ESTIMATION FROM SLICE OCCUPATION

This section will detail the steps that have been carried out in order to infer the size of a partial bitstream from the number of occupied SLICEs of the corresponding reconfigurable area. That is, the design process of estimation function (14) will be presented.

The basis of this estimation is the size and influence of a single frame. The previous section on this annex has already introduced the size of a frame in a Virtex 4 FPGA: 1312 bits. Regarding the influence, a frame is the smallest addressable element in a bitstream. It contains part of the configuration data of the resources present on a certain column of the FPGA, covering a height of a single clock region. That is, even if a reconfigurable area does not span the whole height of the clock region, the complete frame needs to be downloaded. The full configuration of a certain resource needs the download of several frames (e.g. IOB configuration: 30 frames, CLB configuration: 22 frames, BRAM data configuration: 64 frames...etc. [XILINX'08c]). On this basis, analysing the number of resources that need to be configured within a reconfigurable area, the number of needed frames can be inferred and finally the bitstream size calculated. However, taking into account that the only available information is the number of occupied SLICEs, some simplifications are necessary. Figure 59 shows a reconfigurable area defined in Plan Ahead in order to ease this explanation:

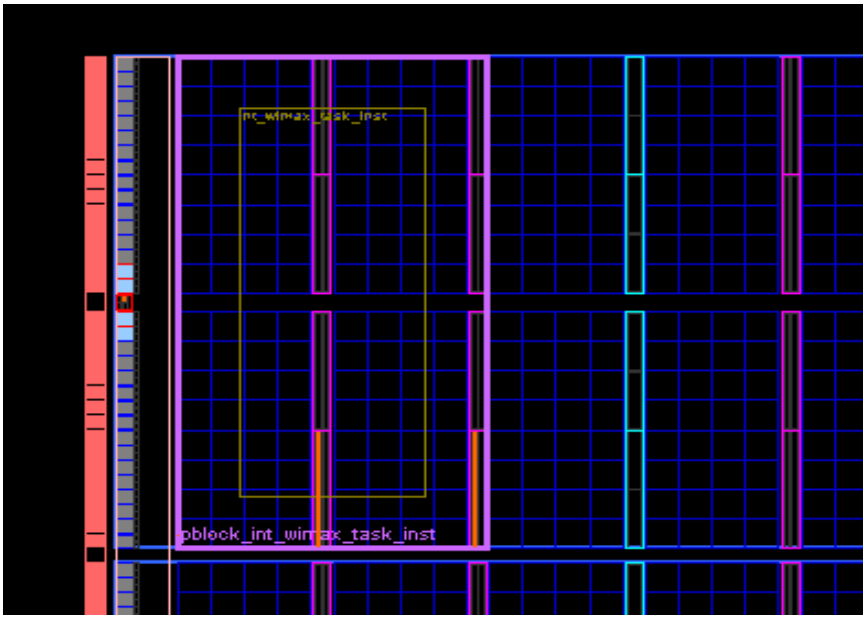


Figure 59: Reconfigurable area in Plan Ahead

Initially, it should be noted that at the time of defining the size of a reconfigurable area, the minimum addressable granularity is the CLB. That is, the small blue squares that can be observed in Figure 59. Besides, each CLB is made up of 4 SLICEs as can be observed in Figure 60.

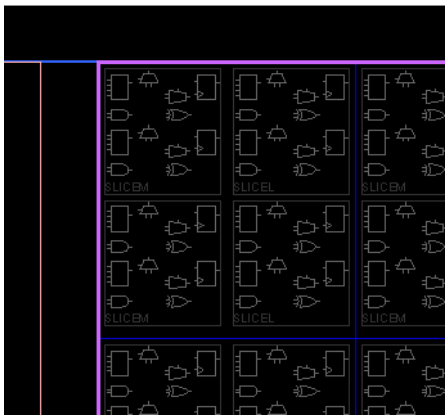


Figure 60: CLB detail

Consequently, we first assume that dividing by 4 the number of SLICEs we can obtain the number of CLBs. However, before carrying out this calculation it is necessary to establish the relation between the number of occupied SLICEs and the number of SLICEs in the reconfigurable area. Bearing in mind that we assume an occupation of the reconfigurable area of a 70%, the number of SLICEs in the reconfigurable area is:

$$SLICEs\ in\ PRA = Occupied\ SLICEs \cdot \frac{1}{Occupation} = Occupied\ SLICEs \cdot \frac{1}{0.7} \tag{19}$$

Substituting the aforementioned relation of 4 SLICEs per CLB, the number of CLB is:

$$CLBs\ in\ PRA = \frac{SLICEs\ in\ PRA}{4} = \frac{Occupied\ SLICEs}{0.7 \cdot 4} \tag{20}$$

Considering that each CLB column requires 22 frames of data to be configured, it is necessary to determine the number of CLB columns. Taking into account that the reconfigurable areas have a rectangular shape:

$$CLBs\ columns = \frac{CLBs\ in\ PRA}{CLB\ height} \tag{21}$$

The optimum PRA shape has to have a height of a complete clock region, as that is the span of each frame. Consequently, we assume that the PRAs have this shape. Virtex 4 FPGAs have clock regions with a height of 16 CLBs. Then:

$$CLBs\ columns = \frac{CLBs\ in\ PRA}{16} = \frac{Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16} \tag{22}$$

Replacing the aforementioned relation of 22 frames per CLB column:

$$CLBs\ frames = 22 \cdot CLBs\ columns = \frac{22 \cdot Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16} \tag{23}$$

However, as can be observed in Figure 59, there are other resources within the PRA that also need to be reconfigured. Considering the type of functions to be implemented, these resources are usually BRAMs (64 configuration frames) or DSP blocks (21 configuration frames). Figure 59 shows that there is one of these resources every 4 CLB columns. Consequently:

$$Other\ resources = \frac{CLBs\ columns}{4} = \frac{Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16 \cdot 4} \tag{24}$$

Considering the worst case, that is, the presence of a BRAM, the corresponding frames need to be calculated:

$$Other\ resources'frames = 64 \cdot Other\ resources = \frac{64 \cdot Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16 \cdot 4} \tag{25}$$

Adding both parts:

$$Frames = CLB\ frames + Other\ resources'frames$$

$$Frames = \frac{22 \cdot Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16} + \frac{64 \cdot Occupied\ SLICEs}{0.7 \cdot 4 \cdot 16 \cdot 4} = 0{,}8482 \cdot Occupied\ SLICEs \tag{26}$$

Besides, it is necessary to apply a correction factor of 1,2 in order to compensate the presence of other type of frames such as clock frames or the inclusion of header information in the bitstream. Definitely, including the correction factor and replacing the size of each frame, the estimation function in Kilobytes (please note the 1/8 and 1/1024) is:

$$Size = 1312 \cdot \frac{1}{8} \cdot \frac{1}{1024} \cdot Frames \cdot 1{,}2 = 0{,}8482 \cdot Occupied\ SLICEs \cdot 1312 \cdot \frac{1}{8} \cdot \frac{1}{1024} \cdot 1{,}2$$

$$Size = 0{,}163 \cdot Occupied\ SLICEs \tag{27}$$

3. "FUNCTION COMMONALITY LIST" TEMPLATE

Table 19: Function commonality list template