FAST IP-CORE GENERATION IN A PARTIAL DYNAMIC RECONFIGURATION WORKFLOW

Matteo Murgida, Alessandro Panella, Vincenzo Rana, Marco D. Santambrogio, Donatella Sciuto

Dipartimento di Elettronica e Informazione,
Politecnico di Milano
Via Ponzio 34/5
0133 Milano, Italy
email: {matteo.murgida, alessandro.panella, vincenzo.rana}@microlab-mi.net
email: {santambr, sciuto}@elet.polimi.it

ABSTRACT

Reconfigurable devices, such as FPGAs, introduce into the design workflow of embedded systems a new degree of freedom: the designer can have the system autonomously modify the functionality carried out by the IP-Core according to the application's changing needs while it runs. The Caronte methodology, based on the modular design approach, is a design workflow that allows the creation and the handling of a partial dynamic reconfigurable architectures using Xilinx FPGAs. In order to speed up its execution, it is important to succeed in quickly generate the EDK-based systems that the flow requires for the elaboration of the correct partial reconfiguration bitstreams.

To achieve this goal, an *IP-Core Generator* framework has been developed, it receives as input the VHDL description of the core functionality of a module, automatically produces as output an IP-Core suitable to be inserted into an EDK system. This binding can be performed in a faster way than using EDK to re-create each time the entire architecture, exploiting the *EDK System Creator* tool. IP-Core Generator can be used each time an IP-Core has to be created, and not only in a dynamic reconfigurability environment. Several tests are presented to validate the proposed methodology.

1. INTRODUCTION

In order to achieve a more flexible behavior, a large variety of applications in communication, computing and consumer electronics need to be able to change their functionalities after the system has been produced. Programmable devices, such as FPGAs, provide this kind of flexibility; moreover, an important feature of these devices is the ability to reconfigure at run time a portion of the system while the rest of the design is still working. The so-called *dynamic reconfiguration* can be exploited in many application fields, for instance to fulfill space requirements in small portable systems, to create a System-on-a-Chip with a very high level of flexibility, to realize adaptive hardware algorithms, and so on.

Xilinx inc. suggests in [1] two different styles of dynamic reconfiguration on a single FPGA: the **Difference-Based Partial Reconfiguration** and the **Module-Based Partial Reconfiguration**. The first approach can be adopted when small design changes are required, and it would be unnecessary to change an entire module. Otherwise, when one or more design blocks have to be reconfigured, the second approach is better suited to achieve the result desired. According with this observation the Module-Based approach implies the definition of distinct portions of an FPGA to be reconfigured, while the remainder of the system is *fixed*. These portions are called *reconfigurable modules*: when a partial reconfiguration is required, one or more of these modules are substituted by others which implements the needed functionality. The Module-Based approach can be considered as strongly connected to *Modular Based Design*, [2], based on the idea of a design implemented considering the system specification as composed of a set of several independent modules that can be individually synthesized and finally assembled to implement the desired system. Each of these modules is called *IP-Core*, that stands for Intellectual Property-Core, and carries out a particular functionality within the system. An IP-Core can be viewed as composed of two parts: the inner logic, referred to as the *core*, which implements the module functionality, and the communication logic, which allows the component to be plugged into a system and interact with other IP-Cores.

It is straightforward that in the design flow of an IP-Core a large amount of time is spent creating the communication infrastructure, that is a repetitive and standard activity. The idea is to provide a methodology to automatically generate the IP-Core, once provided the core functionality, with pos-

itive implications on the time to market required by a dynamic reconfiguration flow.

Section 2 briefly introduces an overview of existing approaches to this issue, with a particular attention to the EDK framework, provided by Xilinx inc. Section 3 will introduce the *Caronte* workflow for the partial dynamic reconfiguration, trying to explain the reasons why the proposed approach will perfectly fit into the Caronte flow. In Section 4 the proposed methodology for fast IP-Core generation is finally presented. Section 5 will present several tests used to validate the proposed approach while Section 6 draws the conclusions.

2. PREVIOUS APPROACHES

The Virtual Socket Interface Alliance (VSIA), [3], has developed the Virtual Component Interface, VCI, [4], that, separating the interface logic and the behavioral logic, simplifies the connection to different bus architectures. The separation between functionalities and interfaces is operated by a bus wrapper, which implements the connection with the bus. VSIA defines a standard interface between the behavioral logic and the wrapper called VCI. When new bus standards are introduced, a new VCI wrapper is designed. Therefore, the designer has to use the standard signals defined by VSIA in order to connect with VCI.

[5] proposes a similar methodology, which introduces less overhead than VCI. The core interface is defined by the Interface Adaptor Logic, that depends on the chosen communication architecture. The authors focus on the idea of core re-usability without introducing a solution that can reduce IP-Core generation time.

Xilinx provides the Embedded Development Kit, EDK, [6], a useful tool which allows the system designer to import a new custom core into a system architecture. The so-called *Import Peripheral Wizard* takes as input the core written by the user and creates the chosen interface in order to provide an IP-Core suitable to be inserted in an EDK system. The user, for instance, can choose between the two IBM CoreConnect standard interfaces, OPB IPIF or PLB IPIF. A limit to this solution is that the provided core must have a strict set of I/O signals that correspond to the inputs of the chosen IPIF interface, otherwise the designer is not going to be allowed to insert correctly the new core into the EDK library. The EDK Import Peripheral Wizard can be also used to create the *template* for a new core. This template has to be completed manually by the designer with the core specification. Since the approach proposed in this paper is based on EDK, let us provide some more details on this framework.

2.1. The EDK framework

EDK, [6], is a FPGA system design framework produced by Xilinx. One of EDK most important features is the possibility of developing complete systems, integrating the creation of both the software and the hardware components of the design in a unique platform. In fact, it provides the designer a rich set of tools and a wide number of peripherals suitable to build embedded systems running MicroBlaze or PowerPC processors, produced respectively by Xilinx and IBM.

These solutions adopt the IBM CoreConnect Architecture, [7], allowing the communication between components plugged into the same System-On-a-Chip (SOC). This architecture includes three buses [7]:

- PLB: The Processor Local Bus is a 64-bit primary bus designed to be used with devices that need high performance, low latency and design flexibility. It is a direct interface to the IBM PowerPC and Microblaze processors;
- OPB: The On-Chip Peripheral Bus is a 32-bit secondary bus designed to alleviate the PLB load and avoid system bottlenecks. It serves lower performance devices like serial and parallel ports, timers, etc;
- DCR: The Device Control Register bus is used only to reduce the capacity loading on the PLB and OPB buses.

Any custom IP-Core that will be inserted into a system based on this kind of architecture must be compatible with the OPB or PLB buses. Xilinx has developed an interface module, called IP-core InterFace (IPIF), in order to allow an easier binding between IP-core and bus; there are two versions of IPIF: PLB IPIF, for PLB attachment, and OPB IPIF, for OPB attachment.

The EDK graphical user interface is called XPS, Xilinx Platform Studio, it integrates tools with various purposes, from design entry to design debug and verification. It is very useful for combining more IP-Cores that need to communicate with each other through the CoreConnect bus architecture and realize the final system implementation that can be mapped onto Xilinx FPGAs. Moreover, XPS provides the capability of expanding its libraries adding custom IP-Cores implemented by the user.

Although EDK is a very powerful tool which allows the easy creation of a complete system description, it presents an unavoidable inconvenient aspect. In fact, each time the necessity to change or add one or more IP-Cores arises, there is the need to re-create the entire system, even if most of the architecture remains identical, implying a remarkable waste of time. To make the insertion of a new custom IP-Core into an EDK-based system faster and less resource-consuming a tool called EDK System Creator has been developed [8]. This tool, once received as input an already created EDK system and the VHDL description of a new IP-Core, sets all the necessary parameters to allow the IP-Core to be treated as an EDK component and inserts it into the final system.

EDK widely uses the Modular Design Style, meaning that the system is built starting from a set of modules that are successively connected with a suitable communication infrastructure. A remarkable feature is that EDK defines a module as an IP-Core provided with the right interface architecture, without considering the internal logic of the IP-Core. This implies that it is possible to instantiate a module composed only of an empty *shell*, characterized by the correct communication interface. This consideration is the base of the *Caronte* Partial Dynamic Reconfiguration framework, described in the next section.

3. THE CARONTE METHODOLOGY

In [9, 10] a design methodology flow for the creation of partial dynamic reconfigurable architectures has been introduced. This solution exploits the capabilities of Xilinx EDK, but can be applied to different design technologies. This workflow adopts the Modular-Based Partial Reconfiguration style, using one or more reconfigurable blocks.

The Caronte flow accepts as input the result of a previous partitioning and analysis stage, and then it creates a set of HardWare-Static System Photos (**HW-SSP**), that are complete descriptions of the system for different instants of the reconfiguration process. Afterward, some information has to be collected to allow the computation of all the partial reconfiguration bitstreams for the physical implementation of the dynamic system. During this phase Caronte identifies the structure and defines the area of each reconfigurable module, and creates a routing infrastructure to solve the communication between reconfigurable modules, introducing Bus Macros to provide a fixed and stable communication channel. Finally, all the partial reconfiguration bitstreams are created.

The Caronte hardware architecture can be viewed as the composition of two parts: the *fixed side* which manages the module's reconfiguration, and the *reconfigurable side*, composed of dynamic modules. The reconfigurable area of the FPGA is divided into several portions, that contain the reconfigurable modules, connected one another with the bus macro technology. These blocks are called *BlackBoxes*, and each of them can be viewed as a shell containing processing elements having its own area which implies that to each BlackBox is assigned a set of constraints for its correct placement. A BlackBox can be considered as an EDK component, but it is not a static module mapped on the FPGA, since it can contain different functional elements in different instants of time. From the implementation point of view, a BlackBox consists of two levels:

- the **processing element logic**, which is the IP-Core that implements the needed functionality;
- the **architecture-dependent logic**, which implements the connection between the inner logic and the bus macros.

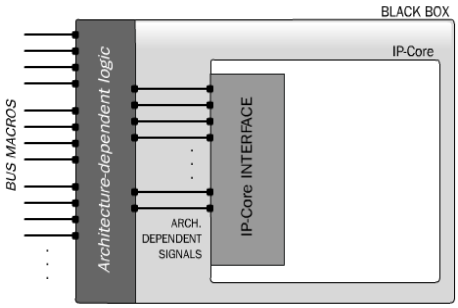

Fig. 1. BlackBox schematic.

Figure 1 represents this two level-structure. When the reconfiguration of a BlackBox is performed, only the processing element description, that is the IP-Core, is changed, while the communication infrastructure logic remains the same.

The first step of the Caronte flow is the HW-System Static Photo Phase which aims is the creating an EDK system description for each configuration in which the system can evolve during the execution. This phase is very time consuming mainly for two reasons: the designer has to design and correctly import his own IP-Core into EDK and once completed this first step it has to create all the necessary EDK-system description to implement all the HW-SSP desired. This processes can be easily implemented in an automated workflow which is composed, as the described situation, of two steps: an IP-Core Generation phase and an EDK System Creation phase, described in [8], which takes as inputs an EDK general system and a configuration file that contains the information about what IP-Cores are to be plugged in the different HW-SSPs. The proposed workflow is shown in Figure 2.

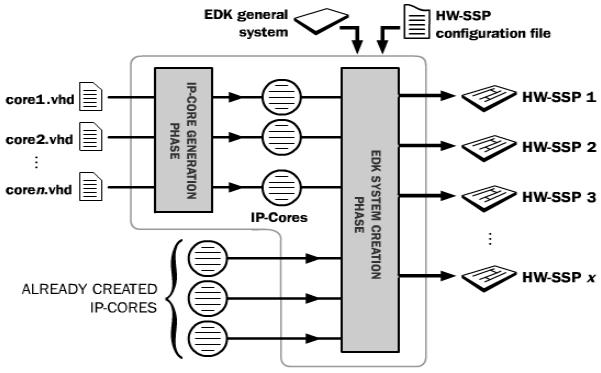

Fig. 2. HW-SSPs complete generation workflow.

The former problem is the starting point of the method-

ology described in the next section, which provides a solution that allows *Fast IP-Core Generation*.

4. THE IP-CORE GENERATOR FRAMEWORK

The design of an EDK-compatible IP-Core can be divided in two phases: the creation of the *core* and the implementation of the communication infrastructure. In order to allow fast IP-Core generation, the second step has to be performed automatically by a software tool, called *IP-Core generator*, which takes as input the VHDL file which implements the core and the name of the chosen communication architecture and produces as output the final IP-Core. The IP-Core Generator is composed of three main steps:

The input phase: the tool needs the VHDL description of the core and the indication of the chosen communication architecture;

The Reader: it reads, interprets and saves all the information needed by the following step. The main operations performed by the reader are the following:

- Recognize the pattern of a VHDL entity declaration in the VHDL core description;
- Build the generics and I/O signal lists: when a generic is recognized, it is analyzed and its information is stored in the generics list, this action is repeated until the end of generics declaration. The signals list is built in the same way;
- Save the core entity name and the file path in two variables used by the following process.

The Writer: it writes the IP-Core VHDL description, performing the following actions:

- Accept as inputs the generics and I/O signals lists, the core's entity name and the core's path;
- Re-write the VHDL description of the core, because it is necessary in some cases to execute little changes to *clean* the core file in order to produce a correct implementation of the complete IP-Core;
- Create a stub VHDL file between the Core and the IP-Core VHDL description, which allows the input signals of the core to be written by the bus master, and the outputs to be read;
- Create the top architecture VHDL file, that is the final IP-Core, which contains the processing logic and the chosen interface.

Therefore three output VHDL files are produced: the *cleaned* core, the stub and the top IP-Core description. The resulting IP-Core structure is represented in Figure 3

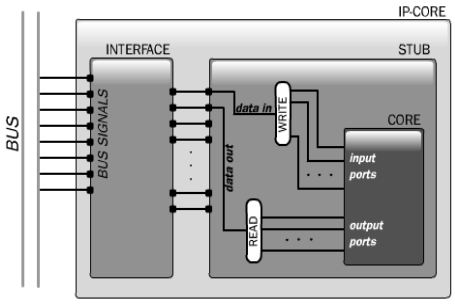

Fig. 3. IP-Core final structure.

During the execution, if one of the two phases fails the tool is halted and an error message, useful to understand where and why the problem occurred, is generated. If the execution ends correctly the created IP-Core is ready to be plugged into an EDK system. It is important to specify that the tool is not a VHDL parser, because it does not validate the analyzed code, except for the entity declaration syntax.

An important feature of the tool is that the address decoding logic is automatically generated and included in the stub. Therefore, by this solution, while creating a hardware module, the designer does not have to deal with the core interconnections within the system, focusing his efforts only on the functionality itself. In other words the core must not include any reference to the interfacing architecture.

4.1. IP-Core Generator in the Caronte workflow: fast HW-SSP generation

The IP-Core Generator can be used in the Caronte methodology, see Section 3, to speed up the execution of the flow. As a matter of fact, in the HW-SSP phase a large amount of time is spent building several complete architectures with EDK, that correspond to the HW-SSPs needed by the dynamic nature of the implemented system. The idea is to adopt IP-Core Generator and EDK system creator to make the creation of HW-SSPs faster.

The starting point is the creation, using EDK, of a system that implements the needed architecture, including both the fixed side and the reconfigurable side. The BlackBoxes included in this initial system are *empty*, meaning that no IP-Core fills the space inside the box shell, and are only composed of the architecture-dependent logic. This *general* system is the base that will be used to generate all the necessary HW-SSPs, just changing the IP-Cores contained in the BlackBoxes. The systems creation process can be done in a time-saving way avoiding to generate by hand the IP-Cores and re-create every time an entire EDK system.

This is possible since the IP-Core Generator can efficiently perform the automatic creation of the IP-Cores, implementing the chosen communication infrastructure. This process requires a very short time to be executed, as shown

in Section 5, especially if compared to manual generation. Once the IP-Cores are ready to be plugged into the Black-Boxes, the EDK system creator can be used to assemble all the designs that represent the steps that the reconfiguration process will pass through, just merging the initial general system with the modules required by each HW-SSP. This flow is schematically shown in Figure 4. At this point the execution of Caronte can proceed to the following Design Phase.

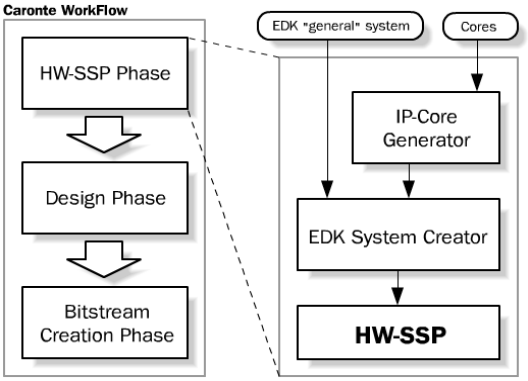

Fig. 4. Fast HW-SSP Generation.

Although IP-Core Generator has been introduced in this paper as a useful tool in the Caronte environment, it can be used every time the creation of an IP-Core is needed, which means every time a new component has to be instantiated in a modular-based system, using EDK or other development frameworks.

5. TESTS AND RESULTS

Several tests have been performed to validate the proposed workflow. Despite the fact that we decided to use Xilinx ISE framework to simulate components' behavior on an FPGA, the results have a general validity. As previously presented in section (2.1), Xilinx uses the IBM CoreConnect bus architecture, therefore we have tested the IP-Core Generator interfacing a set of cores with the OPB bus using both the PSelect [11] and the IPIF [12] interfaces. We have tested various components, both small and simple cores and more complex ones. This because we wanted to analyze the behavior of the tool with cores of different size and complexity. These cores are: a basic 0-to-n counter, a 32-bit adder with overflow detection, a systolic serial parallel squarer of unsigned numbers, a single chip frequency counter, a FAT16 hardware file reader, a division unit, a complex ALU and a complete Floating Point Unit.

All the designs have been simulated on Xilinx VirtexII-IPro xc2vp7 FPGA, except for the Floating Point Unit for which, because of space requirements, VirtexIIPro xc2vp30 has been used. The tests done using OPB IPIF are shown in Table 1.

The results listed in Table 1 show that there is no correlation between the core size and the overhead occupation. The amount of combinatorial and sequential logic needed for the physical implementation of the stub is supposed to depend on the number and the size of core's I/O signals, while the occupation of the IPIF is theoretically constant. Despite of these considerations, it is difficult to find out in the table a direct connection between the ports of the cores and the occupation of the stub, and IPIF dimension is far from being constant. These results are due to the ISE Synthesis process which operates both a space optimization, decreasing the number of registers by deleting unused or unconnected ones, and time optimization, carried out, for example, by duplicating some registers in order to achieve a satisfying processing speed ([13], [14]). This feature can explain the presence of a negative number in the table. Anyway, it is evident that the percentage of space required by the introduced overhead dramatically decreases as the dimension of the core becomes bigger, as clearly shown in figure 5.

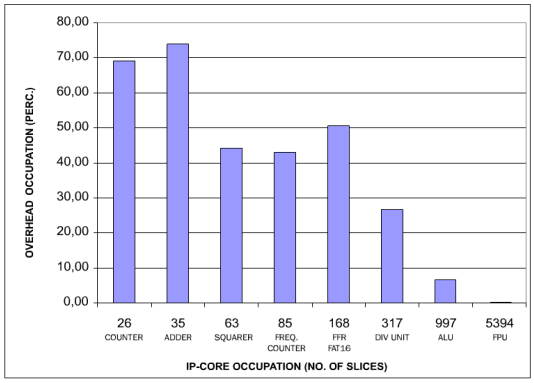

Fig. 5. Overhead Occupation.

Therefore, from this point of view, the best case is to have a big design with few and small ports, while the worst one is to have a little core with many I/O signals. This result may suggest to use bigger cores, for example putting together in a unique module different functionalities. In system design, especially in a dynamic reconfigurable architecture, this solution has to be adopted carefully, for two main reasons: it implies a decrease in the degree of flexibility, that is one of the main features of modular based design, and the time required by the reconfiguration becomes longer.

Hence it is desirable to reach a reasonable trade-off between the size of the modules and the occupation of the overhead required by the communication logic.

Figure 6 shows the execution times for the proposed framework, obtained running the tool several times for each core and making an average of the outcomes, the dark line represents the IP-Core generation using the IPIF interface,

Table 1. IP-Cores' occupation (in slices) using IPIF.

	ALU	Div Unit	FFR-FAT16	F. Counter	Squarer	Adder	Counter
Core	997	317	168	85	63	35	26
Stub	48	39	91	29	26	37	19
Core + Stub	1045	356	259	114	89	72	45
IPIF	17	60	80	35	24	62	39
IP-Core	1065	416	339	149	113	134	84
Overhead	68	111	171	64	50	99	58
Overhead %	**6.38%**	**26.68%**	**50.44%**	**42.95%**	**44.22%**	**73.88%**	**69.05%**
I/O signals dim. (in bit)	140	96	80	42	34	33	11

while the light one the PSelect.

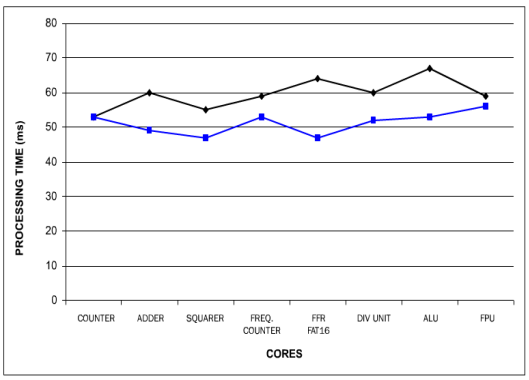

Fig. 6. IP-Core Generation Timing Graph.

Using the IPIF interface the average is about 60 ms, with a standard deviation of 4.47 ms. In the PSelect case the average is about 51 ms while the std deviation is 3.24 ms. Therefore, the running time of the tool can be considered with a good approximation constant.

6. CONCLUDING REMARKS

Section 5 completely validates the IP-Core Generator tool, which is able to quickly generate IP-Cores ready to be used in a modular based system. The aim is to expand the range of applications of the tool, adding full support for a large set of architectures and interfaces. The ideal goal is the creation of a framework in which the user can autonomously import or implement the wanted interface and use it to generate IP-Cores.

7. REFERENCES

[1] Xilinx Inc. Two flows for partial reconfiguration: Module based or difference based. *XAPP290*, September 2004.

[2] Xilinx Inc. Development system reference guide. 2005.

[3] Virtual socket interface alliance. http://www.vsi.org/.

[4] Annette Bunker and Ganesh Gopalakrishnan. Formal specification of the virtual component interface standard in the unified modeling language. In *UUCS-01-007*, 2001.

[5] 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.

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

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

[8] F. Ferrandi, G. Ferrara, R. Palazzo, V. Rana, and M. D. Santambrogio. Vhdl to fpga automatic ipcore generation: A case study on xilinx design flow. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06) - Reconfigurable Architecture Workshop - RAW*, 2006.

[9] 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.

[10] Alberto Donato, Fabrizio Ferrandi, Massimo Redaelli, Marco D. Santambrogio, and Donatella Sciuto. Caronte: a complete methodology to implement partially dynamically self-reconfiguring embedded systems on modern fpga. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2005)*, 2005.

[11] Xilinx Inc. *Designing Custom OPB Slave Peripherals for Microblaze*, 2002.

[12] Xilinx Inc. *OPB IPIF (v. 3.01b)*, 2005.

[13] Hitesh Patel. *Synthesis and Implementation Strategies to Accelerate Design Performance. Xilinx White Paper 229*, 2005.

[14] Kevin Bixler and David Dye. *Synthesis and Implementation Strategies to Accelerate Design Performance. Xilinx White Paper 230*, 2005.