A Design Methodology for Dynamic Reconfiguration: The Caronte Architecture.

Fabrizio Ferrandi
Politecnico di Milano
Milano, Italy
ferrandi@elet.polimi.it

Marco D. Santambrogio
Politecnico di Milano
Milano, Italy
santambr@elet.polimi.it

Donatella Sciuto
Politecnico di Milano
Milano, Italy
sciuto@elet.polimi.it

ABSTRACT

The most common reconfigurable devices today are *Field Programmable Gate Arrays*, FPGAs. Aim of this paper is to propose a design methodology for *dynamically reconfigurable systems* providing a dynamic architecture that, thanks to the processor embedded in the FPGA, is able to dynamically change the design implementation to meet and satisfy all the requirements of the system implementation. The proposed methodology provides a solution for the partial dynamic reconfiguration of an embedded system, using a common FPGA and development board, without any specific or dedicated device. This paper describes the *Caronte* architecture used to implement the proposed approach, showing how it is possible to obtain a reconfigurable system just using tools that are already widely used to design FPGA-based systems.

1. INTRODUCTION

Nowadays many emerging applications in communication, computing and consumer electronics demand that their functionality stays flexible after the system has been manufactured. Such a flexibility is required in order to cope with changing user requirements, improvements in system features, changing protocol and data-coding standards, demands for support of a variety of different user applications, etc. Most applications runnig on FPGA-based systems are implemented using a single configuration per FPGA as [4]. This means that the functionality of the circuit does not change while the application is running. Such an application can be referred to as being Compile-Time Reconfigurable, CTR, because the entire configuration is determined at the compile-time and does not change throughout system operation. Another implementation strategy is to implement an application with multiple configurations per FPGA [7], [5], [2]. In this scenario the application is divided into time-exclusive operations that need not, or cannot, operate concurrently. Each operation is implemented as a distinct configuration which can be donwloaded into the FPGA as necessary at run-time during application operation. This approach is referred to as Run-Time Reconfiguration, RTR or Dynamic Reconfiguration. Dynamic Reconfiguration can be achieved into two different ways: dynamic external reconfiguration and embedded reconfiguration. Dynamic external reconfiguration implies that an active array may be partially reconfigured by an external device such as a Personal Computer, while ensuring the correct operation of those active circuits that are not being changed. Embedded reconfiguration extends the concept of dynamic reconfigurability assuming that specific circuits on the array are used to control the reconfiguration of other parts of the FPGA. Clearly the integrity of the control circuits must be guaranteed during reconfiguration, so by definition embedded reconfiguration is a specialized form of dynamic reconfiguration [9]. A new class of cores called run-time parameterizable (RTP) has been introduced in [3]. RTP cores allow a single core to be computed and customized at run-time. For example, an adder core can be produced, and then parameterized at run-time for different operand widths. An innovation of this approach consists in considering the RTP cores as a specific example of a reconfigurable core, placed on the programmable device in a dynamic manner to respond to the changing computational demands of the application. The problem of this methodology is that the RTP are targeted only to a single device family and there is no information about the communication channel between RTP and about how they solve the physical reconfiguration problem.

In [1] the hardware subsystem of the reconfiguration control infrastructure sits on the on-chip peripheral bus, OPB. The microprocessor, PowerPC or MicroBlaze, communicates with this peripheral over the OPB bus. The hardware peripheral is designed to provide a lightweight solution to reconfiguration. In order to do this it employs a read / modify / write strategy. The program installed on the processor requests a specific frame, then the control logic of the peripheral uses the ICAP to do a readback and loads the configuration data into a dual-port block RAM. When the read-back is complete, the program directly modifies the configuration data stored in the BRAM. Finally the system writes the modified configuration data back to the device.

In this scenario the proposed methodology focuses its attention on just a reconfigurable device, a single FPGA, trying to figure out how this device can be used to implement an embedded dynamic reconfigurable system just without any additional cost due to a use of a dedicated device to program it. The idea is to use EDK, Embedded Development Kit, produced by the Xilinx Inc. as the starting point of the entire methodology. One of the most important feature implemented in EDK is the opportunity to develop both the software and the hardware part of the design in just one tool focusing the attention on the entire implementation of the desired system. To implement this characteristic, EDK is designed to provide designers with a rich set of design tools, such as XPS, Xilinx Platform Studio, gcc, XST, Xilinx synthesizer, and a wide selection of standard peripherals required to build embedded processor systems using the MicroBlaze processor or/and the IBM PowerPC CPU, [10].

2. THE PROPOSED METHODOLOGY

This section shows how to implement a dynamic reconfigurable system onto a common FPGA, using a development tool such as EDK, Embedded Development Kit, produced by Xilinx Inc., just combining different design flows into a new design methodology. Aims of the proposed methodology is to introduce dynamism also in the hardware part of the system without loosing or increasing the overall implementation time, and without changing the devel-

opment tool and implementing the dynamic system onto a common FPGA, [8]. The proposed methodology is depicted in Figure 1 and in particular it shows how it has been inserted the Caronte flow into the standard FPGA flow. As shown in Figure 1, the Caronte Flow

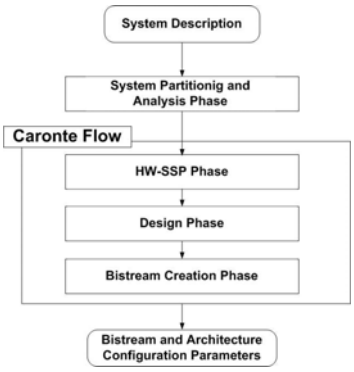

Figure 1: Reconfiguration Design Methodology Flow.

accept as input the result of a previous partitioning and analysis phase, [8]. Both the FPGA and the initial description of the system have to be partitioned into several parts to provide the correct starting point to find out a dynamic reconfigurable design for the desire system description. This first phase identifies all the processing element of the description that will be mapped onto the corresponding part of the FPGA. These elements, in order to be downloaded onto an FPGA, have be transformed into a set of reconfiguration bitstream by the Caronte Flow. To achieve this goal the Caronte Flow is mainly composed by three phases:

HW-SSP Phase The HardWare Static System Photo Phase identifies a set of EDK system descriptions that are going to be reconfigured;

Design Phase This phase aims at creating all the information needed to compute all the bitstream to physically implement the embedded reconfiguration of the FPGA. This phase solves three different problems:

- Identify the structure of each reconfigurable block providing a specific implementation for each of them. This phase is based on the Xilinx Modula Based Design approach;
- Identify, using the *Floorplanner* tool provided in ISE tool chain, the area of each reconfigurable component of the system;
- Solve the communication problem between reconfigurable modules, by introducing *Bus Macros* that allow signals to cross over a partial reconfiguration boundary.

Bitstream Creation Phases This phase creates all the bitstreams needed to implement the system description onto an FPGA through the dynamic embedded reconfiguration.

2.1 HW-SSP Phase

The input of the Caronte Flow is composed by a special set of EDK Cores, the BlackBox elements. Those are used by the HW–SSP phase to create all the HW Static System Photos. An HW–SSP is an EDK system based on the Caronte architecture. This architecture contains a fixed part and several reconfigurable blocks, named BlackBoxes. The application moves from an HW–SSP to another by reconfiguring the BlackBoxes and by leaving the fixed part unchanged. The idea is to consider the system in time as a sequence of static photos whre all the HW–SSPs *share* the static part of the system.
Finally, the EDK output is used as input for the next phase.

2.2 Design Phase

The idea is to implement a specific environment oriented to the reconfiguration that, starting from a system description provided by the use of EDK, uses the *Modular Based Design*, MBD, approach to define all the bitstream for the final implementation of the system. According to this scenario the proposed approach provide to the designer a methodology that strongly decrease the time to market of the final implementation of the system. In order to obtain all the HW-SSPs needed by the MBD the designer, or the automatic version of the Caronte flow, has no reason to use the complete EDK implementation chain, but just a little part of it. The produced VHDL descriptions will be pseudo automatically changed according to the fact that the system has a dynamic nature. The main changes to the system descriptions deal with the communication channel between modules. In order to allow modules to communicate with each others in a dynamic system, a special bus, the *BUS Macro*, has to be introduced into the design description. Each time partial reconfiguration is performed, the bus macro is used to establish unchanging routing channels between modules, guaranteeing correct connections. Once that all the description have been adapted to the embedded dynamic reconfiguration nature of the final system, the problem that still remain to be solved is that the automatic synthesis provided by EDK does not care about placing all the parts of the same component in the same area. In order to cope this *problem* the Floorplanner, a tool contained in the ISE Xilinx package, could be use. The Floorplanner provides a simple way to constrain the placement of every component of a project onto a specific area of the physical architecture. The reason why area constraints are so important for the partial reconfiguration is that partial reconfiguration is nothing more than a configuration of the FPGA with a bitstream which contains configuration data only for a specific part of the FPGA. These bitstreams are called partial bitstreams according to the fact that they are computed as the logical difference between two complete configuration bitstreams for the target FPGA. This means that without constraining the components placement, it is impossible to guarantee that the partial bitstream between two configurations will affects only the configuration of the desired area.

3. THE CARONTE ARCHITECTURE

This section aims at showing the architecture adopted by the proposed methodology for the embedded partial dynamic reconfiguration: Caronte. This architecture is entirely implemented in the FPGA device. It is constituted by several elements such as a processor, memories, a set of reconfigurable devices and a reconfiguring device. Figure 2 presents the EDK Caronte architecture view that has been implemented on a VirtexII-Pro FPGA. The core of this architecture is the PPC405 processor which implements both the controller and the scheduler of the given system implementation. A part from the processor there are other components involved in the reconfiguration action. As shown in Figure 2 it is possible to list at least four classes of other components:

- **ICAP**, used to read/write a configuration from/to the BRAM to/from a specific BlackBox;
- **Memory**, used to store all the partial bitstream data information;

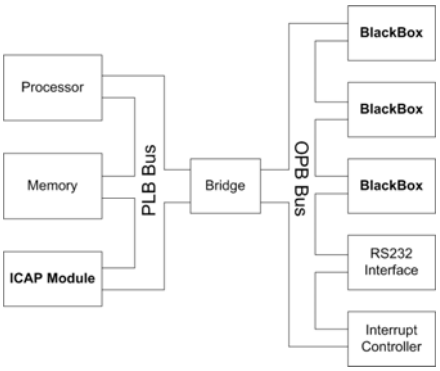

Figure 2: Caronte Architecture Overview

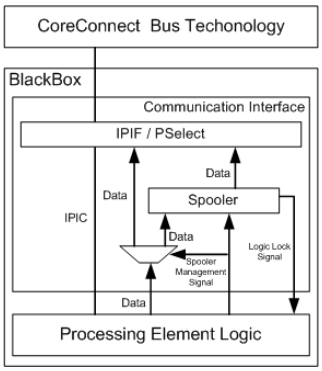

Figure 3: A BlackBox overview

- **BlackBoxes**, the reconfigurable components;
- **Interrupt Controller**, used by the PPC405 processor and the BlackBoxes to dialog one to each other.

3.1 Processing Element Mapping: BlackBox Definition

A BlackBox is a fixed known portion of FPGA that can be reconfigured completely without interfering with the execution of the remaining part of the FPGA. Therefore a BlackBox can be considered as a shell for processing elements. This shell provides also the communication channel interface between the node and the system. This interface allows the node to send data directly on the communication channel or to temporally store a fixed number of data in its communication spooler, which is a sort of data repository used during the reconfiguration action. A BlackBox can be seen as an EDK component although this is a simplified way of thinking of a BlackBox. A BlackBox is not a static component mapped onto the FPGA, as any classical EDK component. A BlackBox can be considered as a virtual shell used to contain different processing elements of the system description that need to be mapped onto the FPGA. In order to be able to implement a partial reconfiguration of a portion of the FPGA it is important to know which is the portion that has to be reconfigured. The Xilinx Platform Studio Tool of EDK, used to create FPGAs architectures, offers an automatic synthesis engine that generates a real project implementation by arranging each logic unit in a standard way. A BlackBox provides the interfaces needed by the VHDL description of a processing elements to dialog with all the other components of the architecture, such as the CoreConnect bus, the processor, the interrupt controller and the other blackboxes. A BlackBox is shown in Figure 3. What is going to really change during reconfiguration is the Processing Element node logic, in fact the communication interface and IP Interconnect (IPIC) between the node logic and the interface remain always the same. This means that a BlackBox is constituted by two VHDL, Verilog or EDIF files, the first one containing the *architecture-dependent* logic interface and the second one the processing element description.

3.2 The Caronte Software Implementation

In a previous implementation, the Caronte software has been implemented as a *standalone* system while now it has been based on a Linux operating system. Caronte implements an architecture where, at the right time, each processing element can be mapped, thanks to the controller, according to the placement information
The time of the reconfiguration has been computed statically, but it can be modified due to the actual execution or external environment inputs. This is the reason why the PPC405 runs also a dynamic scheduler, which takes into consideration the runtime implementation of the system description. The scheduler, according to the information provided by the controller, updates the processing element time information and computes a new schedule on the graph by following a List-based approach, in order to identify the new critical path and reorder accordingly the processing elements.
Caronte sees each processing element as a component that has to be mapped onto a specific FPGA area. According to this scenario the core of the Caronte embedded reconfigurable architecture can be considered the PPC405 processor which implements both the controller and the scheduler of the given system implementation.
The controller stands in a time watching state, controlling that the running time of each BlackBox meets the time statically computed until a BlackBox ends its execution or the running time of a generic y BlackBox exceeds its statically predefined deadline. In the second case the controller informs the scheduler that the run time of the y BlackBox is greater than the estimated one by activating the scheduler to compute the new critical and a new feasible solution. After informing the scheduler the controller returns in its time watching state, waiting for a new event. In the first case, the correct end of a BlackBox execution, implies a reconfigurable action to be performed. At the end of its execution the BlackBox informs the controller of this event. At this point the reconfiguration action begins. The controller knows which is the next node that has to be mapped on this BlackBox and downloads from the memory to the BRAM the correct configuration bistream. At the same time the controller, the PPC, informs all the blackboxes, BBs, which can be disturbed by the reconfiguration action to activate their spooler communication system. At this point the PPC405 allows the ICAP to reconfigure the BlackBox with the new configuration bitstream. The ICAP module is the main module that performs the in-circuit reconfiguration, and it is available only in the Virtex-II, or greater, series devices. The ICAP block is located in the lower right hand corner of the FPGA, this information is very important in the modular based design approach, in fact we implement a *fix shell* for the ICAP connected to the rest of the system through the presences of the BUS Macro. It is used to access the device configuration registers as well as to transfer configuration data using the SelectMAP protocol. Finally, when the new BlackBox has been mapped and it starts its computation, the ICAP informs the processor that the reconfiguration action ended with success. After that the controller enables all the communications interrupted by the reconfiguration.

4. TEST AND RESULTS

The Caronte flow has been applied to the MD5 algorithm to test the Caronte architectural features, like the possibility to store the reconfiguration data on the board without external resources. The MD5 algorithm takes as input a message of arbitrary lenght and produces as output a 128-bit *fingerprint* or *message digest* of the input. It is conjectured that it is computationally infeasible to produce two messages having the same digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be *compressed* in a secure manner before being encrypted with a private key under a public-key cryptosystem such as RSA.
In order to correctly test the Caronte architecture and the embedded reconfiguration, we apply the first phase of analysis and partitioning on the proposed algorithm and we obtain a first HW/SW codesign solution of the entire system. After that previous step we iterate the partitioning phase onto the hardware description of the system to obtain all the processing element needed by the Caronte flow as input. The hardware implementation of the system has been divided into 5 processing elements, modeled as BlackBoxes, and therefore it ispossible to define all the HW-SSPs for the MD5 algorithm (see Table 2). According to this scenario the Caronte architecture choosen for the MD5 application is composed by two BlackBoxes, BB_1 and BB_2, and by the Caronte Core, composed by the processor, the memory, the ICAP module and all the others static part previously described. The access time to the memory,

Table 1: HW-SSP Description

HW-SSP	Fix Module	BB_1	BB_2
0	Emply	Emply	Emply
1	Caronte Core	PE-A	PE-B
2	Caronte Core	PE-C	PE-B
3	Caronte Core	PE-C	PE-D
4	Caronte Core	PE-E	PE-D
5	Caronte Core	PE-E	PE-F
6	Caronte Core	Empty	PE-F

where all the difference bitstream are stored, has been obtained via a timing test: writing 32 bits of data takes 0.135μs, while reading the same amount of data requires 0.020μs. Without considering the first configuration bitstream, which implies a complete configuration of the FPGA, the comparison between the external reconfiguration and the embedded one are shown in Table 2. Although the

Table 2: Embedded Vs External Reconfig.

Action	External Rec.	Embedded Rec.
Rec. Time C block	14.558s	2.159ms
Rec. Time D block	14.597s	2.300ms
Rec. Time E block	14.560s	2.229ms
Rec. Time F block	15.482s	2.932ms

embedded reconfiguration is faster than the external one, it still remains too slow if compared to the block execution times, 431.2μs for each block, from now on considered as BlackBox. Due to this observation we decide to change our execution model to be able to justify the reconfiguration approach using a model similar to the one proposed in [6]. The idea is to iterate the execution of each block on a sufficient amount of data to enlarge its computation time until the execution time is bigger than the one needed to reconfigure the BlackBox. The number of iteration required by a generic processing element that makes its computation on a set of data of a fix-dimension each timeas beeb computed as greater than 8

5. CONCLUSIONS

Preliminary results show that the Caronte methodology, implementing a module-based oriented system approach based on an EDK system description, provides a low cost approach to the dynamic reconfiguration problem. Some improvements can be done introducing an automated version of the area-constraints procedure and an automated binding between the bus interface and the core component VHDL files to define the corresponding BlackBox. It would be also interesting to extend the framework to implement a dynamic reconfigurable soft processor.

6. REFERENCES

[1] B. Blodget, S. McMillan, and P. Lysaght. A lightweight approach for embedded reconfiguration of fpgas. 1991.

[2] P. French and R.W.Taylor. A self–reconfiguring processor. pages 50–59. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machine, D.A. Buell and K.L. Pocek, 1993.

[3] S. Guccione and D.Levi. Run–time parameterizable cores. pages 215–222. IEEE Symposium on Filed Programmable Logic and Application, 1999.

[4] D. T. Hoang. Searching genetic databases on splash2. pages 185–191. Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, D.A. Buell and K.L. Pocek, 1993.

[5] P. Lysaught, J. Stockwood, J. Law, and D. Girma. *Artificial neural network implementation on a fine–grainde FPGA*. R. Hartenstein and M.Z. Servit, 1994.

[6] R. Maestra, F. Kurdahi, M. Fernandez, R. Hermida, N. Bagherzadeh, and H. Singh. A framework for reconfigurable computing: Task scheduling and context management. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 9(6):858–873, December 2001.

[7] D. Ross, O. Vellacott, and M. Turner. An fpga–based hardware accelerator for image processing. pages 299–306. More FPGAs: Proceedings of the 1993 International workshop on field–programmable logic and applications, W. Moore and W. Luk, 1993.

[8] M. D. Santambrogio. *A Methodology for Dynamic Reconfigurability in Embedded System Design*. Graduate Thesis Politecnico di Milano, 2004.

[9] S. Tapp. Configuration quick start guidelines. *XAPP151*, July 2003.

[10] Xilinx Inc. *Embedded Development Kit EDK 6.2i*. Xilinx Inc., 2004.