Development of a Tool-Set for Remote and Partial Reconfiguration of FPGAs

Fernando Gehm Moraes, Daniel Mesquita, José Carlos Palma, Leandro Möller, Ney Calazans
Faculdade de Informática - PUCRS - Porto Alegre -Brazil - moraes@inf.pucrs.br

This work describes the implementation of digital reconfigurable systems (DRS) using commercial FPGA devices. The main goal is to present a set of tools for remote and partial reconfiguration developed for the Virtex FPGA family. Even though the tools are targeted to a specific device, their building principles may easily be adapted to other FPGA families, if they have an internal architecture enabling partial reconfiguration. The main contribution of the paper is the tool-set proposed to manipulate cores using partial reconfiguration in existing FPGAs.

Currently, only two FPGA vendors support partial and dynamic reconfiguration. One of them, Atmel, e.g. produces the FPSLIC, a device including a GPP (general-purpose processor), memory and programmable logic in the same integrated circuit. FPSLIC supports partial and dynamic reconfiguration through context switching [1]. The second vendor, Xilinx, offers e.g. the Virtex family, which also supports partial and dynamic reconfiguration. Reconfiguration is possible because internal configuration elements of this device can be individually addressed [2]. The Virtex family was chosen due to its widespread availability in the market.

Interest in reconfigurable computing has been growing in the past two decades [3]. The evolution of DRS is shown in Figure 1. The first generation comprises systems aiming to increase performance over GPPs, using off-the-shelf FPGAs. The second generation comprises architectures aiming the bottleneck minimization between GPP and FPGA, and reconfiguration techniques. Examples of fine-grain SOCs are FIPSOC and TRUMPET, and of coarse-grain SOCs are GARP and RAW. Dynamic reconfiguration can be achieved by context switching with DPGAs, or by partially reconfigurable devices, like Virtex devices. The third generation is characterized by architectures target to dataflow-based algorithms used in multimedia applications and hardware virtualization.

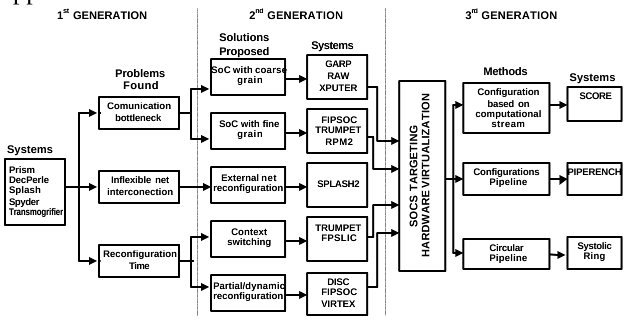

Figure 1- Evolution of reconfigurable architectures.

Usually, a circuit has a set of parameters defining its behavior, being loaded from an external ROM. The function of the ***circuit customization*** tool is to simplify the design, storing parameters directly into the bitstream, without using ROMs or external microcontrollers. Parameters are stored into FPGA memory blocks (e.g. Xilinx LUTRAM blocks), being modified by local or remote reconfiguration. This approach reduces the overall system cost, since it eliminates the need of external devices and/or the associated control logic to allow setting parameters at running time.

A design constraint is that parameters that are to be customized must be associated to a set of LUTRAMs or BLOCKRAMs at fixed positions. Once the initial bitstream is created, the tool helps the designer to create an interface giving access to the parameters. The user downloads his design into the FPGA. Using the interface he may change the parameters at will. It should be noted that partial reconfiguration is used, changing only the FPGA columns containing the specified parameter memory blocks.

There are three actors involved in this tool: the *software developer*, the *circuit designer*, and the *circuit user*.

The *software developer* implements a software layer hiding FPGA architecture details. This software layer is implemented as a Java applet. The applet communicates with the server. The server uses Jbits classes to open/write bitstreams and to access and modify the information contained in the bitstream. This applet is the same for all circuits being customized.

The *circuit designer* uses HTML tags to pass commands and parameters to the applet to customize his circuit. For each parameter the circuit designer specifies: (i) signal name; (ii) format – e.g. binary, decimal, hexadecimal; (iii) physical position of the parameters inside the FPGA, defined by row, column, F/G LUT, slice; (vi) starting and ending bits in the LUTRAM.

Finally, the *circuit user* receives the bitstream and the HTML description responsible to create the reconfiguration page. In the reconfiguration page the values of the signals can be modified, saved and partially downloaded into the device. Therefore, the circuit user can abstract all details concerning the FPGA architecture, and carry out remote and partial reconfiguration.

An important comment is that this tool is addressed to the same goal as the small bit manipulations proposed in [5], but offering a much higher degree of abstraction to its user.

The second tool developed is named ***core unifier***. A fixed core, named controller, is initially downloaded into the FPGA. The controller contains three cores: (*i*) *communication bus*, connecting the slave cores; (*ii*) *arbiter*, granting the data line to a given slave core; (*iii*) *master core*, responsible for the communication with the external world. Other cores, named *slave cores*, can be downloaded at run time.

Each slave core communicates with the controller through virtual pins. To have common routing wires the controller is synthesized using "dummy cores", which include the buffers belonging to the slave cores. The same procedure is applied to the slave cores, which are synthesized with a "dummy controller". "Dummy cores" are also important to avoid floating signals in the communication interface.

This tool creates partial bitstreams, working as follows:

1. A complete *master bitstream* is opened. It contains the *controller* and the *dummy cores*. The *controller* is connected to the *dummy cores* by wires connecting pre-placed (by floorplanning) tri-state buffers.
2. One or more complete bitstreams containing cores to be inserted into the *master bitstream* are opened. Each bitstream contains one core and a *dummy controller*. The user selects the area corresponding to one core, and all components inside this area (routing and CLBs) are inserted into the master *bitstream*.

This procedure is illustrated in Figure 2.

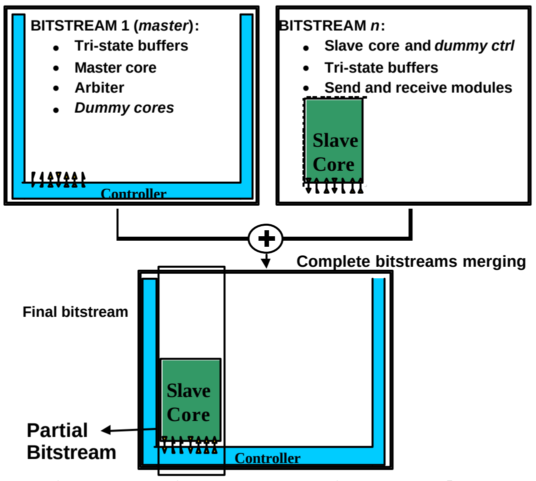

Figure 2 - Bitstream merging procedure.

Figure 3 presents the main window of the *core* unifier tool. This window has a 48*x*32 grid, representing all CLBs of a Virtex XCV300 device being different for other devices. Light and dark gray squares represent CLBs not used (default values). Red squares represent CLBs used by the *master bitstream*. Squares with different colors (e.g. yellow) represent inserted cores. The user can insert new cores into the master bitstream, a feature that adds flexibility to the tool, allowing dynamically inserting and/or removing cores.

This tool permits to implement virtual hardware, in the same manner as virtual memory. As a function of some execution scheduling these may be partially downloaded into the FPGA.

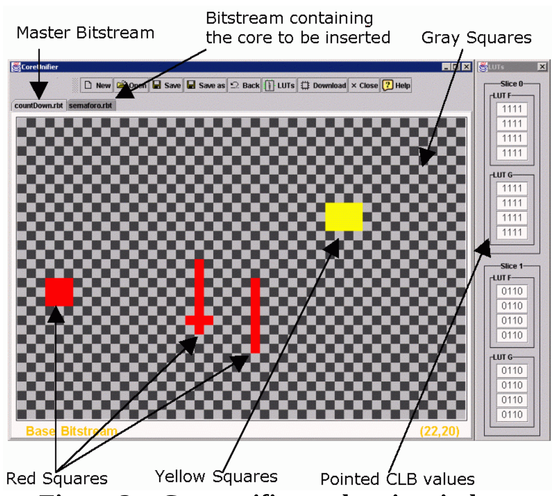

Figure 3 – Core unifier tool main window.

Three main problems exist in this approach, all related to the current state of commercial FPGA devices and associated CAD tools: (*i*) it is hard to constrain the core logic to reside inside the core bounding box defined by floorplanning; (*ii*) it is not possible to constrain routing with the floorplanner; (*iii*) it is not possible to define exactly the same wiring between tristate buffers. To obtain a synthesized core restricted to a fixed area, several routing iterations are performed, requiring even manual user intervention. This can be compared to the manual manipulations proposed in [3] and in [5] to verify that FPGA vendor tools must evolve to better support partial and dynamic reconfiguration.

As suggestions for future work it is possible to 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. Core relocation is the possibility of loading the same core at different places inside the FPGA.

References

1. ATMEL. **Field Programmable System Level Integrated Circuits (FPSLIC)** (Aug. 2002). http://www.atmel.com/atmel/products/prod39.htm
2. XILINX. **Virtex series configuration architecture user guide**. Application Note nb. 151, http://www.xilinx.com/xapp/xapp151.pdf (Mar. 2000)
3. Hartenstein, R. **A decade of reconfigurable computing: a visionary retrospective**. In: Design, Automation and Test in Europe, pp. 642 –649, 2001.
4. Dyer, M.; Plessl, C.; Platzner Marco. **Partially Reconfigurable Cores for Xilinx Virtex**. In: Field Programmable Logic and Applications (FPL'2002), 2002.
5. XILINX. **Two Flows for Partial Reconfiguration: Core Based or Small Bit Manipulations**. Application Note nb. 290. http://www.xilinx.com/xapp/xapp290.pdf (May 2002).