Thesis:

Evaluation of the Implementation of Partial Reconfiguration Projects using OpenPR on the Xilinx development board ML507

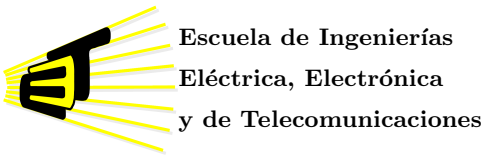
Submitted to:

Comité Asesor de Pregrado E^{3}T

By:

Dorfell Leonardo Parra Prada

This thesis was developing under the cooperation agreement UIS-ICP 005 of 2003



Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2012

Thesis:

Evaluation of the Implementation of Partial Reconfiguration Projects using OpenPR on the Xilinx development board ML507

Submitted to:

Comité Asesor de Pregrado E^{3}T

By:

Dorfell Leonardo Parra Prada

Advisor:

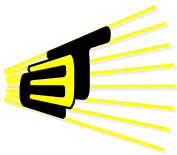
MSc.William Salamanca Becerra

Co-Advisor:

Ing. Carlos Angulo Julio

This thesis was developing under the cooperation agreement UIS-ICP 005 of 2003
in partial fullfilment of the requirements for the degree of

Electronic Engineer



Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2012

Contents

Universidad Industrial de Santander

List of Figures

ACREDITACIÓN INSTITUCIONAL
RES. MEN 2019 – JUNIO 05

Universidad Industrial de Santander

List of Tables

RESUMEN

Titulo:

Evaluation of the Implementation of Partial Reconfiguration Projects using OpenPR on the Xilinx development board ML507 [1]

Autor: Dorfell Leonardo Parra Prada [2]

Palabras Claves: OpenPR, Open-source, Reconfiguración Parcial, FPGA

Este documento presenta la evaluación de la implementación de projectos de reconfiguración parcial usando OpenPR- Una alternativa toolkit código abierto para la reconfiguración parcial- en el sistema desarrollo ML507.

Herramientas de software para la reconfiguración parcial como JBits, Torc y el Toolkit de Xilinx para la reconfiguracion parcial son brevemente mencionadas, así como algunas de sus mÃ¡s importantes carácteristicas.
Después OpenPR es introducido, su flujo de diseño, proceso de instalación incluyendo librerias y herramientas necesarias-GCC, Bison, Boost, Flex-, como obtenerlas y la compilación del código fuente son mostradas.
Los resultados del proyecto ejemplo Counter Project en el repositorio de OpenPR son reproducidos, y un nuevo proyecto para un dispositivo diferente - v5fx70t- es creado. Los archivos de configuración para los diseños estático y dinámico son generados e implementados en el sistema ML507. También se realizá la revisión de los archivos de descripción nativa del circuito en FPGA EDITOR- Una herramienta de enrutamiento de Xilinx- para verificar la correcta ubicación de los busmacros en el diseño.

Luego se desarrolla la etapa de depuración en el chip usando Chipscope para capturar señales dentro de la FPGA en cada componente del diseño- registros y busmacros-. Observaciones son realizadas en base a estas pruebas. Finalmente, conclusiones de todo el proceso y de las observaciones conforman esta evaluación.

[1] Proyecto de grado

[2] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: M.Sc. William A. Salamanca Becerra. Codirector: Ing. Carlos Angulo Julio.

ABSTRACT

TITLE:

Evaluation of the Implementation of Partial Reconfiguration Projects using OpenPR on the Xilinx development board ML507 [1]

AUTHOR: Dorfell Leonardo Parra Prada [2]

KEYWORDS OpenPR, Open-Source, Partial Reconfiguration, FPGA.

This paper demonstrates the evaluation of partial reconfiguration projects implementation using OpenPR -an Open-source partial reconfiguration toolkit alternative-, on the Xilinx embedded system development board ML507.

Partial Reconfiguration software tools like JBits, Torc and the Xilinx Partial Reconfiguration Toolkit are briefly mentioned, such as some of their most important features.

Then OpenPR is introduced, its design flow , instalation process including libraries and tools needed -GCC, Bison, Boost, Flex-, how to get them and OpenPR source code compilation are shown.
Results of the OpenPR repository counter example project are reproduced, and a new project targetting a different FPGA device, the v5fx70t, is created. Configuration files for static and partial design are generated and implemented in the ML507, getting a non-proper working in the board. This led to the native circuit description files check in FPGA EDITOR -a Xilinx FPGA routing tool- to verify the correct placement of the busmacros into the design.

Then debugging on-chip stage is done using Chipscope to test the signals state inside the FPGA in each design component as registers and busmacros. Observations are made based on these tests. Finally, conclusions of all the observation process are integrated into this evaluation.

[1] Thesis

[2] Faculty of Physics Mechanics Engineering. Electrical, Electronics Engineering and Telecommunications School. Director: M.Sc. William A. Salamanca Becerra. Codirector: Ing. Carlos Angulo Julio.

Evaluation of the Implementation of Partial Reconfiguration Projects using OpenPR on the Xilinx development board ML507

Dorfell L. Parra Prada, *CPS UIS*

Abstract—This paper demonstrates the evaluation of partial reconfiguration projects implementation using OpenPR -an Open-source partial reconfiguration toolkit alternative-, on the Xilinx embedded system development board ML507. The design flow, instalation process and project creation are also shown. Then debugging on-chip stage is made to verify the work of the tool. Observations and conclusions are integrated into this evalution.

Index Terms—OpenPR, Open-Source,Partial Reconfiguration, FPGA.

I Introduction

FPGAs are integrated circuits used to design and implement logical functions through a Hardware Description Language (HDL) [1]. Figure 1 shows a Xilinx Virtex 5 FPGA.

Some FPGA applications are digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection [1] and almost every area that use a digital circuit.

FPGAs internal structure consists of programmable logic components called Configurable Logic Blocks (CLB). Connections between them allow the building of digital circuits varying from logic gates to complex functions. In figure 2 an array of CLBs and junction nodes are shown.

Actually there are other integrated circuits which can perform the same tasks as an FPGA.

Fig. 1: Xilinx Virtex 5 FPGA. Taken from [2]

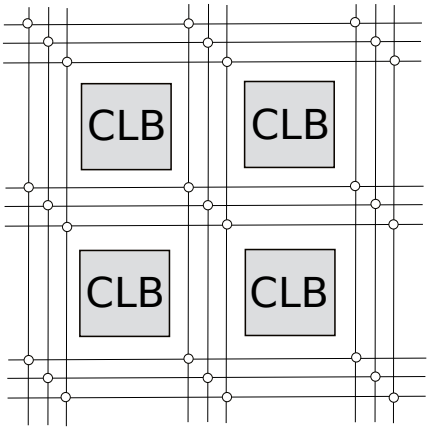


Fig. 2: Connection between CLB. Adapted from [3].

One of these is the Application-Specific Integrated Circuit (ASIC).

• *Dorfell L. Parra Prada is a member of the Connectivity and Processing of Signals Research Group of Electrical, Electronic engineering and Telecommunications School, Universidad Industrial de Santander UIS, Bucaramanga, Colombia, 680002.*
E-mail: dorfell.parra@correo.uis.edu.co

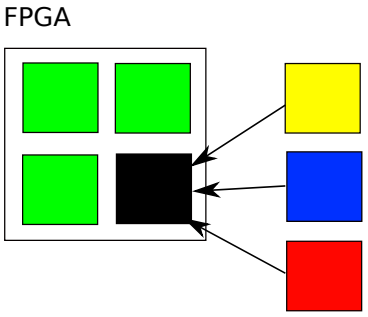


Fig. 3: Partial Reconfiguration Process.

ASICs are designed to execute particular processes. Once the circuit has been built, the designer can not change its configuration because it is hardwired. This causes higher working clock rates than an FPGA design in a smaller area chip, making ASICs cheaper, on a high production scale, than FPGA [6].
However ASICs designing, building and testing time is normally 6 to 12 weeks which is a disavantage when taking into account the easy in-field configurability and design flow of FPGAs [6].
Additionally, the emergence of new techniques such as partial reconfiguration has added other features to the FPGAs, spreading its use to fields such as the High Performance Reconfigurable Computing (HPRC).

Partial Reconfiguration is the process of configuring a part of a hardware circuit when it its still operating. [7].

In figure 3 partial reconfiguration implementation in an FPGA is shown. The green partitions of the FPGA are the fixed portion of the hardware circuit that remains without changes and it is called static region. The black box is the dynamic region (Reconfigurable Partition). The yellow, blue and red boxes implemented here, are the hardware modules, (Reconfigurable Modules); one of them each time.
The static region of the FPGA is configured first. Then, a partial bitstream (configuration file) with the information of the reconfigurable modules configures only the reconfigurable partitio,n without erasing the configuration of the static region, neither stopping the execution of any process in that region. Constraints definitions are required by partial reconfiguration implementations, since according to the region chosen the reconfigurable module will have access to specific resources in the FPGA.

In the last few years several software tools which allow the management of partial reconfiguration projects, have been developed. Some of these tools are: JBits that create Xilinx Virtex bitstreams form Java code, Torc, an open-source infrastructure and tool set for reconfigurable computing, Xilinx Partial Reconfiguration Toolkit which proposed a partial reconfiguration flow using PlanAhead and OpenPR, an open-source Partial Reconfiguration Toolkit for Xilinx FPGAs.

Private software tools such as the Xilinx Partial Reconfiguration Toolkit offers a friendly interface for the user, but its cost could be up to two thousand dollars and its source code is private. Open-source tools are in a continuing process of development and nowadays more users are working to expand their capabilities and improve their features.

Reviews of these tools, as well as performance benchmarks are needed by users who are interested in partial reconfiguration and don't want to waste time nor money.

This paper presents an evaluation of the implementation of partial reconfiguration projects using OpenPR in an Embedded System Development Board ML507 (in figure 4) starting with a brief summary of the partial reconfiguration flows and tools exposed in section II. Section III explains the OpenPR partial reconfiguration flow. The creation of a partial reconfiguration project process is in section IV. Debugging results for OpenPR partial reconfiguration project using Chipscope are exposed in section V. Finally conclusions and observations of this work are in section VI.

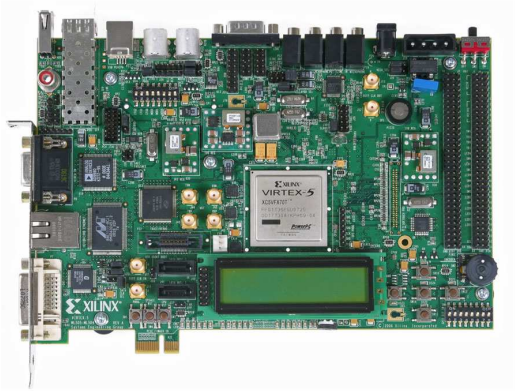


Fig. 4: Virtex-5 FX70T FPGA ML507 Evaluation Platform. Taken form [5].

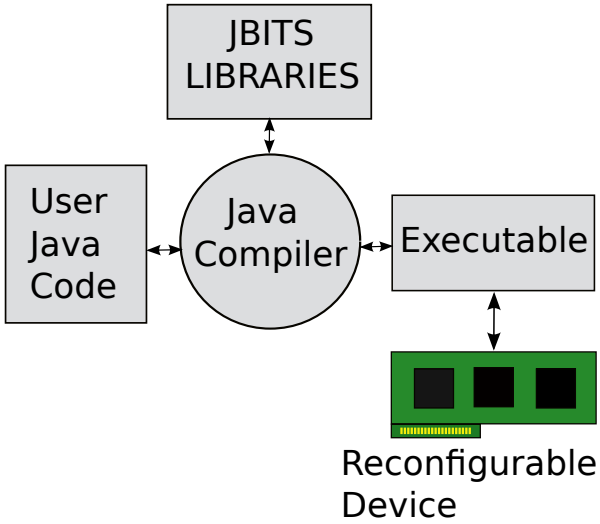


Fig. 5: JBits Design Flow. Adapted from [8]

II Related Work

It is known that every day the number of users working with partial reconfiguration is growing. This is shown by the increase in the development of software tools that allow management of partial reconfiguration projects. This section demonstrates some of the most influencial tools.

A) JBits

Xilinx Bitstream Interface (JBits) was a project supported by Xilinx that allowed users to create and modify Xilinx Virtex bitstreams from Java Code. The bitstream can be changed in a running FPGA or in a host computer. JBits provides an Application Program Interface (API) for a group of software tools.[14]
The flow design of JBits is shown in figure 5. Java code is entered by the user, then JBits libraries are used by the compiler to produce an executable file with the information needed to configure the Reconfigurable Device.
However JBits has some limitations. Most of the steps using JBits require manual intervention. The user has to define everything in the source code, including the routing [8]. Moreover, advanced knowledge of the FPGA architecture is needed. And in spite of the availability of Xilinx FPGAs documentation, users weren't conform with have to review so much background information in order to change a bitstream file, causing Xilinx to stop promoting the project. [15].

B) TORC [11]

Towards an Open-Source Tool Flow (Torc) is developed in C++. CAD tool development, architecture exploration and research applications are some of its applications. Torc reads, writes and modifies generic netlist (circuit description file, including connections and components), physical netlist, and bitstream packages.
Torc is based on ADB (A Standalone Wire Database for Routing and Tracing In Xilinx Virtex FPGAs), a collection of Xilinx Virtex FPGAs wiring databases.
Generic netlist API, Physical netlist API, device architecture API, and the bitstream API, in figure 6, are the databases that made Torc core functionality.

1) Generic Netlist

Netlist that are not mapped to physical primitives in a target device are supported by this API. The circuit design elements, such as libraries, cells, views, ports, instances and nets, can be accessed, queried, added, modified or removed. API also allows the creation

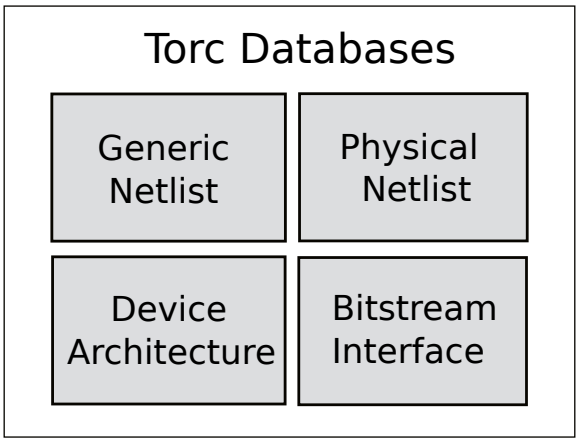


Fig. 6: Torc Databases. Adapted from [11]

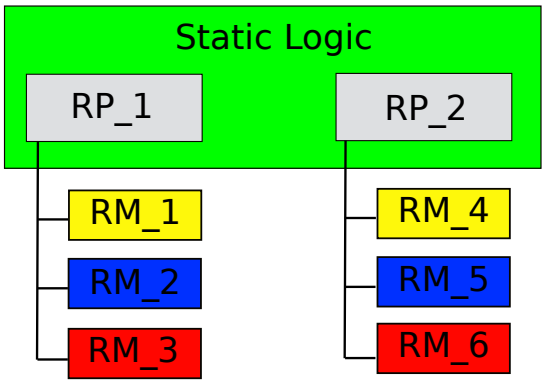


Fig. 7: Xilinx Partial Reconfiguration Design Hierarchy. Adapted from [13]

of new designs and provides bases for synthesis or mapping algorithms.

2) **Physical Netlist**

Netlists that have been mapped to physical primitives in the target device are supported by this API. Placement and routing information can be or not be in the physical netlist.
Reserving of device resources and regions can be made by Torc users, and also create and keep self-assertive routes. The bitstream generation step is performed directly, taking into account every change made to the design. These two reasons extend the design capabilities of the user.

3) **Device Architecture**

Device logic, and wiring information are provide by the Device Architecture API. The state of the resources, nodes, wires and uses inform the routers by this API.

4) **Bitstream Interface**

Read, write and modify bitstream packets are allowed by this API. Actually, this API supports Xilinx Architectures. But with the proper documentation, it can be updated to read, write and modify Altera architectures.

C) **Xilinx Partial Reconfiguration Toolkit**

The Xilinx Partial Reconfiguration Toolkit is a non-free add-on that allows designers to work with reconfiguration projects. In figure 7, the structure of a project is shown. The green region represents the static region of the design in the FPGA. Note that the two gray boxes are the reconfigurable partitions (RP). These partitions are defined by the designers, based on the size and kind of resources needed for the reconfigurables modules (RM), represented by the yellow, blue, and red boxes in the figure [13].
According to that, the project directory structure is:

- **Static Logic**
- **Dynamic Logic**

↦ RP_1.....Reconfigurable Partition.
→ RM_1.....Reconfigurable Module.
→ RM_2.....Reconfigurable Module.
→ RM_3.....Reconfigurable Module.
↦ RP_2.....Reconfigurable Partition.
→ RM_4.....Reconfigurable Module.
→ RM_5.....Reconfigurable Module.
→ RM_6.....Reconfigurable Module.

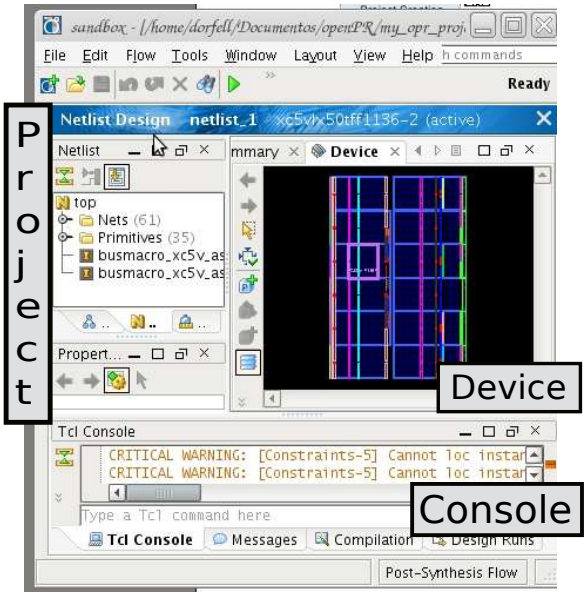


Fig. 8: PlanAhead Window.

Xilinx Partial Reconfiguration Toolkit uses PlanAhead to manage the Partial Reconfiguration Projects.

PlanAhead is a software tool offered by Xilinx that allows designers to manage projects from register-transfer level (RTL) to bitstream designs. A user friendly interface, and access to other software tools such as ISE Project Navigator is also offered.
In figure 8, PlanAhead interface is shown. Project directories, debug console, and FPGA structure graphical description are some of the basic features in PlanAhead. Constraint definition, project implementation, and bitstream generation are among its most useful functions.

Private add-ons like the Xilinx Partial Reconfiguration Toolkit are avalaible in PlanAhead through the Set Partition Wizard.
PlanAhead is a paid software tool but users can download a free evaluation version from its webpage.
The Xilinx Partial Reconfiguration flow is shown in figure 9.

The first step in the process is the project creation in PlanAhead. Here the user must import the netlist file for the static design. PlanAhead will load the device information and will open the FPGA architecture graphics. Then reconfigurable partitions will be created by the designer, using the setting partition wizard, which allows netlists to be added for the reconfigurables modules.

It's time to draw. Drawing of the reconfigurable partition is done based on the amount and kind of resources needed by the reconfigurable modules. After that, the user can save the constraints file and compile the project. This is known as the implementing or importing stage of the design in PlanAhead.

Static and partial bitstreams generation are the last steps in the flow. Impact (a Xilinx FPGA programming tool) can be used to configure the FPGA [13].

In the next section, OpenPR, a non-paid partial reconfiguration tool, is presented.

III THE OPENPR PROCESS [15]

OpenPR is an Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs, presented by its designers in 2011. The main difference in the Xilinx Partial Reconfiguration Toolkit relies in the nature of its code, since it is an open-source.
Actually OpenPR only works for Xilinx FPGAs. But its functionality can be expanded to Altera FPGAs with the proper documentation.
This section briefly describes the partial reconfiguration flow using OpenPR demonstrated by the authors in [15]. Basic partial reconfiguration concepts are kept in OpenPR. Its flow for creation of partial reconfiguration projects is shown in figure 11.

The dynamic region here represents the reconfigurable partitions proposed in the Xilinx Partial Reconfiguration Designs. The

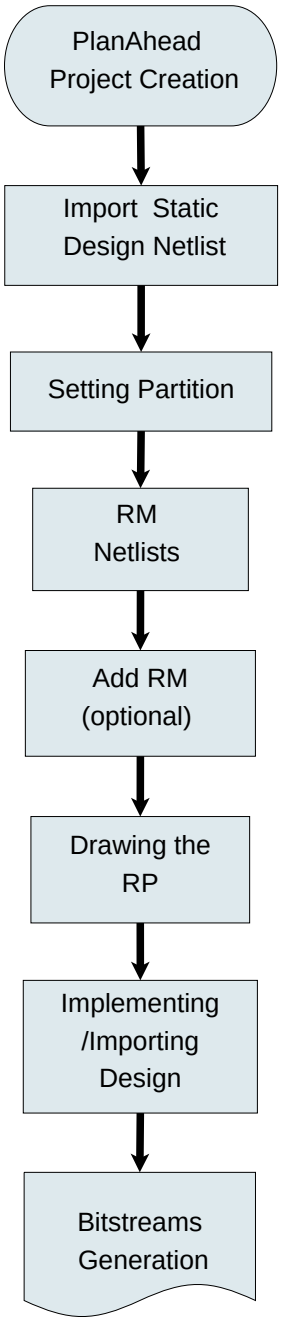


Fig. 9: Xilinx Partial Reconfiguration flow using PlanAhead.

reconfigurable modules are now known as the partial modules (PM).

OpenPR aims to be an easy and little manual intervention requiered tool. In Figure 11 the partial reconfiguration flow is shown.

The process is divided into the four main stages described below.

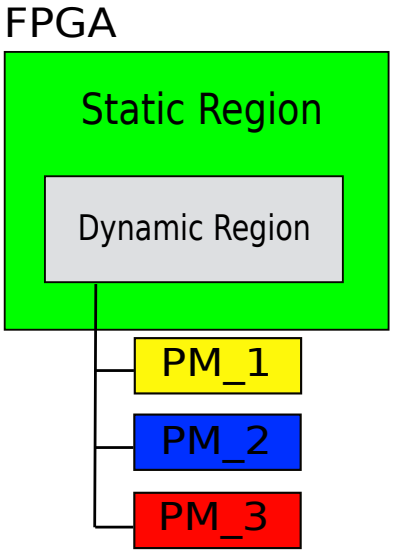


Fig. 10: OpenPR Partial Reconfiguration Design.

A) **OpenPR XML Project File**

The first step is to create the OpenPR XML project file, which has the project information needed by OpenPR. Some important parameters such as the *busMacroPrefix* prefix for bus macro, in this file are listed in table 1.
Bus macros are templates that allow OpenPR to connect partial and static design as shown in figure 12.
An example of the OpenPR XML project file can be found in the OpenPR repositories [15].

B) **Floorplaning**

In this stage, the designer can use PlanAhead to draw the dynamic region, create an area group and export the constraints to the OpenPR user constraints file (ucf), which contain all the design restrictions. The area group must have the same name as the *dynamicAGName* parameter in the XML project file. This process is called floorplanning.

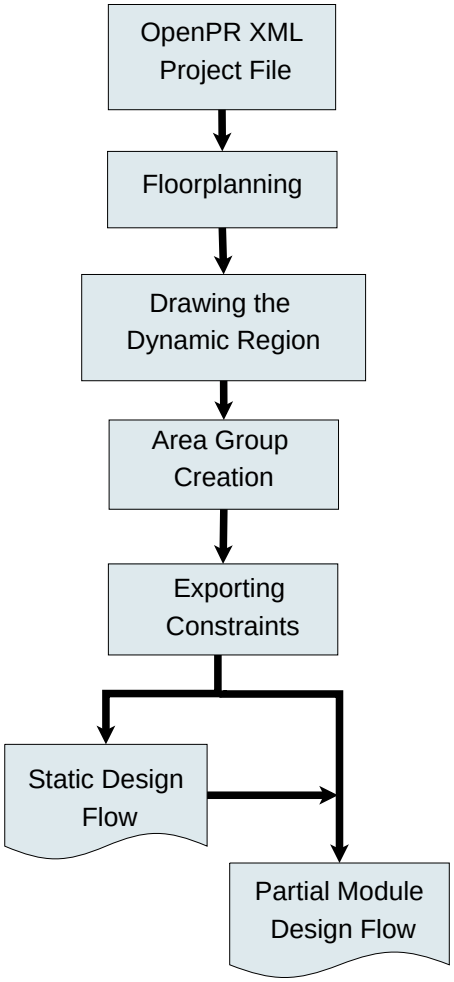


Fig. 11: Overview of the OpenPR design process.

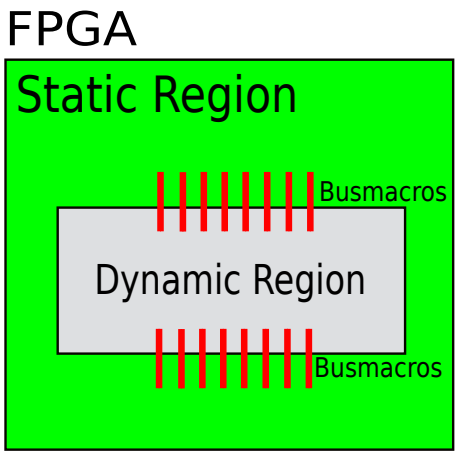


Fig. 12: Static and Dynamic Region connection using bus macros. Adapted from [10].

Parameter	Description
designName	Top-level design entity.
staticPath	Full path static design directory.
partialPath	Full path partial design directory.
isPArtial	Boolean value indicating whether project is a sandbox or a partial module.
ucfPath	Full path to UCF file with pin/timing constraints for static design.
dynamicAGName	Name of dynamic region AREA_GROUP as defined in UCF file.
busMacroPrefix	Naming prefix for bus macro defining target architecture and bus macro type.
passThrougNet Name	For sandbox, identify nets passing through dynamic region; for patial design, identifiy nets connecting virtual IO pins to dynamic region inputs/outputs
clkNetNames	Collection of clock nets in the design.
busWidth	Witdh of dynamic region datapath.
deviceName	Full device name.
regionDefined	Boolean value indicating whether OpenPR region is defined yet.
x/y Min/Max	Vertices of dynamic region.

TABLE 1: OpenPR Project File Parameters. Adaptep from [15]

C) Static Design Flow

Every project in OpenPR has a Makefile for the static directory and a Makefile for the partial directory. These makefiles run a script with the routines needed to execute the partial reconfiguration. Some of these routines are to synthesize, translate, map, place & route and bitstream generations. The map and place & route process has as an output, a Native Circuit Description (NCD) file, where the physical description of the design is made [17]. The bitstream generation process has to output the bit files to configure the static and partial design in the FPGA.

Once the constraints defined in the floorplanning stage have been added to the ucf file, it's time to use the Makefile. The designer can create the bit file for the static design with the command: *make static*.

D) Partial Module Design Flow

The partial directory has a similar static design Makefile, but in the XML project file the *isPartial* parameter is set to 1, to indicate that the directory contains a partial design. This makefile must be used after succesfully running the static design makefile to compile the files for the partial design.
Partial bit file is created with the command: *make partial*.

IV Creating Reconfiguration Projects with OpenPR

This section demostrates the tracking of the OpenPR Partial Reconfiguration Project flow using the example project Counter, which is an OpenPR project example for Virtex 5 LX FPGAs. Then, a new project is created to evaluate the OpenPR capabilites.

A) The Counter example

The Counter example is available from OpenPR repository, which has Documentation, source-code and OpenPR example projects. OpenPR projects have a directory structure shown in figure 13.

Static design files are in a Sandbox folder, partial designs are in partial module (PM) folders.
There are OpenPR Project files in both the Sandbox and PM folders with the *isPartial* parameter defined to indicate OpenPR whether the folder has a Static or a partial design.
Hardware description language (*.vhd) files have design descriptions and are used to synthesize the digital circuits. Physical constraints are in an ucf file. Routines needed to create bit files for the static and partial design are in the Makefile. A precompiled folder has bus macros templates.

- **Sandbox**
  - ↦ Makefile.....Static Design Makefile.
  - ↦ top.proj.....OpenPR Sandbox Project File.
  - ↦ src\
    - → sandbox.vhd.....Static Design Top Level.
    - → system.ucf
    - → precompiled\
      - ↪ busmacro_*.....Pre-built Bus Macros.
- **PM_0**
  - ↦ Makefile.....Partial Module Makefile.
  - ↦ top.proj.....OpenPR PM Project File.
  - ↦ src\
    - → sandbox.vhd.....PM Top Level with Bus Macros.
    - → prm.vhd.....PM Implementation.
    - → system.ucf
    - → precompiled\
      - ↪ busmacro_*.....Pre-built Bus Macros.
- **PM_1**
- **PM_2**

Fig. 13: OpenPR Partial Reconfiguration Projects Directory Structure. Taken from [10].

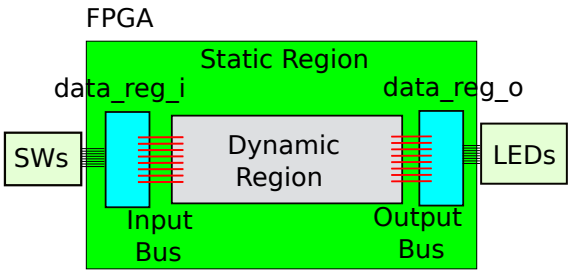


Fig. 14: Counter Example Circuit.

The counter example circuit is shown in figure 14.
Data_reg_i and data_reg_o are the dedicated registers to the LEDs and SWs peripherals. Bus macros, in red, allow the connection of the dynamic region with the static design.

The partial modules names in the Counter example are passthrough, increment and decrement. In the passthrough module the SWs state is shown in the LEDs. Increment

Partial Module	Description	Implementation
Passthrough	Input-Output bus macros direct connection	Leds show SWs state
Increment	Ascending counter	Leds blinking
Decrement	Descending counter	Leds blinking

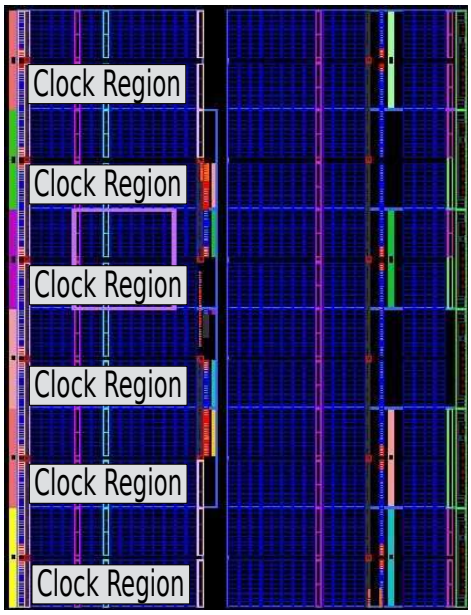
TABLE 2: Partial Modules Description.

and decrement modules are counters with their first value given by the SWs state. Table 2 summarizes their behaviour.
Dynamic region location can be made using PlanAhead.
In figure 15 a) the architecture of a Virtex 5 LX series FPGA is shown. The rows in the FPGA are known as clock regions. There are 6 of them.
Figure 15 b) is a detailed view of how FPGA logic resources like: Slices -in blue-, Ram block (BRAM) -in Red-, and Digital Signal Processing (DSP) -in green-, are organized in columns of the FPGA. A slice is a set of 4 CLB.
Indications of specific places in the FPGA are made using slices coordinates. (i.e. SLICEX0Y0, where X and Y are the positions of the slice) [18].

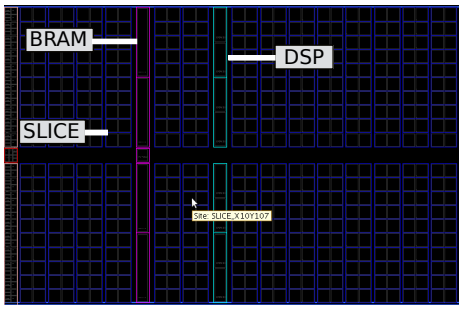
The area group selected for the counter project is shown in figure 15 c). The constraints for this region were:

```
AREA_GROUP "counter"
RANGE=SLICE_X8Y60:SLICE_X23Y79;

AREA_GROUP "counter"
RANGE=DSP48_X0Y24:DSP48_X0Y31;

AREA_GROUP "counter"
RANGE=RAMB36_X0Y12:RAMB36_X0Y15;
```

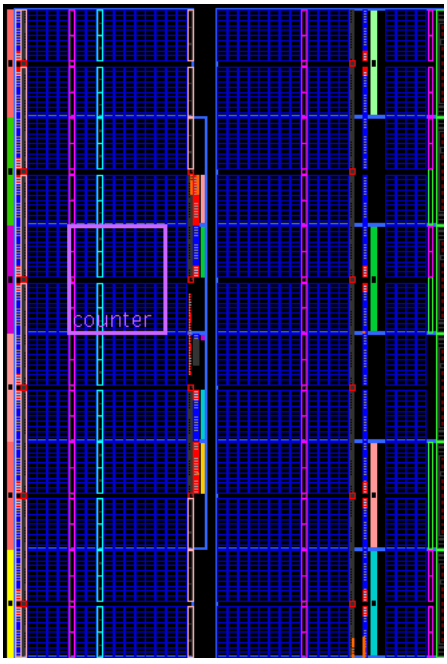
FPGA_editor is a Xilinx Tool that allows the edition of the internal FPGA circuit routing [19]. Figure 16 shows the *.ncd file generated with the bus macros correctly placed. The blue rectangle is the FPGA and



(a)



(b)



(c)

Fig. 15: a) Virtex 5 LX FPGA architecture. b) FPGA resources. c) Area Group Counter selected.

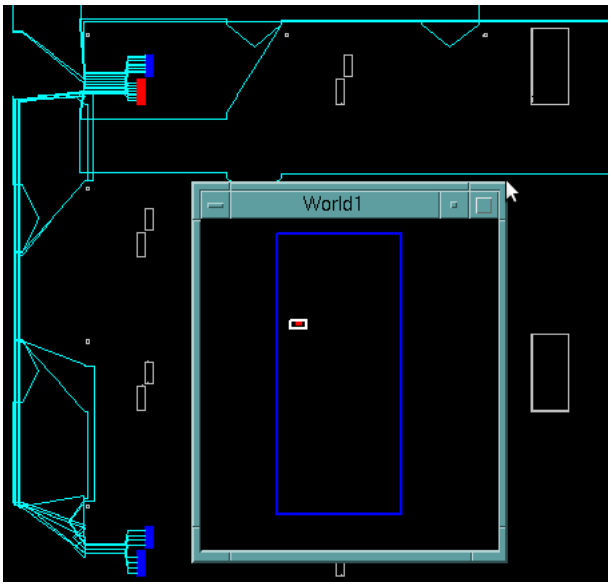


Fig. 16: Counter example bus macro position in the LX FPGA.

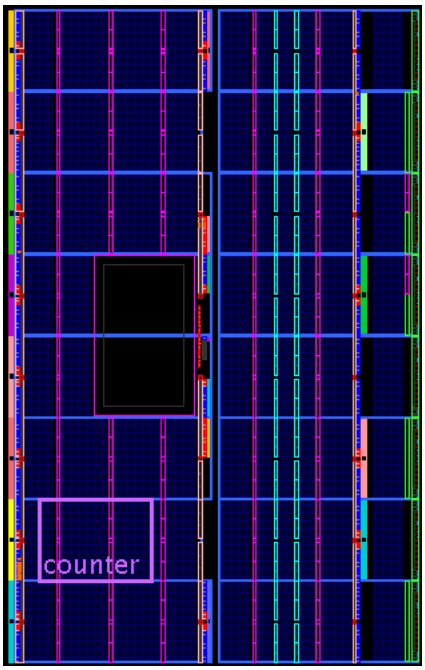


Fig. 17: Area Group Counter Selected for the New Project.

the red point is the Busmacro position.

Implementation of the project was made executing the Makefiles for the static and partial design. That process result in the succesfull generation of the static and partial bit file.

B) **Creating a New Project**

A counter example based project was created to evaluate the proper working of OpenPR in the ML507 development board. Makefiles were changed to use the ML507 FPGA as well as the place for the area_group. This new position of the area group is shown in figure 17, where the black box represents the PowerPC hardcore processor for this Virtex 5 FX FPGA. Constraints for this new position were:

```
AREA_GROUP "counter"
RANGE=SLICE_X4Y20:SLICE_X29Y39;

AREA_GROUP "counter"
RANGE=RAMB36_X0Y4:RAMB36_X1Y7;
```

After the floorplaning stage the implementation process produced some errors which are shown as well as the explanation of the tests done for the project in the next section.

V DEBUGGING IMPLEMENTATION OF PR PROJECT

The counter project for the ML507 development board created in the previous section was used to evaluate the proper operation of the OpenPR. Implementation process and an analysis of the results are presented in this section.

All the test made can be summarized in two cases with differents dynamic regions selected.

A) **Region 1**

The Area group counter in region 1 for the reconfigurable modules was placed in:

AREA_GROUP "counter"
RANGE=SLICE_X4Y20:SLICE_X29Y39;

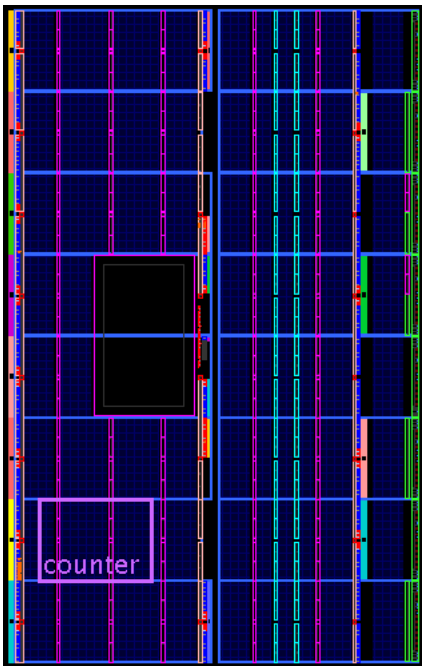
AREA_GROUP "counter"
RANGE=RAMB36_X0Y4:RAMB36_X1Y7;

as is shown in figure 18 a). The resources in this area group are magnified in figure 18 b). Then, the Makefile was executed, producing the mapping process error in figure 19.

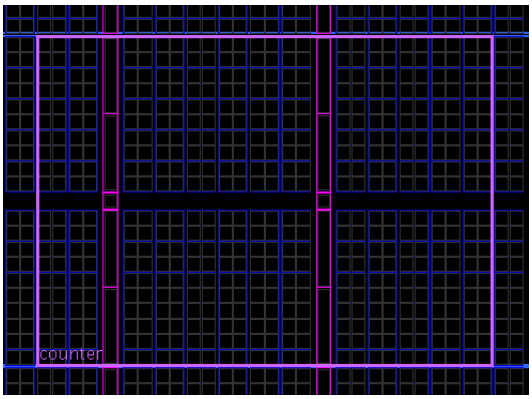
In the mapping report this error was repeated by each bus macro component of the Counter project. There are two possible causes:

- The first is that OpenPR assumes that the design requieres DSP resources, but the region doesn't have these resources, so it produces an error message saying that the other resources needed by the components are blocked; *"Some of the logic associated with this structure is locked"* in figure 19.

- The second reason is that OpenPR assumes the whole left side of the FPGA as a region, and this has a PowerPC hardcore processor, so the PowerPC resources are considered to be locked and OpenPR can not place the bus macros nor the other components of the project.



(a)



(b)

Fig. 18: a) Area group counter place in region 1. b) Area group counter resources in region 1.

B) **Region 2**

The area group counter in region 2 for the reconfigurable modules was placed in:

RANGE=SLICE_X40Y120:SLICE_X67Y139; Logic Resources
RANGE=DSP48_X0Y48:DSP48_X1Y55; DSP Resources
RANGE=RAMB36_X3Y24:RAMB36_X4Y27; RAM BLOCK

as it is shown in figure 20 a). A detailed

Phase 1.1 Initial Placement Analysis
ERROR:Place:346 - The components related to Macro Instance <busmacro_xc5v_async_vert_input_0_0> of the Macro Definition
<busmacro_xc5v_async_vert_input_0> can not be placed in the required relative placement form

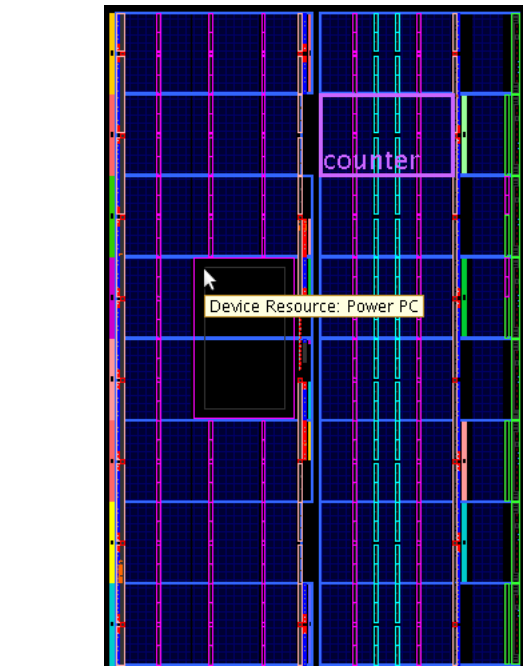
The following components are part of this structure:
LUT busmacro_xc5v_async_vert_input_0_0/$COMP_3, locked to site SLICE_X5Y39
LUT busmacro_xc5v_async_vert_input_0_0/$COMP_4
LUT busmacro_xc5v_async_vert_input_0_0/$COMP_5
LUT busmacro_xc5v_async_vert_input_0_0/$COMP_0

The reason for this issue is the following:
Some of the logic associated with this structure is locked. This should cause the rest of the logic to be locked. A problem was found at site SLICE_X4Y39 where we must place LUT busmacro_xc5v_async_vert_input_0_0/$COMP_0 in order to satisfy the relative placement requirements of this logic. This site appears to be prohibited.

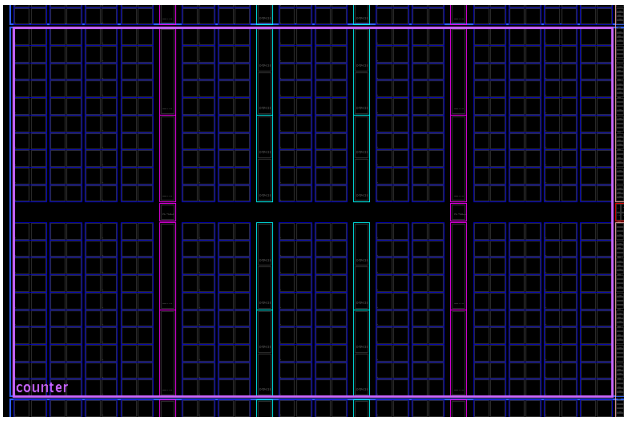
Fig. 19: Mapping Process Error Message.

view of the resources in that region is shown in figure 20 b), where it can be seen that the area group now has DSP resources. Placing the area group in region 2 allowed OpenPR to successfully finish the execution of the *Makefile*.
The *.ncd files for the sandbox and the passthrough partial modules were checked using *FPGA_editor*.
In figure 21 a) the input and output bus macros have been automatically placed by OpenPR in the boundaries of the reconfigurable region for the static design in sandbox. But in figure 21 b) the .ncd for the partial module Passthrough analized in FPGA_editor shows that the bus macros appear in another position different to the position seen before in the static design. Therefore, there is no match between the static design bus macro location and the dynamic design bus macro location.

After the .ncd files check, proof of the bit files generated was the next step. Configuration of the FPGA was made using the impact tool provided by Xilinx, downloading first the static bit file, and then the partial bit file. The bit files



(a)



(b)

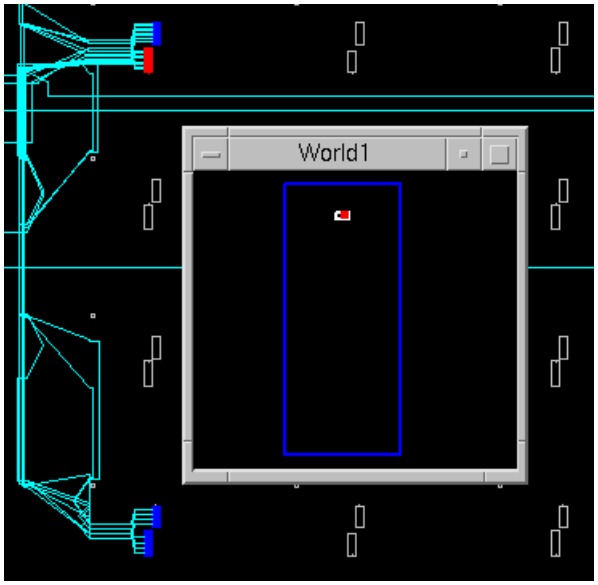
Fig. 20: a) Area group place in region 2. b) Area group resources in region 2.

Module	Bit file	Size
Sandbox	genesys_top.bit	3.2 [MB]
Passthrough	top_full.bit	3.2 [MB]
	top_partial.bit	86.5 [kB]

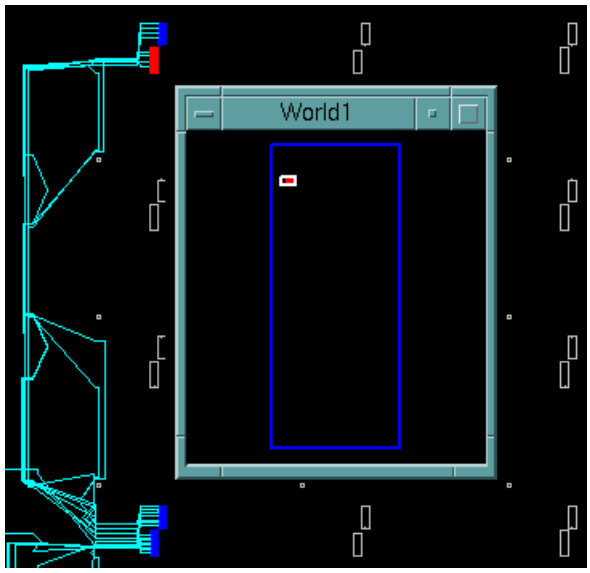
TABLE 3: Bit files generated

generated are listed in table 3. Partial bit file has a smaller size than the static bit file because the partial module circuit is smaller than static circuit.

The circuit was tested in the board. But



(a)



(b)

Fig. 21: Native Circuit Description for Static and Partial design; a) Bus macros structure and position for Sandbox. b) Bus macro structure and position for the PM Passthrough.

according to the behaviour listed in table 2, there wasn't a proper working of the example. This led to the debugging process.

C) **Debugging Process using ChipScope**

Parameters	Value
Input Netlist	top.ngc
Device Family	Virtex 5
Trigger Width	1
Match Units	1
Data Same As Trigger	Unchecked
Data Width	[0:31]
Data Depth	1024 Samples
Clock Signals	Clk_BUFGP
Trigger Signals	Clk_BUFGP
Data CH [0:7]	data_reg_i[0:7]
Data CH [8:15]	data_reg_o[0:7]
Data CH [16:23]	module_in[0:7]
Data CH [24:31]	module_out[0:7]

TABLE 4: ILA and ICON parameters

SWs	LEDs
0x00	0xFF

TABLE 5: SWs and LEDs state for Sandbox implementation.

ChipScope was used to get a better image of the behaviour of the circuit. ChipScope offers the ChipScope Pro Core Inserter to create IPcores as ILA, ICON, IBERT, etc, and the ChipScope Analyzer to show the signals that are being checked.

The generation of the IPcores ICON and ILA involved the parameters in table 4 which are related to the names in figure 14.

Data_reg_i and data_reg_o were associated to data channels [0:15], and the module_in and module_out, which are the signals that connect the input output registers to the busmacros, were associated with data channels [16:31].
Once the ILA and ICON core were inserted into the design, ChipScope Analyzer was used to inspect the signals.

Data Channel	Output
DataPort [0:7]	0x00
DataPort [8:15]	0xFF
DataPort [16:23]	0x00
DataPort [24:31]	0xFF

TABLE 6: Data Channel Outputs for Sandbox.

SWs	LEDs
0xFF	0xFF

TABLE 7: SWs and LEDs state for top_full.bit implementation.

Data Channel	Output
DataPort [0:7]	0xFF
DataPort [8:15]	0xFF
DataPort [16:23]	0xFF
DataPort [24:31]	0xFF

TABLE 8: Data Channel Outputs for Partial Module Passthrough.

Table 5 shows the leds and switch status for the sandbox implementation. The connections between SWs and the data_reg_i are working as seen in table 6. It is important to note that there is no connection between the data_reg_i and data_reg_o because any partial bit file has been downloaded to the FPGA.
Table 7, shows the behaviour of the implementation of the top_full bit file. It seems similar to the sandbox bit file described in table 5 , but as it is shown in table 8, all the signals have the boolean value 1 as output. Therefore, there are no connections between the data_reg_i and the dip switches.

In order to continue with the process the partial bitstream was downloaded to the FPGA, but instead of configuring just the area group counter, it changed the whole configuration of the FPGA, erasing the chipscope IPcores. In table 9 the SWs and LEDs status are shown.

VI CONCLUSIONS

This paper has demonstrated an evaluation of OpenPR working on the development board ML507. The evaluation process follows the flow design proposed by the authors in[10], including a debugging on-chip stage using Chipscope. The main conclusions of this work will be shown below.

SWs	LEDs
0xFF	0xC4

TABLE 9: SWs and LEDs state for top_partial.bit implementation.

The OpenPR partial reconfiguration design flow has been documented, giving a brief description of each stage of the process, making such it easy to compare other design flows such as the Xilinx Partial Reconfiguration Toolkit.

The installation process of libraries, compilers and databases needed by OpenPR, as well as the download and source code compilation are described in a step-by-step guide and added to the paper as an appendix.
Furthermore, The Partial Reconfiguration OpenPR project creation using the OpenPR examples and a review of the walk-through is demonstrated.

Although the implementation on the ML507 board was not succesful, the debug stage allowed to be made the following observations and hypothesis.

- The automatic location of the bus macros depends on the resources outside the partial region that are avaliable, OpenPR does not take into account the resources, when placed in the bus macros for the partial region.
  Because, as seen before, bus macros encountered logic resources blocked for the XC5V70T-ff1136-1 where the left chip architecture is different than the right, due to the PowerPC hardcore processor. The causes of this leak of information might be in the structure definition made in the Virtex5DeviceInfo file.

- Once that OpenPR has located the bus macros for the static region in sandbox, the location of the bus macros in the partial modules differs from the previous place chosen in the static design leading to mismatches in the implementation

of the FPGA partial configuration. A consequence of these mismatches is the loss of design parts.

- Partial bitstream generation is not being correctly made. It is shown by tests, that when a user downloads the partial bistream, part of the static design is changed. This issue is expected to be corrected in future OpenPR updates.

In spite of these bugs, OpenPR is free allowing more users to get involved in the process without paying for expensive tools. Its Open Source code admits to include others FPGA's that actually are not supported by private tools. OpenPR sets a strong alternative in FPGA reconfiguration tools, and its development process will be the key for the next improvements, like user graphics interfaces, and support for the Virtex 5 FX series.

APPENDIX A
WORKING WITH OPENPR

OpenPR needs some compilers, libraries, parser generators, lexic analyzer and other programs to work. Before starting to work with OpenPR, the Synaptic Package Manager can be used to install some of these programs.

In this section the programs needed, how to get that programs and the OpenPR installation process will be explained.

A brief review of each program follows:

1) **GCC-4.2.4 [20]**

   GCC, GNU Compiler Collection is developed by the GNU project. It includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages. G++ was originally written as the compiler for the GNU operating system.

2) **Boost [21]**

   Boost provides free peer-reviewed portable C++ source libraries. It contains more than eighty C++ libraries which expand the funcionality of the C++ language. Works well with the C++ standard Library. Boost's use speeds initial development, results in fewer bugs, reduces reinvention-of-the-wheel, and cuts long-term maintenance costs. And nowadays, many programmers are already familiar with them.

   To work with OpenPR Boost version it has to be 1.42 or similar and it must be installed in /usr/local/include/boost-1_42/ [15].

3) **Bison-2.3 [22]**

   Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1) parser tables. As an experimental feature, Bison can also generate IELR(1) or canonical LR(1) parser tables. Once you are proficient with Bison, you can use it to develop a wide range of language parsers, from those used in simple desk calculators to complex programming languages.

4) **Flex [23].**

   Flex is a tool for generating scanners. A scanner, sometimes called a tokenizer, is a program which recognizes lexical patterns in text.
   The flex program reads user-specified input files, or its standard input if no file names are given, for a description of a scanner to generate.

After the installation of the tools mentioned

before, the next step is to download an updated copy of OpenPR from its repository. It can be downloaded from:

https://openpr-vt.svn.sourceforge.net/svnroot/openpr-vt/trunk

The Linux terminal can be used for its download with the command:

≫ **svn co https://openpr-vt.svn.sourceforge.net/svnroot/openpr-vt/trunk trunk**

The repository directory structure should be:

trunk

- **doc**
  - ↪ refman.pdf.....Reference Manual.
- **res**
  - ↦ counter.....Project Example.
  - ↦ video_filters.....Project Example.
- **src**
  - ↦ devices.....devices databases.
  - ↦ openpr.....OpenPR source code.
  - ↦ torc.....Torc source code.
  - → anticore.....Anticore executable.
  - → Doxyfile.....Documentation File.
  - → Makefile.....OpenPR Makefile.

Use the /usr/Makefile file to build openPR typing in a terminal:

≫ **make**

Then install the anticore executable in /usr/local/bin, and the TORC devices folder to /usr/local/share/openpr/devices using the following command.

≫ **make install**

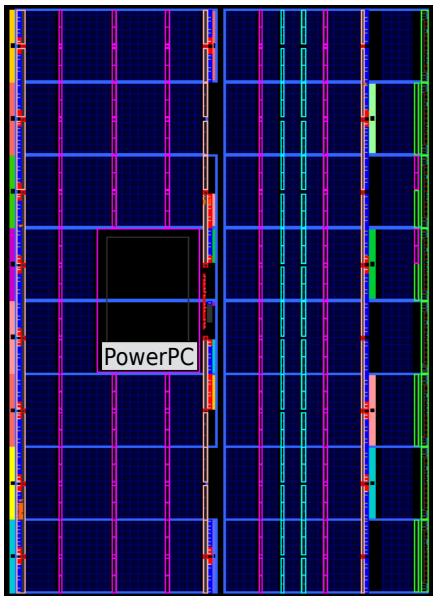


Fig. 22: Virtex 5 FX FPGA architecture.

Finally add the anticore to the working path, writing the next lines to the .bashrc file;

"export TORC=/usr/local/share/openpr"

OpenPR is now installed in the computer.

APPENDIX B
ADDING A SPECIFIC FPGA

OpenPR was developed for work with some Virtex 5 FPGAs. One of its strongest features is its open source-code. It allows the addition of new FPGAs to OpenPR modifying some files. Figure 22 shows the structure of the ML507 system FPGA, the xc5vfx70t-ff1136-1 which belongs to the Virtex 5 FX series. The differences between the LX and the FX series are the amount of logic resources available and the implementation of a PowerPC hardcore processor in the last one.

The steps needed to include the xc5vfx70t FPGA in OpenPR are:

- In **trunk/scr/openpr/bitstream/**:

A user has to write the following lines in the v5_devices.h file. These lines define the bitstream id and row_layout structure of the FPGA;

```
class xc5vfx70t:public virtex5 {
protected:
enum
{ xc5vfx70t_num_rows = 8, xc5vfx70t_num_cols = 57, };
static const tile_types xc5vfx70t_row_layout[];
static const string xc5vfx70t_name;
static const int xc5vfx70t_id;
public:
xc5vfx70t():virtex5(xc5vfx70t_num_rows,
xc5vfx70t_num_cols,
xc5vfx70t_row_layout, xc5vfx70t_name, xc5vfx70t_id)
{build_xdl_layout();};
};
```

Edition of the bitstream.cpp file is needed to indicate the FPGA device and initialize the variable my_dev adding;

```
else if (device_name == "xc5vfx70t")
my_dev = new xc5vfx70t();
```

- In **trunk/scr/torc/bitstream/**

The Virtex5DeviceInfo file must have the FPGA structure information. If that's not the case, add the following lines to the file.

```
// xc5vfx70t static device information.
boost::uint32_t xc5vfx70tColumns[] =
{ Iob, Empty, Empty, Empty, Empty, Empty, Clb, Empty,
Clb, Empty, Empty, Clb, Empty, Clb, Empty, Empty,
Bram, Empty, Clb, Empty, Clb, Empty, Empty, Clb,
Empty, Clb, Empty, Clb, Empty, Clb, Empty, Empty,
Empty, Bram, Empty, Clb, Empty, Clb, Empty, Clb,
Empty, Clb, Empty, Empty, Clb, Empty, Clb, Empty,
Empty, Bram, Empty, Clb, Empty, Clb, Empty, Empty,
Clb, Empty, Clb, Empty, Empty, Empty, Empty, Empty,
Iob, Clock, Empty, Empty, Clb, Empty, Clb, Empty, Clb,
Empty, Clb, Empty, Empty, Empty, Bram, Empty, Clb,
Empty, Clb, Empty, Empty, Dsp, Empty, Empty, Clb,
Empty, Clb, Empty, Empty, Dsp, Empty, Clb, Empty, Clb,
Empty, Empty, Empty, Bram, Empty, Clb, Empty, Clb,
Empty, Clb, Empty, Clb, Empty, Empty, Empty, Empty,
Empty, Iob, Empty, Empty, Empty, Clb, Empty, Clb,
Empty, Clb, Empty, Clb, Empty, Empty, Bram, Empty,
Empty, Empty, Empty, Gtx, END };
DeviceInfo xc5vfx70t(23718, 177, 134, xc5vfx70tColumns);
```

OpenPR users need to *make install* again to have the device properly added to the source-code.

ACKNOWLEDGMENTS

The author would like to thank God, to his mother, Elisa, his father, Felix, and his two little girls, Dori and Nubia. The author would also like to say thanks to CPS, Sergio, Carlos A, Carlos F, for their support and specially to William for his patience. And finally last, thanks to his family and friends for their understanding and love.

REFERENCES

[1] *Field-programmable gate array*, 2012 [Online] Avaliable: http://en.wikipedia.org/wiki/Field-programmable_gate_array

[2] *StarBoard Design: Specializing in rapid, effective, analog and mixed-signal electronic design* , 2009 [online] Avaliable: http://www.starboarddesign.com/examples.html

[3] *Adding Large Embedded Block Support to Versatile Place and Route*, 2006 [Online] Avaliable: http://www.eecg.toronto.edu/ aling/ece1718/project/Main/Welcome.shtml

[4] I. Kuon, J. Rose, *Measuring the gap between FPGAs and ASICs, Proceedings of the internation symposium on Field programmable gate arrays-FPGA'06*, 2006

[5] *Virtex-5 FXT FPGA ML507 Evaluation Platform* , 2012 [Online] Avaliable:http://www.xilinx.com/products/boards-and-kits/HW-V5-ML507-UNI-G-image.htm

[6] D. Money, S. Harris, *Digital Design and Computer Architecture*, Mogan Kaufmann Publications, 2007

[7] *Partial re-configuration*, 2012 [Online]. Avaliable: http://en.wikipedia.org/wiki/Partial_re-configuration

[8] S. Guccione, D. Levi, *JBits: A Java-Based Interface to FPGA Hardware*, Xilinx, 1998.

[9] S. Guccione, D. Levi, *JBits: A Java-Based Interface for Reconfigurable Computing*, Xilinx, 1998.

[10] A. Sohanghpurwala, P. Athanas, T. Frangieh and A. Wood, *OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs*, IEEE, 2011.

[11] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, M. French *Torc: Towards an Open-Source Tool Flow*, *Proceedings of the 19th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA 2011 (Monterey, California).

[12] N. Steiner, *A Standalone Wire Database for Routinng and Tracing in Xilinx Virtex, Virtex-E, and Virtex-II FPGAs Virginia Polythecnic Institute and State University*, 2002, (Blacksburg, Virginia).

[13] *Using PlanAhead to manage the Partial Reconfiguration flow*, 2010 [Online]. Avaliable: http://www.xilinx.com/tools/partial-reconfiguration.htm
[14] *JBits SDK*, 2012 [Online]. Avaliable: http://www.xilinx.com/products/jbits/index.htm
[15] *OpenPR FPGA ToolKit*, 2012. [Online]. Avaliable: http://openpr-vt.sourceforge.net/OpenPR/OpenPR.html
[16] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, M. French *Torc: Tools For Open Reconfigurable Computing*, 2011 [Online]. Avaliable: http://torc-isi.sourceforge.net/documentation.php
[17] *NCD file*, 2012 [Online]. Avaliable: http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/floorplanner/html/fp_d_ncd_file.htm
[18] *Virtex-5 FPGA User Guide UG190*, 2010 [online]. Avaliable: http://www.xilinx.com/support/documentation/virtex-5_user_guides.htm
[19] *FPGA Editor*, 2008 [Online]. Avaliable: http://www.xilinx.com/itp/xilinx10/isehelp/ise_n_fed_navpage.htm
[20] *GCC, the GNU Compiler Collection*, 2012. [Online]. Available: http://gcc.gnu.org/
[21] *Boost C++ Libraries*, 2012 [Online]. Avaliable: http://www.boost.org/
[22] *BisonGNU Parser Generator*, 2012 [Online]. Avaliable: http://www.gnu.org/software/bison/
[23] *Flex: The Flast Lexical Analyzer*, 2012 [Online]. Avaliable: http://flex.sourceforge.net/

Dorfell L. Parra Prada is an electronic engineer who graduated in December 2012 from the Universidad Industrial de Santander. He is a working man, dedicated to the digital design research field.