Core Communication Interface for FPGAs

José Carlos Palma, Aline Vieira de Melo, Fernando Gehm Moraes, Ney Calazans
{jpalma, alinev, moraes, calazans}@inf.pucrs.br

Abstract

The use of pre-designed and pre-verified hardware modules, also called IP cores, is an important part of the effort to design and implement complex systems. However, many aspects of IP core manipulation are still to be developed. This paper presents an approach to solve problems related to the dynamic interconnection of hard IP cores. The approach targets system-on-a-chip designs build in a single FPGA device. The paper proposes a communication interface that allows IP cores replacement during the FPGA normal operation. The same interface also allows the communication among distinct IP cores to take place.

1. Introduction

Cores are complex pre-designed and pre-verified hardware modules, which can be considered as key component in the development of system-on-a-chip (SOC) designs. According to a recent ITRS report [1], by 2012 90% of the area of integrated circuits area will be implemented using cores.

Cores are used (or integrated) during system design, i.e., before logical or physical synthesis [2][3]. For ASIC designs, this corresponds to the normal flow. However, for FPGAs a different scenario can be envisaged: dynamic hard-core insertion and removing at run time. This idea allows inserting a hardware module into the FPGA according to the application requirements.

For instance, considering the situation where a FPGA board is connected to a host computer and a new piece of hardware may be necessary to speed up the processing in a graphic application. This piece of hardware (hard-core) could be downloaded through the Internet and sent to the FPGA.

Another example could be the "dynamic co-design". The hardware generated by a co-design tool requires a FPGA with sufficient area to implement all the application modules. Using the proposed approach (hard-core reuse), the co-design tool could generate small modules, loading them according to a timing schedule. This approach is similar to the "virtual memory" concept, found in traditional operating systems and could be named "virtual hardware" [4].

To achieve hard-core reuse some requirements have to be fulfilled: (i) FPGA allowing partial and dynamic reconfiguration; (ii) floorplanning tool to fix the core position; (iii) software for partial bitstream generation; (iv) software for partial bitstream download; (v) a core communication interface; (vi) input/output pin virtualization. Requirements (i) to (iii) can be satisfied with existing devices and tools. Works fulfilling requirements (iv) to (vi), in our knowledge, are very scarce or simply do not exist.

The core communication interface is a critical problem, since it must allow dynamic insertion and removal of cores without system interruption.

This paper is organized as follows. Section 2 presents the requirements for hard-core reuse. Section 3 details the main contribution of this work, the communication interface. Preliminary results are presented in Section 4. Section 5 presents some conclusions and directions for future work.

2. Requirements for Dynamic Hard Cores Reuse

Reconfigurable computing mostly emphasizes the use of coarse grain reconfigurable arrays with path widths greater than 1 bit, like in fine-grained architectures. Fine-grained architectures, like commercial FPGAs, are much less efficient with a huge routing area overhead and poor routability [5].

In spite of this fact, our goal is to investigate the feasibility of implementing reconfigurable computing systems into commercial FPGAs. The granularity of our approach is an important point, and it can be classified as coarse-grain, since the reconfiguration is achieved with hard cores. The next item detail the requirements discussed in the introduction.

- **FPGA Architecture –** Examples of available FPGA devices allowing partial and dynamic reconfiguration are Xilinx Virtex [6] and Atmel At40k [7]. Xilinx Virtex FPGAs have been selected as the target devices for this work, because of facilities such as satisfactory CAD tools, sufficient gate counting at affordable costs, and diversity of prototyping boards available.
- **Floorplanner –** Partial reconfiguration is only possible if the cores to be inserted do not overlap the existing cores inside the FPGA. So, placement restrictions must be inserted during the physical synthesis. Example of such tool is the Xilinx floorplanner [8].

- **Partial bitstream generation –** Tools provided by FPGA vendors do not generate partial bitstreams. JBits classes [9] can be used as an abstraction layer hidden the Virtex FPGA configuration details.
- **Communication interface –** In the early stages of SOC development, cores are designed with different interfaces and communication protocols. Some standard interfaces, such as Wishbone [10], AMBA [11] and CoreConnect, were created to increase core reuse. These interfaces are used during the design phase of the SOC, for ASIC and FPGA flows. After the SOC implementation, it is not possible to connect a new core to the system.
- **I/O Pins Virtualization –** I/O pins virtualization means that cores exchange data only with the communication interface and the real input/output communication is a function of the interface.

3. Communication Interface

The basic idea is outlined in Fig. 1. A fixed module, named *controller*, is initially downloaded into the FPGA. Functional cores, named *slave cores*, are downloaded at run time. The controller is responsible for the communication with the external world (I/O pins of the device) and for the communication with slave cores. Each slave core communicates with the controller through *virtual pins*. Virtual pins are in fact tri-state buffers, having the position defined by the floorplanning tool. The connection between slave cores and the controller is done by overlapping virtual pins.

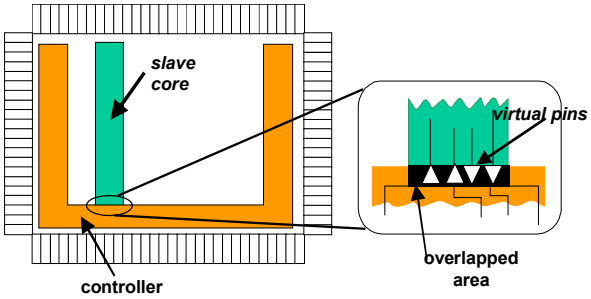


Fig. 1 - Communication interface with one tri-state buffer layer (tri-state control signals are not show).

The interface proposed in Fig. 1 is not feasible due to restrictions imposed by the FPGA architecture. Such restrictions are:

- Input and output of tri-state buffers could be used as virtual pins by slave cores. As it is not possible to fix wire positions, the controller and slave cores must use a tri-state buffer as virtual pin.
- Each CLB contains LUTs, flip-flops, tri-state buffers and routing resources. It is not possible to restraint the CLB usage only to tri-state buffers and routing. Therefore, overlapping slave cores over the controller could destroy some controller functions.
- Each CLB contains only 2 tri-state buffers. The reduced number of buffers limits the bus width. In addition, the buffer routing wires share common hex lines [6], restricting the routing tool. Due to this limitation, we adopted a serial bus (1-bit data line).

In order to overcome these limitations, a communication interface with two tri-state buffer layers with common routing wires is implemented. To have common routing wires the controller is synthesized by using "dummy cores", which have the buffers belonging to the slave cores. The same procedure is applied to slave cores, which are synthesized with a "dummy controller". "Dummy cores" are also important to avoid floating signals in the communication interface.

3.1. Controller Implementation

The controller contains three modules, illustrated in Fig. 2: *Communication bus*, connecting the slave cores; *Arbiter*, granting the data line to a given slave core; *Master core*, responsible for the communication to the external world.

The communication bus has the following signals: reset (global), clock (global), individual request and grant lines for each slave core, and a bi-directional data-line (global). Each slave core communicates with the bus through six tri-state buffers, connected to the following signals: *clock*, *reset*, *request*, *grant*, *datain* and *dataout*. *Datain* and *dataout* are connected together to the data line. The 1-bit serial data line transports a 40-bit word, containing an 8-bit core address and a 32-bit data word. A simple protocol is employed, using a starting bit to indicate the transmission of a new word.

3.2. Case Study

A simple case study was implemented to validate the communication interface. Tree modules compose this

application, as illustrated in Fig. 2: (1) controller (2) slave1; (3) slave2.

The function of each slave core is very simple: reception of two 32-bit words, execution of an arithmetic operation over these words (*slave1* executes addition and *slave2* subtraction), sends the result to the master core, waits two new words.

The master core has three internal registers, acting as data memory, and a program memory holding the operations. The program memory is implemented in LUTRAM, being its contents defined by specialized reconfiguration tools.

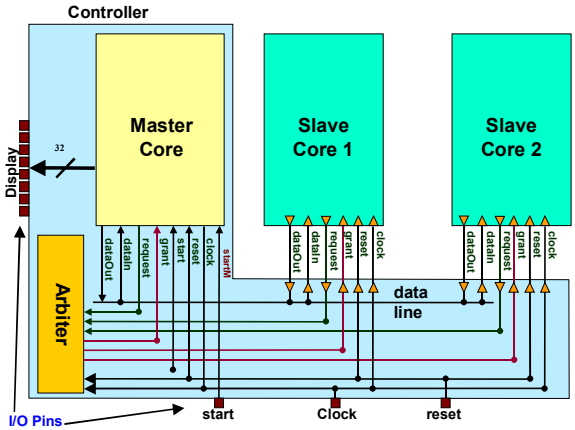


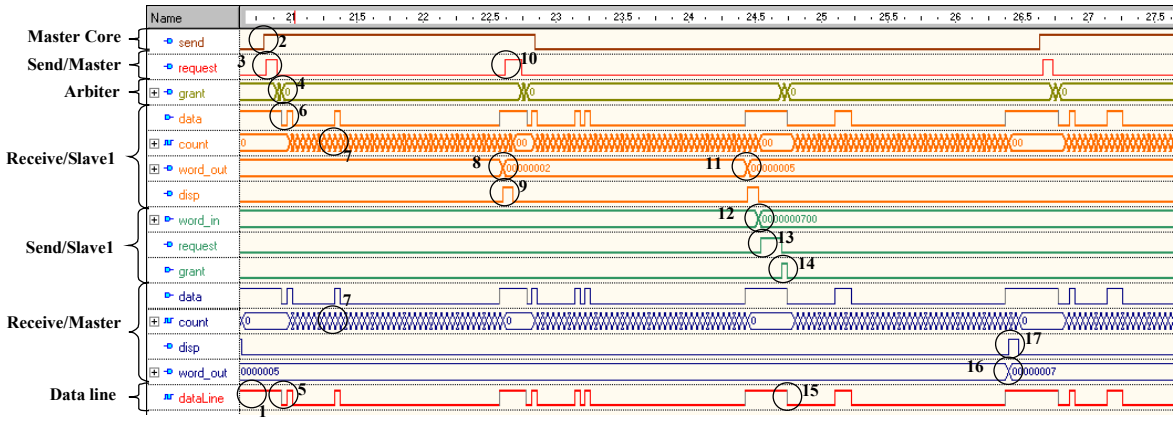
Fig. 2 - Controller modules: master core, arbiter and communication bus. Excepting the tri-state buffers controlling the dataOut line, the control line of all other tri-state buffers are set to '1', i.e. conducting.

4. Preliminary Results

This section presents preliminary results: functional communication interface validation, partial bitstream generation, and CAD tool for core manipulation.

4.1. Functional validation

All modules are described in VHDL language, and simulated using the Active-HDL simulator, from Aldec [12]. Fig. 3 illustrates the system operation, using the case study presented in section 3.2.



1. Initially, the data line is in steady state (logic state 'HIGH').
2. The master core asserts its internal send signal, indicating it has available data to transfer.
3. The master's send module asserts the request signal, and waits for the grant signal.
4. The grant signal is sent to the master's send module.
5. Data transmission starts, and the first '0' correspond to the start bit.
6. Receive modules detect the start bit, starting data storage.
7. After 8 clock cycles, all receive modules compares the stored byte to the internal address. If they match, the receive module continues to store the arriving bits. If they do not match, the receive module waits 32 clock cycles to read again the data line.
8. 40 clock cycles after the starting bit a new 32-bit word is available.
9. The slave1 receive module signs an available word to the hw-core.
10. At the end of the first transmission, the master's send signal is still in logic '1', indicating that it wants to send the second operator. A new request is sent to the arbiter and operations 4-9 are repeated.

Fig. 3- Functional simulation of the communication interface.

Once the system is functionally validated by simulation, the next step is to prototype it. The prototyping board *Virtual Workbench* [13], with a Virtex XCV300, was used. The system works correctly in the prototyping board, showing that the communication interface, with tri-state buffers, can be used in Virtex devices to connect hard cores.

4.2. Partial bitstream generation

Once the communication interface is functionally validated, the next step is to generate an individual bitstream for each module. The floorplanning tool is used to restrict the physical position of each module. Each slave core is synthesized with a "dummy controller" and the controller is synthesized with "dummy cores", to ensure the common routing wires. The main difficulties to generate these bitstreams are: (i) Placement of the tri-state buffers in fixed positions. Due to the scarceness wire resources, the routing may fail; (ii) The logical synthesis tool can eliminate tri-state buffers connected to the FPGA outputs. Problem solved with constraints inserted in the design; (iii) Clock signals are routed with global clock lines. Constraints are applied to the routing tool to avoid the usage of such resources; (iv) Ensure that the routing will be restraint to the reserved module area; (v) Ensure the same routing between tri-state buffer layers. This is the critical restriction. The routing is manually executed between signals belonging to the communication interface, and after all other signals have been automatically routed.

Complete bitstreams are created, one for each module. Dedicated tools were implemented to extract a partial bitstream from complete bitstreams, since the employed CAD tools do not generate partial bitstreams.

The last validation step is to insert the slave core into the controller core, achieving partial and dynamic reconfiguration. A CAD tool to insert or to remove cores in a FPGA, named *core unifier,* was developed. This tool creates a partial bitstream, containing the modified area. Partial reconfiguration is then executed, inserting a new core into the FPGA. At present, the tool creates complete and partial bitstreams. Complete bitstream download with core insertion was achieved successfully, however partial bitstream fails due to the lack of partial download tools.

5. Conclusion and Future Work

The main contribution of this work is the presented method to reuse hard cores in FPGAs. The requirements to achieve reuse were presented and the proposed communication interface is described. The communication interface is functionally validated, both by simulation and by prototyping. Customized tools were developed to merge independent bitstreams and to create partial bitstreams.

This work shows that it is possible to implement "virtual hardware" with commercial FPGA devices. The main obstacles to develop automatic CAD tools are the internal architecture of these devices, which are quite difficult to be used for partial reconfiguration, requiring complex manual operations (manual placement and manual routing).

As suggestions for future work we can enumerate: (i) to extend the bus structure to more bit lines and different bus arbitration schemes; (ii) to develop CAD tools to automate the manual steps mentioned above; (iii) to develop techniques for core relocation.

6. References

[1] International Technology Roadmap for Semiconductors, 2001, http://public.itrs.net.
[2] Bergamaschi,R.A.; et.al. **Automating the design of SOCs using cores.** IEEE Design & Test of Computers, Volume: 18(5), Sept.-Oct. 2001, pp. 32-45.
[3] Keating,M.; Bricaud,P. **Reuse methodology manual for system-on-a-chip designs**. Kluwer Academic Publishers, Norwell, MA, 1999, 286p.
[4] Dehon, A.; Caspi, E.; Chu, M.; Huang, R.; Yeh, J.; Markovsky, Y.; Wawrzynek,J. **Stream computations organized for reconfigurable execution (SCORE): Introduction and Tutorial**. Proceedings of Field-Programmable Logic and Applications, 2000.
[5] Hartenstein,R. **A decade of reconfigurable computing: a visionary retrospective** Design, Automation and Test in Europe, 2001, pp. 642 –649.
[6] XILINX. **Virtex 2.5V Field Programmable Gate Arrays (DS003).** Virtex Series datasheet, 2000.
[7] ATMEL. **At40k Series Configuration**. http://www.atmel.com/atmel/acrobat/0896s.pdf.
[8] **XILINX. Floorplanner Guide 3.1i**. 2001.
[9] James-Roxby,P.; Guccione,S.A. **Automated extraction of run-time parameterisable cores from program-mable device configurations.** IEEE Symposium on Field-Programmable Custom Computing Machines, 2000, pp. 153 -161.
[10] Opencores. **WISHBONE SOC Interconnection**. http://www.opencores.org/wishbone.
[11] Flynn,D. **AMBA: enabling reusable on-chip designs.** IEEE Micro, Volume: 17(4), July-Aug. 1997, pp. 20-27.
[12] ALDEC, INC. **Active-HDL 5.1**. 2001. http://www.aldec.com/ActiveHDL/default.htm.
[13] VCC Inc. **The Virtual Workbench**. 2000. http://www.vcc.com/vw.