Exploring the self reconfiguration of FPGA: design flow, architecture and performance

Mohamed Nidhal Krifa
Laboratory of electronics and microelectronics
Faculty of Science of Monastir, Tunisia
kmnidhal@yahoo.fr

Bouraoui Ouni
Laboratory of electronics and microelectronics
Faculty of Science of Monastir, Tunisia
ouni_bouraoui@yahoo.fr

Abdellatif Mtibaa
Laboratory of electronics and microelectronics
Faculty of Science of Monastir, Tunisia
abdellatif.mtibaa@enim.rnu.tn

Abstract - Run-time partial reconfiguration of programmable hardware devices can be applied to enhance many applications in high-end embedded systems, particularly those that employ recent platform FPGAs. Partial Reconfigurable FPGAs allow tasks to be placed and removed dynamically at runtime. These reconfigurable systems have a 2-layer hardware and software architecture that permits a variety of different interfaces. Further, these systems enable self-reconfiguration under software control through a reconfiguration hardware interface called Internal Configuration Access Port (ICAP). In this paper, experiments are conducted in order to evaluate the design complexity and reconfiguration latency of self reconfiguration. The results show that the main goal of self reconfiguration is to shorten the reconfiguration time while not degrading the performance of the final design.

Key words- self reconfiguration, ICAP, JTAG, FPGA, design flow

I. INTRODUCTION

The partial reconfiguration meets the requirements needed by complex design. Indeed, the dynamic partial reconfiguration makes hardware more and more flexible and allows user to modify internal structure of FPGA on the fly, without having to turn off. The partial reconfiguration allows the configuration of parts of FPGA while other parts are still running. Today's, a new FPGA chips with partial reconfiguration capability such as Virtex II, Virtex II Pro, and Virtex-4 families are available. These new devices include two subsets of resources; the system resources and the operational resources [1]. The system resources may be composed by a processor (PowerPC) and by other internal peripherals (SRAM modules and RS232 interface, etc). However the operational resources are zones in the device reserved to the user; it may be used to build reconfigurable modules. During the execution of application, the content of reconfigurable module may be loaded by many functions, which are configured one after one. The most important, for the new FPGA families, is their ability to be self-reconfigured under software control through a reconfiguration hardware interface called Internal Configuration Access Port (ICAP) [2] [3]. The system utilizes the On-Chip PowerPC core and FPGA logic to automatically reconfigure bit streams from an external memory, like compact flash. The main benefit to be gained from using self reconfiguration is the time needed to configure the design. The benefit of reducing reconfiguration time is particularly apparent when making small changes to large designs but reduced benefit can still accrue with larger changes on smaller designs. As a result, if the reconfiguration time takes much time then designer should use self reconfiguration. In fact, result shows that the self reconfiguration may reduce the reconfiguration time up to 90%. However, the self reconfiguration is fairly preferable if significant control over the placement of reconfigurable modules in the reconfigurable fabric of the FPGA chip is desired and/or more flexible area management is required. In generally, the self reconfiguration consumes upper logic resources and device pins than other configuration port (JTAG). In this paper, a self reconfiguration design flow has been proposed. Next, in order to demonstrate the performance of self reconfiguration, a comparison of FPGA partial configuration with two different interfaces: JTAG and ICAP, has been introduced.

II. RELATED WORKS

In the literature there have been some efforts invested in building and loading modules on a partial reconfigurable device. Xilinx initially proposed two types of dynamic reconfiguration [4]. The modular method divides the FPGA in modular regions (static and dynamic) for required functions with the limitation that the modules must occupy the full height of the device and thus the connectivity and topology were limited to 1D. Resources placed inside the modules could not be shared by other modules nor was any routing permitted through the module. Communication between modules was carried out by bus macros placed at the edge of the modules. The second approach, the difference based method is used for minute manual changes to only the design and is not suitable for large applications. In [5][6] the objective of authors consists in placing modules in

optimal places in side the FPGA. The methods proposed by authors were developed under two constraints; impossible placement zone relative to a module and impossible placement zone relative to boards. The impossible placement zone of M relative to M', is the zone where M cannot be placed without overlapping with an already placed module M'. The impossible placement zone relative to board is the zone where M cannot be placed without overlapping with the device boards. In [7] authors presented a design methodology for dynamic relocation of hybrid tasks in side the FPGA. The method starts from a unified task representation, and goes to the final virtual implementation of such hybrid tasks. A framework was also proposed to help user in designing a hybrid task, which also generates automatically the underlying infrastructure that is in charge of performing the dynamic relocation of a hybrid task. A design flow was presented in [8] [9] [10], it based on a scheduler and a placer. The scheduler is used to calculate the optimal time when each module will be submitted to placer. Then, the placer finds the appropriate place on the device where a received module should be placed. In spite of their invested efforts in this field, the authors did not show the practical concept that allows mapping a given module in side the reconfigurable device. Indeed, they just calculated the optimal place of each module inside the device. In [11] the author proposed a methodology allowing partial reconfiguration of the Xilinx spartan II. But, in his work the author did not consider the ICAP to load the FPGA. In [12] Raghavan and Sutton's tool called JPG was developed for Xilinx Virtex devices. The JPG tool is based on the Xilinx Java-based JBits (JBITS) to instantiate a component, generate its corresponding bitstream, and download it to a reconfigurable device such as a Virtex FPGA. In [13] authors describe how integrating microprocessor system into a reconfigurable flow. They propose a slot based architecture with the novelty that they can give a high level specification of the setup and all the communication macros are automatically placed. Each module connects to the On-chip Peripheral Pus (OPB) via the communication macros. The tool flow generates the entire bitstream in EDK and then uses JBITS to cut out the partial bitstreams. JBits is a tool provided by Xilinx that uses Java classes to represent the bitstream and has functions to aid modifying the bitstream at runtime. It requires a Java Virtual Machine to run, which is generally too heavy for an embedded platform, and hence this technique has been restricted to FPGA boards connected to PCs. A good overview of the tools and techniques needed to meet for partial reconfiguration of Xilinx Virtex II, virtex II pro and Virtex-4 was presented in [3]. The author presented a design methodology that uses pre-routed IP cores for communication between static and dynamic modules. Indeed, Authors replaced the hard-wired tristate buffers (TBUFs) with pre-routed IP, bus macros, to implement the communication ports to interface static and dynamic regions of a design. In this approach, to test their methods, authors used iMPACT (the Xilinx tool for downloading bitstreams to program devices) to download partial bitstream via JTAG interface.

III. MODELLING

In this paper, we adopted the model presented in [5][6] for module and reconfigurable architecture, figure 1. The module M has been defined by four parameters M [(X_M, Y_M), W, H]. Where; X_M and Y_M are the coordinates of the origin of M, W (width) is number of columns consumed by M and H (height) is number of lines consumed by M. The FPGA device has been modelled by a matrix Q (Ln Cn), where Ln represents the number of lines and Cn represents the number of columns.

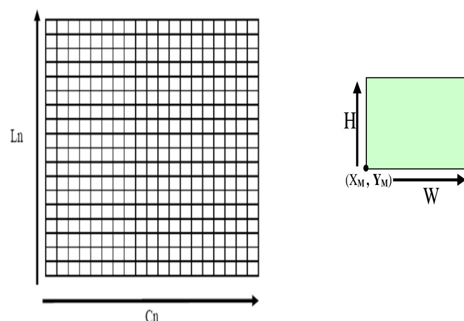

Figure 1: Models of device and module

IV. SELF RECONFIGURABLE DESIGN FLOW

A number of companies offering FPGA design tools provide a self reconfiguration design flow. The synthesis tools available to the designer have improved vastly both in speed and in the quality of results. The following paragraph provides a detailed description of self reconfiguration design flow. As shown in figure 2, the design flow includes four mainly steps:

- System resources phase
- FPGA area management phase

- Operational resources phase
- Assembly phase

A. System resources phase

This step aims to build and to map the netlist of the system resources, such as power PC, UART, OPB bus and BLB bus, ICAP interface, etc in side the FPGA device. The system resource acts as static part of the design. Xilinx Embedded Tool EDK may be used to build the system resource efficiently. The system resource uses a set of compiler tools to handle high-level languages such as C, C++, or assembly language, and produce reliable code for their embedded target. In this context user should develop a software code, executed by the PowerPC, which aims to load automatically the bitstream files of dynamic functions from the compact flash to appropriate modules through the ICAP.

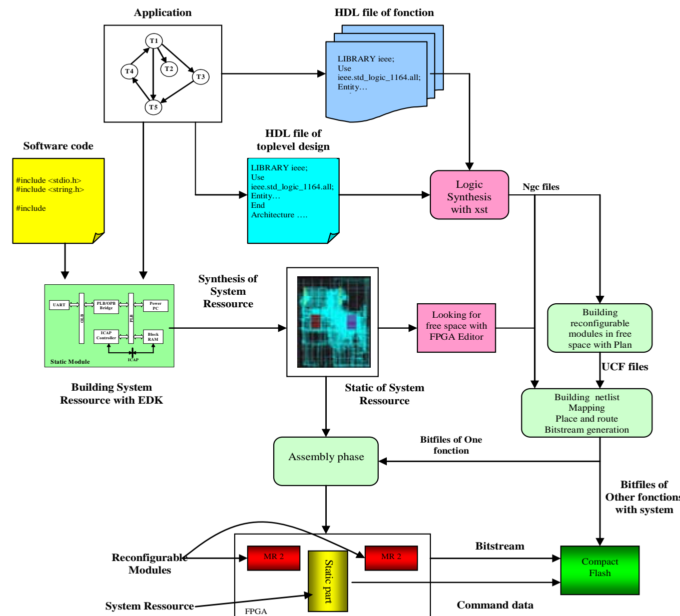

Figure 2: Self reconfiguration design flow

B. Looking for free space

In this step, user looks for free space inside the FPGA after implementing the system resource in order to build reconfigurable modules. The figure 3 shows an example of system resources netlist and the places where reconfigurable modules may be built.

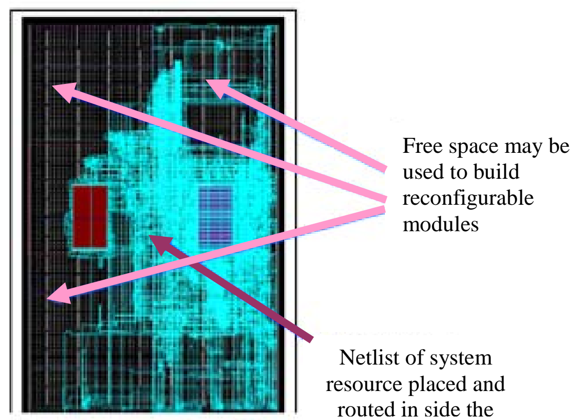

Figure 3: Free space in side the device

C. Operational resources phase

This step aims to build reconfigurable modules inside the free space, and then it aims to map each dynamic functions of the target application to its appropriate reconfigurable module. It composed by the following steps.

1) HDL description:

The reconfigurable modules should be described with HDL as components in side a macro box; the later is called top level design. In addition the communication means, macro bus, signals, between modules each other, between modules and the environment should be described in this step. Further, in this step all functions of the application should be also described.

2) HDL Synthesis:

The synthesis approach allows the generation of optimized architecture from a HDL files described in the first step. User can use the xilinx XST tool to achieve this synthesis. After this step an ngc file is automatically created by xilinx XST tool.

3) Building reconfigurable modules:

The goal of the budgeting phase is to determine the size and location of the reconfigurable module and to lock down the placement of the bus macros. The budgeting phase can be done manually. The process, however, is laborious and instead many of the steps have been automated with a tool called PlanAhead. Indeed, from the ngc file, created by the pervious step, the planAhead tool allows building modules any where in side the device. After this step a user constraints file, UCF file, is automatically generated. The later is very useful for place and route tools. The figure 4 shows two reconfigurable modules Mr1 and Mr2 in the free space.

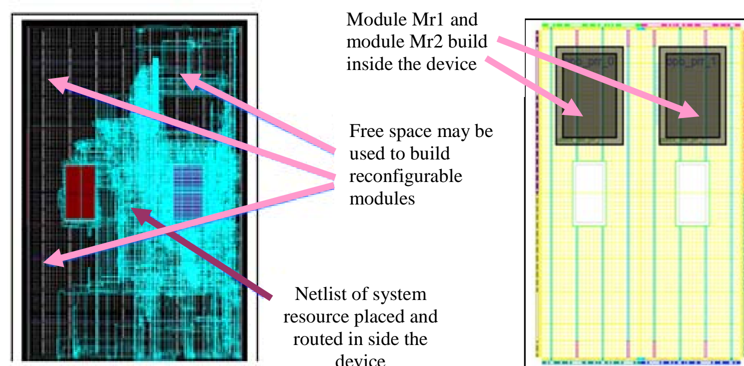

Figure 4: Reconfigurable module inside FPGA device

4) Building communication means:

After buliding modules in side the device, now user should achieve communication means between them. Bus Macros may be used to maintain correct connections between the modules by spanning the boundaries of these rectangular regions. Of course, the location of a macro buses should be closed to the modules location. Further, some modules may communicate with the environment by using the device pins. So, user should use the data sheet of the target architecture to find appropriate pins.

5) Building netlist, mapping, place and route of functions

The ngdbuild tool uses the ngc file and the ucf file, generated from pervious steps, to generate a netlist file for each function and then to map it to each appropriate reconfigurable module. The figure 5 shows a successful mapping as well as a successful place and route of dynamic function in side the reconfigurable modules (Mr1) of figure 4. The red zone in the picture represents the macro bus and the green zone represents the signals.

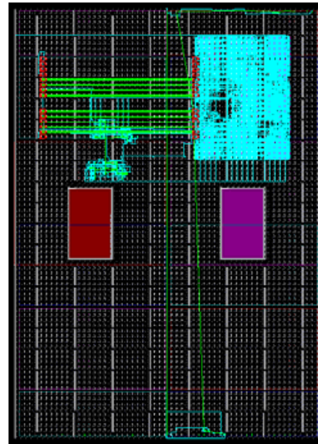

Figure 5: Place and route

6) Bit stream generation

The BitGen tool takes the file of each function completely placed and rooted and produces a series of bits needed to configure the FPGA. The most important, that this file carries the necessary and sufficient information that allows each function to be loaded in its appropriate zone (module) inside the FPGA.

D. Assembly phase

The last phase of the flow is the assembly of the static and reconfigurable parts. The final bitstreams are generated as full bitstreams of the system resource and one reconfigurable module. After loading the full bitstreams it is now possible to reconfigure part of the device with a partial bitstream by the Internal Configuration Access Port (ICAP).

V. EXPERIMENT

Hardware architecture, XUP Virtex-II, on which the design flow is to be mapped, is presented in figure 6. The XUP Virtex-II Pro FPGA development system can be used at any virtually level of the engineering curricula, from introductory courses through advanced research projects.

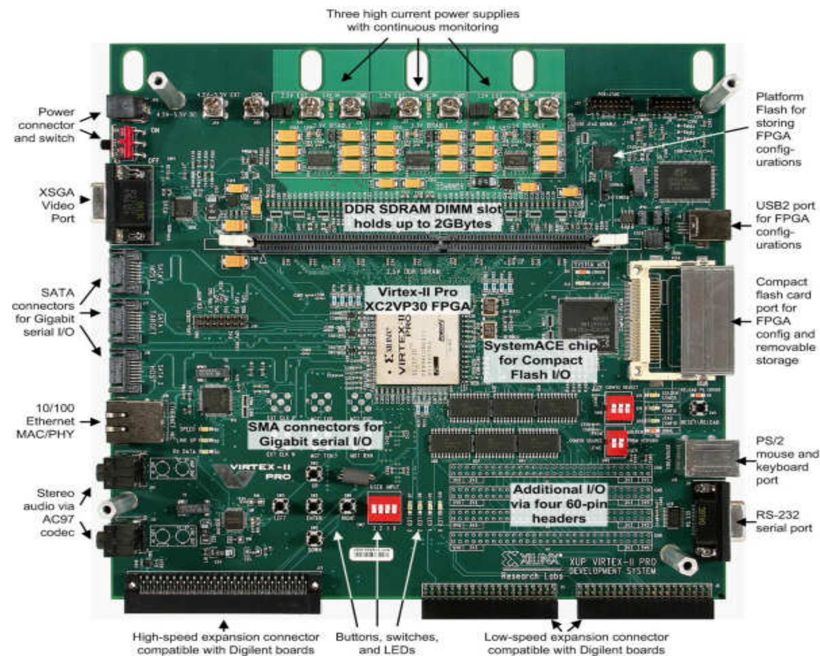

Figure 6: Target architecture

We develop the design flow steps on audio treatment application composed of 3 various filters, band pass, high pass and low pass, shown in figure 7. Let us consider, Fr1 be the low pass filter, Fr2 be the band pass filter and Fr3 be the high pass filter. Let us consider two reconfigurable modules (Mr1 and Mr2) built inside the FPGA device. Let us consider {Fr1, Fr2, Fr3}→ {Mr1 or Mr2}, signify that bitstream files of {Fr1, Fr2, Fr3} may be loaded, one after the one, from the compact flash to the reconfigurable modules Mr1 or Mr2 through the ICAP. The choice of the module reconfigurable {Mr1 or Mr2} as well as the function to be loaded is achieved from the keyboard. We click on "1" to load the low pass filter, "2" to load the band pass filter, "3" to load the high pass filter, "L" to choose Mr1 and "R" to choose Mr2. For example, to load the band pass filter inside the reconfigurable module Mr1 we should click on "2" then he clicks on "L".

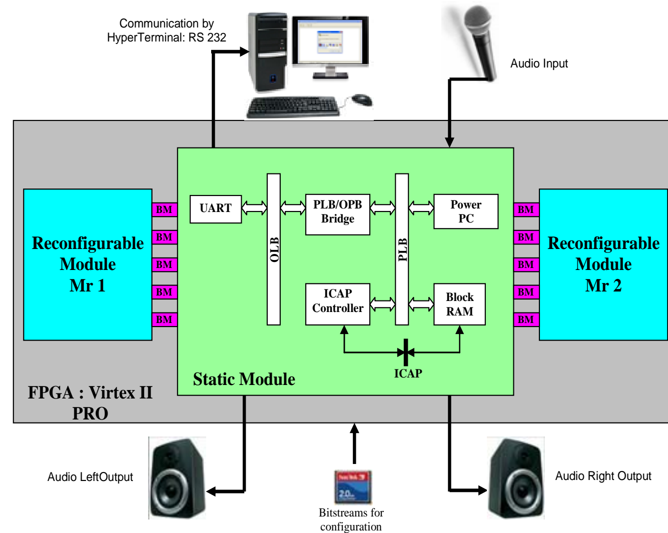

Figure 7: Target application

We started our experience by loading the device by bitstream files of the system resource and the low pass filter. Next, the loading of functions and the choice of module on witch a selected function will be mapped is achieved through the keyboard. For example, when clicking on "2" then on "R" the bitstream file of the band pass filter is automatically loaded, from the compact flash to reconfigurable module Mr2. by the same way; we can load the bitstream file of other function on reconfigurable modules (Mr1 or Mr2). During the reconfiguration of the bitstream of filters, the system resources part is still running. Here, we meet the partial reconfiguration definition; indeed the partial reconfiguration is the process of configuring a portion of a FPGA while the other part is still running. In table 1, experience shows quantitatively comparison the reconfiguration times of two reconfiguration interfaces: ICAP and JTAG. The results show that the ICAP interface is highly suitable when reconfiguration latency needs to be minimized. The whole latency needed to reconfigure our application through the JTAG port is 37, 46 ms however only 0,248 ms are needed to meet the same aim with the ICAP interface. So**,** experiment shows that the reconfiguration speed of the ICAP interface is almost 152 times faster than the speed of the JTAG interface.

This result shows the magnitude of benefit possible using self reconfiguration. Today's large and complex designs are now commonly implemented in FPGAs, however designer suffers principally from the time needed to reconfigure, which is still relatively high, in same case it consumes over than 70% of the whole design latency. A high reconfiguration time may lead to not practical design mainly when designer focuses on the overall latency minimization of the application. This problem may be easily faced when using self configuration; in fact, there is always much gain in reconfiguration latency versus JTAG interface [1].

TABLE 1: DESIGN RESULTS

Partial reconfiguration through ICAP				
	First configuration	Second configuration	Third configuration	
Functions to be loaded	System resources and low pass filter	Only band pass filter	Only high pass filter	
Reconfiguration time (ms)	0.17	0.039	0.039	
Whole Reconfiguration time (ms)				0,248
Partial reconfiguration through JTAG				
	First configuration	Second configuration	Third configuration	
Functions to be loaded	System resources and low pass filter	Only band pass filter	Only high pass filter	
Reconfiguration time (ms)	26	5.73	5.73	
Whole Reconfiguration time (ms)				37,46
Gain in latency (ms)	25,83	5,69	5,69	37,212

The table 2 shows that system resources consume 13 % of the whole FPGA area. Also, this table shows the area occupation of the reconfigurable modules Mr1 and Mr2, each one consumes 12, 5% of whole FPGA area. As a result we used only 38 % of the available area inside the FPGA to build the system resources and the reconfigurable modules. Further, we calculated the used area (%) of each dynamic function. For example the low pass filter consumes 90% of the available area inside reconfigurable module, the band pass filter consumes 86% and the high pass filter consumes 91%. We have good values; in fact the area occupation of function should be closed to the area of reconfigurable module where it will be mapped. To meet this aim, user should calculate the area occupation of each dynamic function, and then he can build the reconfigurable modules correctly.

TABLE 2: DESIGN RESULTS

	System resource	**Reconfigurable Module Mr1**	**Reconfigurable Module Mr2**	**Dynamic function**		
				Low pass filter Fr1	Band pass filter Fr2	High pass filter Fr3
Area (Slice)	1814	1700	1700	1544	1472	1547
Used area %	13	12.5	12.5	90	86	91
Average used area %	38			89		

Generally, the self configuration suffers from the drawback that it fairly considers the geometrical properties of the tasks to be mapped into the FPGA. This leads to a non efficient use of the regularity structure of the FPGAs and resources, which allows fairly flexibility in the placement of the reconfigurable modules. There is always some loss in design performance in term of used resource when using self configuration. In fact, in same case the ICAP design consumes more than 3 times upper logic resources than JTAG interface [1]. Though, with the high capacity, in term resources, of current FPGA, ICAP interface may easily satisfy the resource constraint, as shown in table 2.

VI. CONCLUSION

FPGA designer is faced with many design challenges created from the increased time to market demands and complexities of the design. Therefore, designer should develop new design methodologies of configuration and synthesis, such as self configuration. In fact, self configuration offers the needed flexibility for many complex FPGA designs. In this paper, we present a typical self configuration design flow. This paper shows that self

configuration flow may be used to reduce the overall time that the designer spends to configure the design while not degrading the performance of the final design.

REFERENCES

[1] Heng Tan, Ronald F. DeMara, Abdel Ejnioui Jason, D. Sattler, “Complexity and Performance Evaluation of Two Partial Reconfiguration Interfaces on FPGAs: A Case Study”, Reconfigurable Architectures Workshop (RAW), Greek, 2006.

[2] OPH HWICAP Product specification datasheet - DS 280 (v1.3), March 2004.

[3] Patrick Lysaght, Brandon Blodget, Jeff Manson, Jay Young, Brendan Bridgford, “Invited Paper: Enhanced Architectures, Design Methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs”, Xilinx University Program, 2006.

[4] Xilinx.Inc, “Two flows for partial reconfiguration: Module Based or Difference Based” Xilinx Application Note XAPP290,Septembe 2004.

[5] C. Bobda, “Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications, Springer Publishers (2007).

[6] Ali Ahmadinia, Christophe Bobda, Jurgen Teich, “ Online placement for dynamically reconfigurable devices”, International Journal of Embedded Systems - Vol. 1, No.3/4 pp. 165 - 178. Inderscience Publishers, 2005.

[7] Katarina Paulsson, Michael Hübner, Jürgen Becker, “Dynamic power optimization by exploiting self-reconfiguration in Xilinx Spartan 3-based systems”, Microprocessors and Microsystems, pages 46 - 52, Elsevier Publishers, 2009.

[8] Jan C. van der Veen, Sandor P. Fekete, Mateusz Majer, Ali Ahmadinia, Christophe Bobda, Frank Hannig, and Jürgen Teich, “Defragmenting the Module Layout of a Partially Reconfigurable Device”, Hardware Architecture (cs.AR); Data Structures and Algorithms (cs.DS), May 2005.

[9] S.P.Fekete, J.C.van der Veen, A.Ahmadinia, D.Gohringer, M.Majer, J.Teich, “Offline and Online Aspects of Defragmenting the Module Layout of a Partially Reconfigurable Device”, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 16, Page(s):1210 - 1219, Sept. 2008.

[10] Ali Ahmadinia, Jürgen Teich, “Speeding up Online Placement for Xilinx FPGAs by Reducing Configuration Overhead”, proceeding of 12th VLSI SOC, December 2003

[11] Grégory Mermoud, “A Module-Based Dynamic Partial Reconfiguration tutorial”, available in http://ic2.epfl.ch/~gmermoud/DPRtutorial.zip

[12] A.K. Raghavan, and P. Sutton, “JPG - A Partial Bitstream Generation Tool to Support Partial Reconfiguration in Virtex FPGAs”, in Proceedings of International Parallel and Distributed Processing Symposium, (IPDPS’02), Fort Lauderdale, Florida, USA, April 15-19, 2002.

[13] M. Hübner, J. Becker, "Exploiting dynamic and partial reconfiguration for FPGAs: toolflow, architecture and system integration", Proceedings of the 19th annual symposium on Integrated circuits and systems design, tutorial session, 2006.