Implementation of a Real-time Signal Processing Application using Partial Dynamic Reconfiguration in FPGA

Maamar Touiza[1], Kamel Messaoudi[1], El-bay Bourennane[1], Abderrezak Guessoum[2]

[1] *Université de Bourgogne, Laboratoire LE2I, BP 47870 Dijon 21078 Cedex France*
[2] *Université de Blida, Laboratoire LATSI, BP270 Route de Soumaa, Blida 9000 Algérie*
maamar.touiza@u-bourgogne.fr, kamel.messaoudi@u-bourgogne.fr, ebourenn@u-bourgogne.fr , guessouma@hotmail.com

Abstract **- Reconfigurable devices, such as modern FPGA provide advanced capabilities which allow the creation of embedded systems on single chips (SoC). One of the most challenging opportunities provided by this kind of devices is the ability to dynamically and partially reconfigure themselves. It consists in the modification of a portion of the circuitry mapped on the FPGA while the system is running. This ability allows development of flexible systems that can deal with changes in the requirements, standards and operational conditions.**
This paper presents partially reconfigurable FIR filter design that employs dynamic partial reconfiguration. Our scope is to implement a low-power, area-efficient autonomously reconfigurable digital signal processing architecture that is tailored for the realization of many response FIR filters using Xilinx FPGA. The implementation of design addresses area efficiency and flexibility allowing dynamically inserting and/or removing the partial modules to implement the partial reconfigurable FIR filters with various types. This FIR filter design method shows the good area efficiency and flexibility by using the dynamic partial reconfiguration method.

Keywords **- Dynamic Partial Reconfiguration, FPGA, Signal Processing.**

I. INTRODUCTION

The Dynamic Partial Self-Reconfiguration concept is the ability to change the configuration of part of an FPGA device by itself while other processes continue in the rest of the device [1].
The general reason for partial reconfiguration is changing the design behavior without full reconfiguration. Dynamic partial reconfiguration has additional advantages when runtime-reconfiguration and efficient resource utilization is desirable.
Runtime reconfiguration is especially useful for applications that require adaptive and flexible hardware since they need to change the behavior of a system to adapt it to externally changing environment. Dynamic partial reconfiguration also facilitates more efficient changing configurations which results in reduced chip area and power consumption and availability of more FPGA resources.
In this work we present a practical application of signal processing implemented in a FPGA and uses the principle of dynamic reconfiguration in its execution. The considered application is a filtering device for audio signal. Two digital filters were made, one is low-pass type and other is high-pass type. These two digital filters present our two reconfigurable modules. All aspects of its design and implementation will be exposed.
This paper is organized as follows. Section 2 gives an overview of dynamic partial reconfiguration. In Section 3 we present the total design flow to implement a reconfigurable architecture. In section 4, we describe our implementation of a reconfigurable digital filter. Some results are given in section 4. Finally, Section 5 concludes the paper and gives some perspectives.

II. PARTIAL RECONFIGURATION OVERVIEW

Partial reconfiguration is useful for systems with multiple functions that can time-share the same FPGA device resources. In such systems, one section of the FPGA operates continuously while other sections of the FPGA are disabled and partially reconfigured to provide new functionality. This technique is analogous to a microprocessor managing context switching between software processes, except in the case of FPGA partial reconfiguration, it is hardware functionality, not software, that is affected (Figure 1).
A partially reconfigurable design consists of a set of full designs and partial modules. The full and partial bitstreams are generated for different configurations of a design. The idea of implementing a self-reconfiguring platform for Xilinx Virtex family was first reported in [1]. The platform enabled an FPGA to dynamically reconfigure itself under the control of an embedded microprocessor. The hardware component of Self Reconfiguring Platform is composed of the internal configuration access port (ICAP), control logic and an embedded processor. .
The embedded processor can be Xilinx Microblaze,

which is a 32-bit RISC soft processor core [6]. The hard-core Power PC on some Virtex can also be used as the embedded processor. The embedded processor provides intelligent control of device reconfiguration run-time.

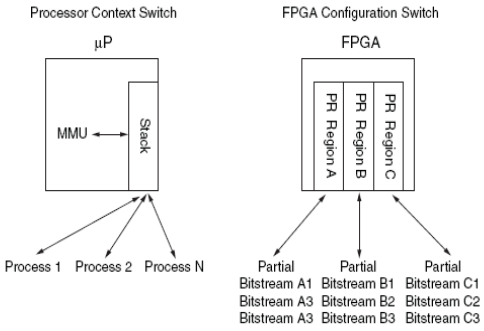


Figure 1 Analogy between Microprocessor Context Switching and FPGA Partial Reconfiguration Regions

The provided hardware architecture established the framework for the implementation of the self-reconfiguring platforms. Internal configuration access port application program interface (ICAP API) and Xilinx partial reconfiguration toolkit design flow provide methods for reading and modifying select FPGA resources and support for relocatable partial bitstreams.

A. Design flow of a dynamically reconfigurable architecture:

For Xilinx FPGA, the design flow that supports partial reconfiguration uses largely the Xilinx ISE tool with a few complements to enable the partial reconfiguration management [5]. These changes are needed because the basic tool can generate just a single total configuration bitstream. In the case of partial reconfiguration, we need to produce multiple configurations bitstreams for the same application (one for the static part and other partial configurations for each reconfigurable module completed).
Several versions of ISE are dedicated to the partial reconfiguration; the latest supported version available is ISE 9.2 SP4 with the additional PR tool. Using Xilinx PlanAhead tool will further simplify the task. It offers a graphical interface to manage the entire process of reconfiguration in the same project and run the various tools for static and partial bitstreams generation.

B. Starting a Partial Reconfiguration design with EDK:

The Xilinx EDK (Embedded Development Kit) is a graphical environment tool for developing a complete application around an embedded processor and integrated it into an FPGA. Based on a design with EDK, we can generate a partially reconfigurable architecture. This involves the creation of reconfigurable devices, changing their HDL descriptions and their implementations using the design flow for partial reconfiguration mentioned previously. Given the highly automated platform EDK, manual intervention is necessary to merge the two design flows. The placement of reconfigurable blocks and the generation of different bitstreams of system can be made via the PlanAhead tool.

C. Bus Macro Communication

The design flow for partial reconfiguration is used to generate a configuration for the static part and partial configurations for each reconfigurable block. To satisfy the communication constraints between the static region and reconfigurable regions, components called Bus Macros must be added between the two parties.

These bus macros ensure firstly that whenever a partial reconfiguration is done, the routing of signals between the two parties remains unchanged. Secondly, they serve to isolate the two parties during the reconfiguration of the partial bitstream to avoid the unstable state of the interconnection signals.

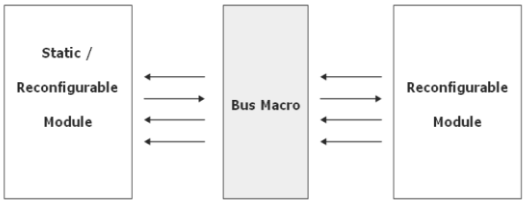


Figure 2 Physical Implementation of a 4-bit Bus Macro by Xilinx

III. DESCRIPTION OF THE EXAMPLE DESIGN

Our example as mentioned is based on partially reconfigurable architecture around a Microblaze processor designed under the EDK environment. The application considered is a filtering device for an audio signal. Two digital filters were made, one is low pass type and the other is high-pass type. These two digital filters are our two reconfigurable modules. The audio signal, acquired from the controller, pass through the reconfigurable filter which can be loaded by choosing one of this two variants. The processed signal will be sent to the sound controller for its reading. The reconfiguration process is controlled by a Microblaze processor through a program written in C. The platform used is

the ML501 Xilinx prototyping board, around a Virtex 5 FPGA.
With the EDK tool we have specified a system containing a Microblaze processor and peripherals strictly necessary (see Figure 3). The Microblaze (MB) is a soft-core processor; we would have added a memory LMB (local memory block) to store code and data. MB processor communicates with devices via a PLB bus (Processor Local Bus). We have attached to this bus some peripherals such : SDRAM memory controller, a UART, Audio CODEC AC'97 controller, ACE controller for access to Card Flash memory, ICAP controller (Internal Configuration Access Port) for lead partial reconfigurations and finally our designed PRR reconfigurable

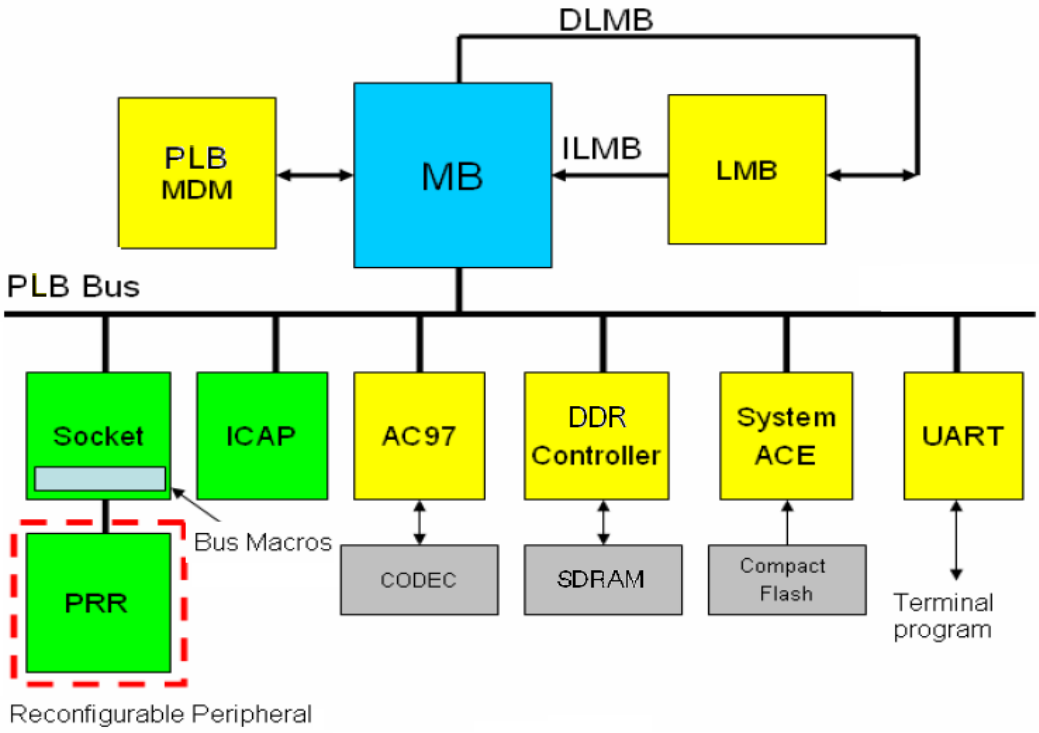


Figure 3 Block diagram of our system

A. *Design of digital filters:*

The realization of our two filters has been made through the Xilinx System Generator tool. This tool is very powerful. It uses the Matlab/Simulink development environment to design an application using specific components provided by Xilinx as a library of graphical components [8]. Once the model is designed, it can be simulated in Simulink for validation and then we can generate with System Generator tool the VHDL template code and a Netlist of the model after synthesis.

Our design model of Numeric Filter (Figure 4) uses the component FIR compiler as main element. This feature allows to dynamically loading all the filter parameters that we wish to achieve. These parameters are determined by the component PDATool (Figure 5) which provided a graphical programming filter.

Using System Generator, we have synthesized the two variants of our filter and can then be inserted into our platform. Their main parameters are described in Table 1.

Filter device attached to the bus through an interface adaptation Socket containing Bus Macros components.

TABLE 1
FILTERS PARAMETERS

	Low Pass	**High Pass**
Filter type	Direct FIR	Direct FIR
Freq pass (Hz)	3000	5000
Freq Stop (Hz)	7000	5000
Filter Order	31	31

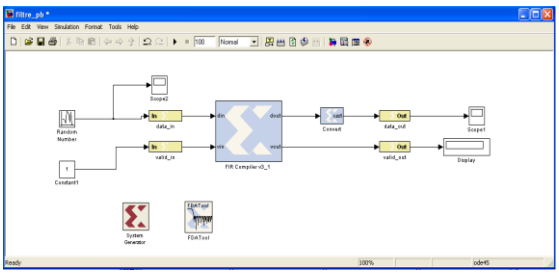


Figure 4: Filter design in Simulink with System Generator

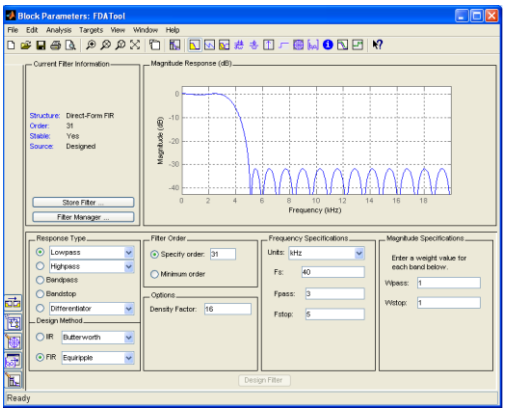


Figure 5 The PDATool component for filter programming

A. *Connecting the two filters to the system bus PLB*

As mentioned, the PLB is a one element of the IBM CoreConnect Bus designed for connection of processor to high-performance peripheral devices under EDK. In order to control our filter modules with processor we have equipped these cores with an interface layer called PLB IPIF that facilitates the connection to the Bus (PLB). This connection has been made automatically by wizard provided in EDK named Create and Import Peripheral. A small intervention in the generated code was necessary to instantiate the filter components in the interface. A PLB core was then constructed for each variant of filter.

To meet the partial reconfiguration design constraints, Bus Macros components must be added between the PLB Bus and the PLB-filter modules. To do this we have designed a second layer called Socket (see Figure 6). This socket allows passing all signals from PLB bus through Bus Macros and controlling them via the control bus DCR.

B. *System Implementation Steps*

Following the design flow for partial reconfiguration in EDK, several steps are needed to build our reconfigurable architecture. These steps are described as follows:

C.1 Creation of our architecture in EDK

The first step is to create a project in EDK witch contains all the modules presented in Figure 3 using the Base System Builder support. The IP core catalog in EDK does not contain all the modules needed for our design such a sound controller. For this we have added to the system this module and one version of our filter module (we have begun with low pass type) associated with socket module (containing Bus Macros components). Once all modules are added and connected to the PLB bus, the address generation system is done by specifying the address range required for each device (referring to the specific documentation of modules). The system is now ready for synthesising to produce all modules netlists. To synthesis the other version of filter module (high pass) we have created a similar project in EDK. In the latter, we have just swap the filters modules.

C.2 Implementation of reconfigurable modules:

After synthesis phase of system in the EDK environment, we pass now to implement our reconfigurable architecture using powerful Xilinx PlanAhead graphical tool. With this tool, we can firstly specify the reconfigurable regions in FPGA. . We associate for each region their reconfigurable modules concerned. . We put Bus Macros components in the appropriate places.
In second phase, we produce the different configurations which will be assembled to generate different bitstreams of system (static and partial bitstreams).

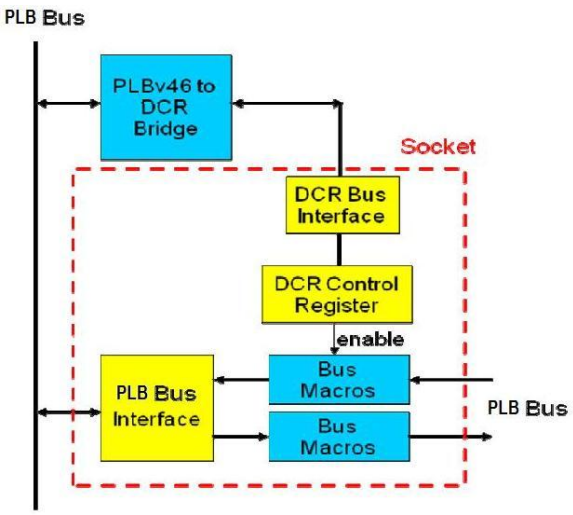


Figure 6: Socket for crossing PLB bus signals through Bus Macros.

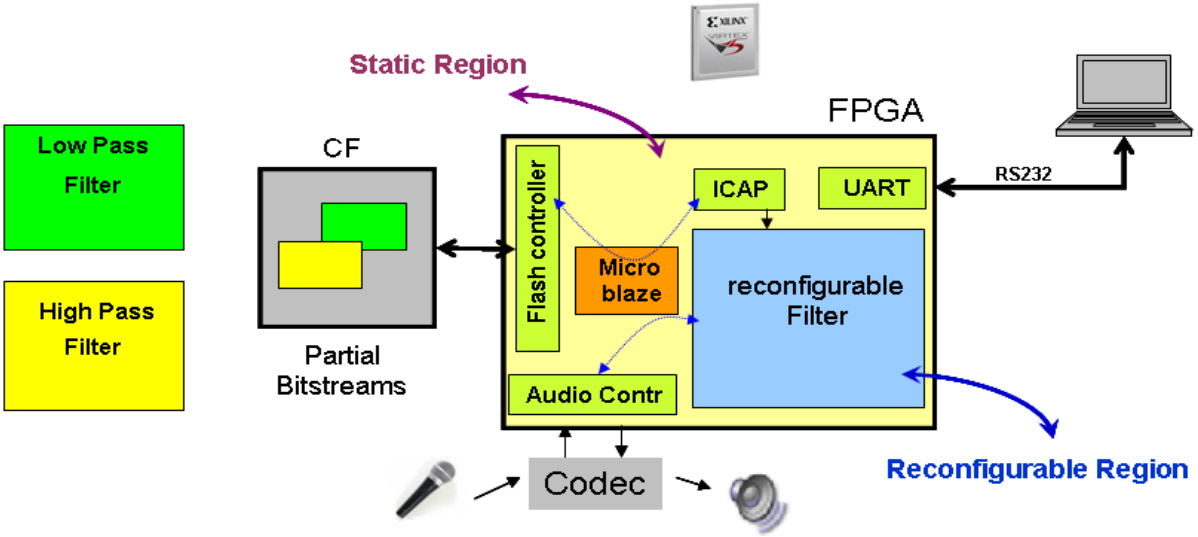
For our architecture, we have a single reconfigurable region that can be loaded by one of two Filter

modules. The dimensions of the region and the number of resources to be included are defined based on resources estimation given by the synthesis tool in EDK for filter modules. The placement of Bus Macros components is done manually within the reconfigurable region. We have 48 Bus Macros for circulate all PLB bus signals between the static and reconfigurable regions. It is necessary that Bus Macros will be placed in orderly manner (in one column for example) to facilitate subsequent routing procedure.

After this phase of construction, DRC (Design Rule Checks) will be launched to ensure that the location of the reconfigurable region and the placement of bus macros are correct. The next step is the generation of configurations of the static part and two reconfigurable modules. This is done via tools ISE_PR mapping (MAP) and placement and routing (Place & Route) launched directly from the PlanAhead tool. The last step in this flow is the execution of PR Assemble command allowing the creation of partial bitstreams and the total bitstream that will be first loaded into the FPGA.

C.3 Driving Program:

As mentioned, C program has been developed to control the



Audio data will be transferred to the filter core (low pass filter at the top). Data processed by filter will be retrieved and sent to a FIFO memory included in the AC97 controller for real-time playback. The partial reconfiguration is controlled via a menu that gives the choice to load one of two filter bitstreams. The selected bitstream is read from the flash card. After examining the file header, the reconfiguration data will be sent to the ICAP device to reprogram the reconfigurable region in the FPGA. The serial port is configured in FPGA as Stdin/Stdout to give us a user interface of a system.

IV. Results :

Table 2 shows the utilization of slices, DSP48E macros of the two reconfigurable Filters and static part of our architecture after mapping operation.

TABLE 2
RESOURCES UTILIZATION FOR PR-FILTERS AND STATIC PART

	PR-Low Pass Filter	PR-High Pass Filter	Static Part
Slices	1268	1257	6106
DSP48E	16	16	3

The partial reconfiguration of filter module can save about 14.7% of slices and 45% of DSP48E macros compared to the full implementation where the static and the two filters types coexist together in FPGA.

V. Conclusion

This paper describes the dynamic partial reconfiguration process in Xilinx FPGA and shows the importance of such method for hardware optimization of resources through a time-sharing of one or more regions of the FPGA by different functionalities which leads to a reduction in hardware cost.

In the future, we are planning to focus on the implementation of self-reconfiguration of FPGA through a real-time operating system to take into account the constraints of real-time applications.

REFERENCES

[1] E Horta, John W. Lockwood, et al., "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," Proceedings of the 39th conference on Design automation, NewOrleans, Louisiana, 343-348, 2002.

[2] G. Mermoud, "A Module-Based Dynamic Partial Reconfiguration Tutorial," Technical report, Logic Systems Laboratory, Ecole Polytechnique Federale de Lausanne, November 2004.

[3]. Xilinx, "Two Flows for Partial Reconfiguration: Module Based or Difference Based," www.xilinx.com, Xilinx Application Note 290, September 2004.

[4] P. Lysaght, B. Blodget, et al., "Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," International Conference on Field Programmable Logic, Madrid, Spain, June 2006.

[5] Xilinx, "Early access partial reconfiguration user guide" Xilinx Application Note, March 2006.

[6] Xilinx, "EDK Design using PlanAhead for Partial Reconfiguration," preliminary copy edition, 2007.

[7] N. Dorairaj, E. Shiflet, and MarkGoosman, "PlanAhead Software as a Platform for Partial Reconfiguration," Xcell Journal Online, (4):68–71, 2005.

[8] Xilinx, "System Generator for DSP User Guide", Xilinx Application Note, 2009.