VHDL to FPGA automatic IP-Core generation: A case study on Xilinx design flow

Fabrizio Ferrandi
DEI - Politecnico di Milano
e-mail: ferrandi@elet.polimi.it

Giovanna Ferrara
Siemens S.p.A.
e-mail: giovanna.ferrara@siemens.com

Roberto Palazzo
DEI - Politecnico di Milano
e-mail: roberto.palazzo@email.it

Vincenzo Rana
DEI - Politecnico di Milano
e-mail: vincenzo.rana@email.it

Marco D. Santambrogio
DEI - Politecnico di Milano
e-mail: marco.santambrogio@polimi.it

ABSTRACT

This paper aims at introducing a methodology that allows an easy implementation of IP-Cores focusing only on their functionalities rather than their interfaces and their integration in a given architecture. The proposed approach implements all the communication infrastructure needed by a component, described in VHDL, to be finally inserted into a real architecture that can be implemented on FPGAs, reducing the time to market of the final implementation of the system. To validate the entire methodology, we have performed a comparison based on the CoreConnect communication infrastructure, between our results with the classical Xilinx design flow using EDK and ISE.

1. INTRODUCTION

The design of embedded systems has rapidly changed during the last decade. It is possible to identify two main responsible factors: hardware/software codesign and dynamic reconfigurable architecture.The typical target architecture to which this kind of methodologies applies is one in which a general-purpose microprocessor is used in conjunction with a so called Intellectual Property Core (IP core), typically implemented as a custom hardware component. The products of the co-design process are thus the hardware description of the IP core and the code for the software portion that will run on the CPU. Together, the two components implement the overall system. Recently, reconfigurable devices such as Field Programmable Gate Arrays (FPGAs), which allow fast and relatively low-cost prototyping, are also being used in target architectures as IP cores. Moreover, these devices introduce yet another degree of freedom in the design workflow: the designer can have the system autonomously modify the functionality performed by the IP-Core according to the application's changing needs while it runs. Research in this field is, indeed, being driven towards a more thorough exploitation of the reconfiguration capabilities of such devices, so as to take advantage of them not only at compile-time, i.e. at the time when the system is first deployed, but also at run-time, which allows the reconfigurable device to be reprogrammed without the rest of the system having to stop running. In both the co-design scenario and the dynamic reconfiguration case, the IP-Core functionality identification, its generation and the possibility of having a flexible and easy mechanism of plug-in of the IP-Core into a fixed and known system are issues of primary concern.

In order to propose a methodology that can be really used into the common design flow of an embedded system we decide to focus our attention on the system design using the Xilinx tool chain composed by EDK and ISE. The final implementation of the system has been tested using the VirtexIIPro Xilix FPGAs such as the 2vp7 and the 2vp20.

The paper presents a first part that explain the IP-Core integration into an EDK system, with special attention on the standard bus interfaces already available in the EDK library. After the EDK description, in section 3 will present the proposed methodology. In section 4, a real case study based on the proposed methodology and the Caronte architecture has been presented and finally, in section 5, we will propose a set of tests to validate the entire methodology.

2. THE EDK CASE

The Embedded Development Kit, EDK, produced by Xilinx, is designed to provide designers with a rich set of design tools and a wide selection of standard peripherals required to build embedded processor systems using MicroBlaze, the Xilinx soft processor solution, and the new and unique feature in Virtex-II Pro, the IBM PowerPC CPU [1].

EDK can be used to easily implement a complete system description that has to be implemented using the Virtex II and the Virtex II Pro SoC platform. IBM PowerPC and MicroBlaze soft-core processor are both implemented with the IBM CoreConnect bus architecture, [2].

The IBM CoreConnect bus architecture eases the integration and reuse of processor, system, and peripheral cores within standard product and custom system-on-a-chip (SOC) designs, [3]. The versatility of CoreConnect bus architecture allows engineers to assemble custom SOC designs using cores designed to CoreConnect specifications. With time-consuming performance, functional, and timing pattern issues resolved, designers can focus on product differentiation - dramatically reducing costs and time-to-market for simple and complex SOC designs, [3, 4]. The CoreConnect bus architecture is a standard SOC design point, [3], and consists of three buses, [2, 4]:

- PLB: The Processor Local Bus is a 64-bit width primary high-performance memory bus that interfaces directly with the processor, both IBM PowerPC and MicroBlaze.
- OPB: The On-chip Peripheral Bus is designed for less time-critical connections to peripherals, it is a 32-bit

data-width bus used for interfacing slow peripheral, or slow peripheral interfaces to the system.

- DCR: The Device Control Register bus is used mainly to save OPB and PLB bandwith.

Any custom IP-Core that connects to the CoreConnect bus must meet the principles of the PLB or the OPB protocol and the design must meet the requirements of Platform Generator and CoreGen flow to take advantage of the simple automated flow that generates the system-level architecture as well as other template scripts supported by Xilinx.

The Xilinx Platform Studio, *XPS*, the EDK tool chain used to implement hw and sw platforms, can be used to design embedded system combining together several IP-Cores that communicate with each other through the CoreConnect bus architecture and the final system implementation that can be mapped onto the Xilinx FPGA. In order to develop EDK-based system designs able to meet different requirements according with the designer needs, XPS provides the possibility of extending its library components adding custom IP-Cores designed by users. Although this operation is very powerful and flexible it still requires a lot of time to be completed and it can be done only using the EDK integrated design environment, that means a huge amount of computational resources.

3. THE PROPOSED METHODOLOGY

The proposed approach allows the designer to create his own architecture and system description using EDK; therefore we decided to focus our attention on this solution just to be able to propose a first real prototyping implementation of the flow that can be used in real application, as shown in section 4 and 5. Once the architecture has been defined, it provides a tool chain that, accepts as input the system architecture and the new functionalities described in VHDL and produces the final configuration bitstream file describing the architecture with the new functionalities plugged in it and ready to be used on the FPGA.

3.1 IP-Core generator

The creation of a component is a process that can be divided into two distinct parts: the implementation of the IP-Core logic and the interconnection with the external world. These two parts allow the component to carry out its functionality, but only the first one is responsible of data elaboration, while the second one interacts the architecture. This second part can be further separated in two other parts: the communication with the bus and the registers mapping.

The IP-Core generator tool has been developed to solve the problem of the automatic creation of an EDK-compatible core starting from a generic VHDL functionality.

The IP-Core generator tool can be described as constituted by three main steps:

- The input phase: the tool needs the name and the VHDL description of the core;

- The **Reader** process computes the first part of the IP-Core generator tool:
  - Parse the VHDL core description;
  - Builds the signals list: the basic idea is that when a signal is recognized, it is analyzed and its information is stored in the signals list, this action is repeated until the end of the core is reached;
  - Save the core name in a variable used by the following process.

- The second process, the **Writer**, is characterized by the following actions:
  - Accept as inputs the signals list, generated by the Reader, and the core name, provided by the user;
  - Create a stub file between the Core and the IP-Core description VHDL files;
  - Design the top architecture which is the final IP-Core: according to the bus architecture chosen, OPB or PLB, the correct communication infrastructure is created to complete the generated stub file to realize the final IP-Core design.

During this flow, if one of the two processes fails, the flow is halted and an error message containing useful information about the problem, is provided to the user. If the flow ends with no error the system reaches the end of the IP-Core generation tool and it is ready for the next phase, which is described in the following section.

3.2 EDK System creator

Aim of this phase is to provide a tool able to automate the binding between a generic IP-Core, in the proposed flow it is provided by the IP-Core generator tool, and a given system EDK-compatible architecture. Both these components are considered as the input of the EDK system creator tool with also a configuration file containing the physical addresses for the IP-Core. This tool has been tested using a IBM PowerPC-based architecture, but there is no conceptual problem of using also a MicroBlaze or a multiprocessor one. Basically the EDK system creator tool is composed by two phases:

- Import of the IP-Core: set all the necessary parameters to allow the IP-Core to be treated as an EDK component;

- Insertion of the IP-Core: insert the IP-Core in the EDK architecture provided as input.

3.2.1 Import of the IP-Core

The *import phase* receives as input the address space of the IP-Core, specified in the configuration file, and the VHDL files and creates the full IP-Core directory tree of the IP-Core, the PAO and MPD description files, needed by EDK, according with the EDK requirements. The input files and the IP-Core name are taken from a repository, specified by default, or modified by the user changing the configuration file, as "./repository", while the IP-Core addresses, used to define its working space, are provided in the configuration file. The *import phase* starts with the creation of the correct tree directory structure, using system calls, in order to adapt the IP-Core definition to an EDK compatible one; if one of these calls does not succeed, the program rises a warning, but does not abort its computation, this is useful in the case a directory already exists, but the files are not yet correctly stored. After ending the directory structure creation the program copies the files stored in the repository, whose name respects naming requisites, to the correct directory, copying also all the necessary files for the correct interface architecture. At the end of this process, the correct PAO and the MPD files are created in the correct location, e.g $correct_path/data. Also during this part of the process, if one of the operations fails the program just stores the error in a log file but does not abort itself, for the same reason explained for the tree directories creation.

3.2.2 *Insertion of the IP-Core*

After the first phase, Section 3.2.1, the IP-Core has been correctly imported into the available IP-Core list of the architecture, but it has not been inserted into the base architecture. This action has to be done by the second phase of the EDK System creator tool. This second phase consists in the connection of the IP-Core to the correct bus and its correct integration in the base architecture.

To achieve this point the program must modify two files that are in the base architecture directory, and are named system.mhs, containing a hardware description of the connection between IP-Core and architecture, and system.mss, defining the software description. In order to know which part of these files must be modified, this tool needs to know the IP-Core name that has been provided by command line at the tool invocation. If no name has been passed to the startup, the program aborts itself. If no error occurs, at this point the EDK system creator modifies the system.mss and the system.mhs files to add the characteristics of the new IP-Core. In the last file the EDK System creator correctly sets the name of the instance (usually IP-Core_name_-component_1) and the bus to connect to the new IP-Core. Now the IP-Core is fully linked to the architecture and, once a suitable driver for it is written, it can be synthetized and executed on the target FPGA.

4. A REAL CASE STUDY: CARONTE

In this section we will discuss how to apply the proposed methodology into a real working flow, Caronte, for the implementation of a dynamic reconfigurable system using a common FPGA. The Caronte methodology could be applied within any specific device just porting it to a different design technology, [5]. In order to show the possibility of implementing the *reconfiguration design flow* we decided to use the Xilinx tools but it could be easily ported to be reused for different systems that can achieve embedded dynamic reconfiguration [1].

The Caronte Flow is mainly composed of three phases:

HW–SSP Phase The HardWare Static System Photo Phase identifies a set of EDK system descriptions that will be (partially) dynamically reconfigured at run–time. These functional blocks are called *BlackBox cores*, see section 4.2 for a deep definition of this component. In the Caronte flow this phase can be considered as the most expensive one, although this is a very repetitive activity, according to the time that the designer has to spent to define each HW-SSP. Using the IP-Core generator tool to define all the BlackBoxes and the EDK system creator tool to create all the HardWare Static System Photo needed to represents all the feasible system configuration, the fulfillment time can be drastically reduced, as shown in the section 5.

Design Phase Aim of this phase is collecting all the information needed to compute all the bitstreams to physically implement the embedded reconfiguration of the FPGA.

It 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 Modular Based Design approach;
- Identify, using the *Floorplanner* tool provided in the 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 Phase This phase creates all the bitstreams needed to implement the system description onto an FPGA through the dynamic embedded reconfiguration.

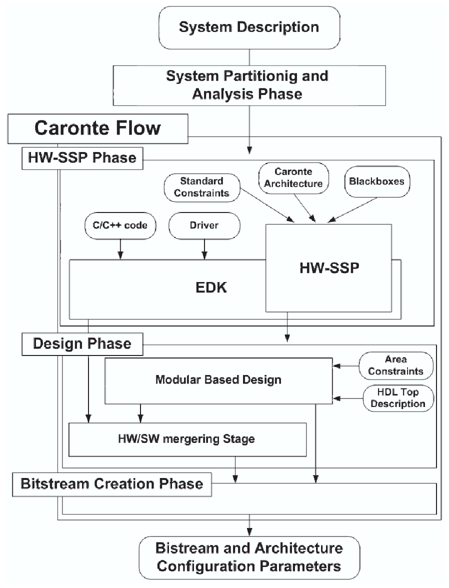

Figure 1: Caronte design flow.

Figure 1 shows the described methodology and how it can be included into the standard FPGA flow.

4.1 The caronte architecture

Let us first describe the architecture of the solution resulting from the Caronte Methodology described in [5].

In order to manage reconfiguration internally it is necessary to always have a processing element running on the chip that communicates with the ICAP port. This means that it will be necessary to have a part of the FPGA which always remains the same during runtime (the *fixed side*, managing reconfiguration, while the rest of the available area is free for dynamic reconfiguration (the *reconfigurable side*).

Hence the area of the reconfigurable part is divided in rectangular boxes, all sharing a minimal interface that allows them to interact with the rest of the system, such as the

IBM CoreConnect bus. These boxes are called *BlackBox*es, 4.2. Aside from that, each BlackBox has also a *processing layer* part which can be reconfigured to various tasks, always retaining the communication functionalities offered by the communication layer. The *bus macro technology* is used to establish fixed routing channels between modules. From the implementation point of view, this means that each BlackBox is in fact an EDK component made up of two VHDL, Verilog or EDIF files, the first one containing the *architecture–dependent* logic interface and the second one the *processing element* hardware description and that is exactly the perfect scenario to use the IP-Core generator tool to have a fast implementation of each BlackBox, focusing the attention only on the processing element description.

4.2 Fast BlackBox generation

The BlackBox generation phase is a *static* and *automatic* process that takes a lot of time if implemented using the EDK import peripheral wizard. The user has always to repeat the same actions for each BlackBox, starting from a VHDL description, create an EDK IP-Core component without changing the communication interfaces and insert the resulting IP-Core in the EDK system architecture.

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 XPS 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 all the interfaces needed by the VHDL description of a processing element to dialog with all the other components of the architecture, such as the CoreConnect bus, the processor, the interrupt controller and the other BlackBoxes. This operation is part of the HW-SSP phase described in section 4. Defining an HardWare System Static Photo basically means to create an EDK system that is composed of all the necessary IP-Cores to complete a specific task that can be assigned to the system in a certain time during its computation. That means that the designer has to create all the EDK system architecture just preserving the core part of the architecture, which remains always the same, and plug into it the IP-Cores needed. According to this description is easy to see that, as the BlackBoxes generation phase, also this phase is characterized by the fact that it is extremely long and moreover it always remains the same. In order to speed up this process the designer can use the EDK system creator that will autonomously create all the HW-SSPs without any further waste of time.

5. TEST AND RESULTS

The flow described in the previous section has been tested under different operating systems and architectures with a variety of components, starting from some small IP-Core such as an adder, a xor, two different multipliers to some more complex examples, e.g a Direct Fourier Transformation core, various implementations of the AES algorithm, a Siemens description of a complex ALU and a video editing core that changes the image coloring plane from RGB to YCbCr, as shown in Table 1. The proposed flow has successfully passed all these tests, generating working components, imported them into an EDK architecture and finally the resulting bitstreams have been downloaded onto the VirtexIIPro, xc2vp7, FPGA to verify their correctness. Table 1 presents some results produced by the IP-Core generator tool. The input component of the tool is the core component implemented by the designer, while the output of the system is the final EDK-compliant IP-Core.

Table 1: IP-Core generator tool tests

IP-Core	4-input luts	Perc.	Slices	Perc.	Time (s)
Core: Mult2	64	1%	37	1%	
IP-Core: Mult2	339	4%	205	4%	0.053
Core: IrDA	15	1%	11	1%	
IP-Core: IrDA	146	1%	103	2%	0.045
Core: FIR	273	2%	153	3%	
IP-Core: FIR	308	3%	173	3%	0.058
Core: AES128	4124	42%	2132	43%	
IP-Core: AES128	4314	44%	2250	46%	0.075
Core: RGB2YCbCr	1028	10%	913	18%	
IP-Core: RGB2YCbCr	848	9%	940	19%	0.063

The time of elaboration is quite constant and on the average is of 0.065 seconds. The range in which elaboration time is located starts from 0.045 seconds and ends to 0.075 seconds, as shown in Table 1, while for the EDK system creator the import phase is executed in about 0.2s, while the insert phase in about 10ms. According to these results the entire flow starting from the Core VHDL specification to its insertion into an EDK-based system architecture needs less than 0.3s.

6. CONCLUSIONS

The proposed approach allows the designer to focus only on the development of the VHDL core description without taking into account communication problems and without loosing time into plugging it into a prototyping system architecture. This goal is reached due to the definition of an automated flow for the interfacing process of IP-Cores and their integration into a complete system architecture without requiring the user interaction. Preliminary results show that the proposed methodology, implementing an IP-Core automate generator system, based on an EDK system description, provides a low cost approach to the fast generation and prototyping of embedded systems. Some improvements can be done by introducing a support also for other reconfigurable devices trying to keep abstract the problem as much as possible. It would be also interesting to extend the framework with other works such as the one proposed in [4].

7. REFERENCES

[1] Xilinx Inc. *Embedded Development Kit EDK 7.1i.* Xilinx Inc., 2005.

[2] IBM corporation. *The CoreConnect Bus Architecture, white paper.* International Business Machines Corporation., 2004.

[3] IBM Corporation. *IBM official website.* http://www-03.ibm.com/chips/products/coreconnect/.

[4] Tien-Lung Lee and Neil W. Bergmann. An interface methodology for retargettable fpga peripherals. In *Engineering of Reconfigurable Systems and Algorithms*, pages 167–173, 2003.

[5] Alberto Donato, Fabrizio Ferrandi, Marco D. Santambrogio, and Donatella Sciuto. Exploiting partial dynamic reconfiguration for soc design of complex application on fpga platforms. In *IFIP VLSI-SOC 2005*, 2005.