Tool-Set for Remote and Partial Reconfiguration

Leandro Möller, Daniel Mesquita, Fernando Moraes
{moller, dmesquita, moraes}@inf.pucrs.br

Abstract

Using the features of the Virtex device, a set of tools for remote, partial and dynamic reconfiguration was developed. Remote reconfiguration allows to upgrade or to modify a system from a distant location, by sending a partial (or complete) bitstream via radio or any other media. Partial reconfiguration can be used to quickly modify the system, inserting or removing cores, interconnecting different pre-verified hardware modules, called IP cores, in a single Systems-on-Chip (SOC) design. Even though the tools are target to a specific device, their principles can easily be used to other FPGA families, if they have an internal structure allowing partial reconfiguration

1. Introduction

Hardware cores can be classified as hard, firm and soft. Soft cores are often described using HDLs. A synthesized core, in a netlist format as EDIF, is an example of firm core description. Hard cores are modules ready to be downloaded (FPGA bitstream) or in a mask format (ASIC). The tools presented in this paper are target to FPGA hard cores.

Typical workload of mainstream digital systems is dataflow-based algorithms used in multimedia and telecom applications, requiring a great amount of logic in a reduced area. Digital Reconfigurable Systems (DRS) are an option to implement such circuits in small devices. In DRS hardware can be virtualized, reducing the area requirements, and consequently the overall system cost.

Dynamic reconfiguration is achieved through the following steps:

(i) the system is divided in modules, or simply *cores*;
(ii) a floorplanning tool is employed to fix each core position;
(iii) a set of cores responsible for basic system operation are synthesized and downloaded into the FPGA;
(iv) a partial bitstream is generated for the remaining cores;
(v) remote or locally download the new cores when they are required (in the same manner as virtual memory);
(vi) optionally, partial core reconfiguration to modify its internal parameters (store in memory).

This paper is organized as follows. Section 2 presents the developed tool to merge different cores in a single bitstream. This tools is responsible to achieve the step (iv), i.e., partial bitstream generation. There is no available commercial tool to execute such task, and few academic works [ROX 00]. Section 3 presents a client-server tool allowing to execute the steps (v) and (vi).

2. Core Unifier Tool

In order to connect pre-designed and pre-synthesized cores (*hard cores*), it is necessary to insert or remove them to/from the original bitstream. The developed tool, named *core unifier*, works as follows:

1. A *master bitstream* is opened. Typically this bitstream has an interface to the external world (pads) and *virtual pins* (tri-state buffers inputs/outputs) to connect new cores.
2. One or more bitstreams containing cores to be inserted into the *master bitstream* are opened. The user selects the area corresponding to the core, and all FPGA components inside this area are inserted into the master *bitstream*.
3. The tool creates a partial bitstream, containing the modified area. Partial reconfiguration is then executed, inserting a new core into the FPGA.

The *master bitstream* is necessary due to the FPGA internal architecture. The Virtex device is organized in vertical columns, and the minimal (re)configuration size is one column [XIL01]. So, the *core* to be inserted must be merged to the pre-existing contents of the columns where it will be placed.

This tool was implemented using the Virtex address equations [MES 01]. Figure 1 presents the main window of the *core* unifier tool. This window has a 48*x*32 grid, representing all CLBs of a Virtex 300 device. 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 LUT values can be viewed in an auxiliary window, selecting the CLB with the mouse.

Added to the complexity to implement such tool, the real main problem is to constrain the core position. The floorplanner tool assigns only CLB positions. The routing is not constrained by the floorplanner. So, to obtain a synthesized core restricted to a fixed area, several routing iterations are performed, requiring even

manual user intervention.

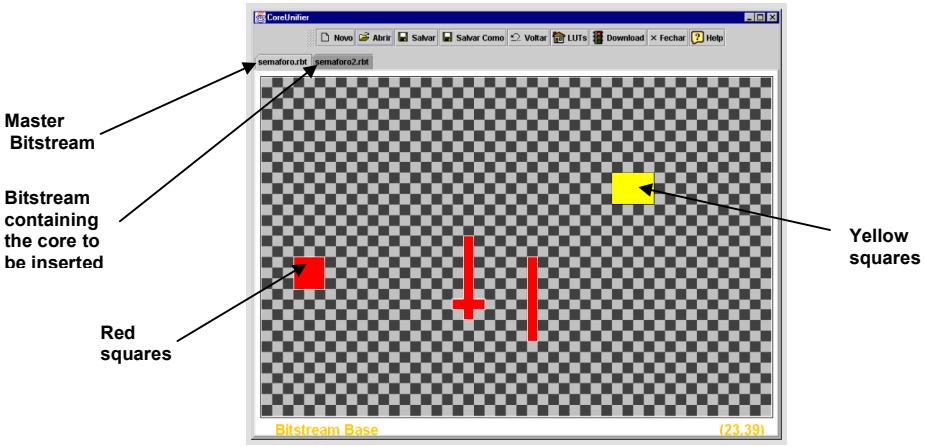

Figure 1 - Core unifier tool main window.

The motivation to implement this tool is to achieve "virtual hardware", in the same manner as "virtual memory". The user can have several hardware modules stored in memory (*hard-cores*), and in function of the execution schedule they are partially stored into the FPGA. This concept is illustrated in Figure 2. The user core is constrained to a fixed area (Figure 2a). The *core unifier* reads data belonging to the core and data belonging to master bitstream in the same column. The resulting bitstream is illustrated in Figure 2c. The partial bitstream is then downloaded into the FPGA.

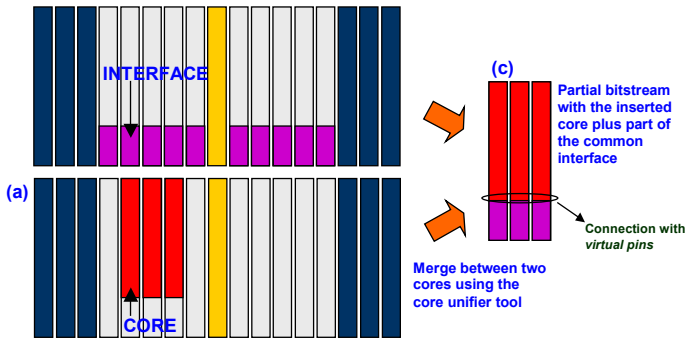

Figure 2 - Bitstream merging operation.

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. Another observed problem is related to routing. Besides the difficulties found to fix routing position, sometimes the core insertion fails.

3. Remote Reconfiguration Tool

Usually, a circuit has a set of parameters defining its function. These parameters are stored into memory blocks, being loaded by an external microcontroller or ROM. The function of this tool is to simplify the design, storing the circuit parameters into FPGA memory blocks (LUTRAM). The advantages of such approach are:

- FPGA can be remotely configured;
- parameters can easily be changed by reconfiguration;
- no external device is needed.

The remote configuration/reconfiguration permits to fix design errors, to change circuit parameters or to upgrade the circuit functions without the customer knowledge. This feature decreases the design maintenance costs. This tool was implemented using the client-server paradigm. The server communicates with the client(s) by sockets, receiving configuration files and downloading it into the FPGA.

Parameters customization is the main goal of the tool. Initially, the circuit is synthesized with a set of LUTRAMs at fixed positions. These LUTRAM store the circuit parameters. Once the bitstream created, the tool described below helps the designer to create an interface giving access to the LUTRAM data. The user downloads its design into the FPGA, and using the interface stores the parameters defining the circuit function

into the FPGA. Note that if the design is already downloaded into the FPGA, partial reconfiguration can be used, changing only CLB columns containing the specified LUTRAM.

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 implement as an *applet*. The *applet* implemented uses JBIT [GUC 00] 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. Figure 3 shows an example of such description. The reference to the *applet* is in line 6 (BITGeneric.class). The parameter "path" (line 7) specifies the bitstream location. The parameters "ip" and "port" (lines 8 and 9) specify the server address. This is necessary to remotely (re)configure the FPGA. The parameter "nrosinais" indicates the number of configurable parameters (line 10). For each parameter the *circuit designer* specifies: (*i*) signal name; (*ii*) format – bin, dec, hex; (*iii*) physical position of the parameters inside the FPGA, defined by row, column, F/G LUT, slice; (v*i*) starting and ending bits in the LUTRAM.

```
1.   <html>
2.     <head> <title>  Parks  </title>        </head>

3.     <body bgcolor=blue>
4.       <center>
5.       <font color=white size=7 face="Arial Black"> Parks  </font>
6.       <APPLET code="BITGeneric.class" width=400 height=300>
7.             <PARAM name="path" value="top_e1.bit">
8.             <PARAM name="ip" value="200.17.93.93">
9.             <PARAM name="port" value="5000">
10.            <PARAM name="nrosinais" value="8">
11.            <PARAM name="l[1]" value="CRCControl  bin, 32, 37, G, 0, 0, 0">
12.            <PARAM name="l[2]" value="Code        bin, 32, 37, G, 0, 1, 1">
13.            <PARAM name="l[3]" value="SlotPattern hex, 32, 37, G, 0, 2, 9">
14.            <PARAM name="l[4]" value="n64         bin, 31, 37, G, 0, 4, 0">
15.            <PARAM name="l[5]" value="StartSlot   bin, 31, 37, G, 0, 9, 5">
16.            <PARAM name="l[6]" value="ServicoIn   bin, 31, 37, G, 0,14,10">
17.            <PARAM name="l[7]" value="InsertServ  bin, 31, 37, G, 0,15,15">
18.            <PARAM name="l[8]" value="DataInsert  hex, 28, 37, G, 0, 0,15">
19.      </APPLET>
20.      </center>
21.    </body>
22.  </html>
```

Figure 3 - Example of HTML description to create the circuit customization interface.

Line 13 of Figure 3 specifies the constraints applied to the SlotPattern signal. It is specified as a hexadecimal value (hex), placed at row 32, column 37, G-LUT, Slice 0, bits 2 to 9 (8-bit value). The *circuit designer* must specify during physical synthesis the same constraints to all configurable memory blocks (LUTRAMs) .

Finally, the ***circuit user*** receives the bitstream and the HTML description. The resulting reconfiguration page, from Figure 3 description, is presented in Figure 4. In the reconfiguration page the values of the signals can be modified, saved and downloaded into the device. Therefore, the *circuit user* can abstract all details concerning the FPGA architecture, and carry out remote and partial reconfiguration.

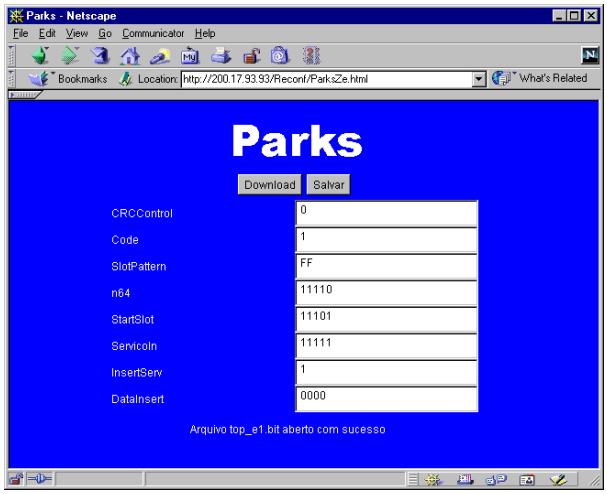

Figure 4 - Circuit customization tool.

A second version of this tool is under implementation, replacing JBITS by the Virtex address equations, adding a set of new features, including partial remote reconfiguration of the bitstreams generated from the first tool.

4. Conclusion and Future Work

The main contribution of this work is a set of tools for remote, partial and dynamic reconfiguration. The main conclusions are:

- Remote reconfiguration works fine, being possible to exploit commercially this feature to update and/or fix hardware modules;
- Parameters reconfiguration can be used to customize a circuit, avoiding extra devices as microcontrollers and ROMs;
- Virtual hardware is feasible in off-the-shelf devices, even if the routing model imposes hard constraints.

The core unifier tool can be integrated with co-design tools. Currently, the hardware modules of a SOC require a programmable device having sufficient area to implement all modules. Another possibility is the generation of several small hardware modules (cores) by the co-design tool, with a scheduler to download these modules into FPGA device. This can be seen as a "*dynamic co-desing*", a new concept not yet explored.

5. References

[GUC 00] Guccione,S.A., Levi,D., Sundararajan,P. **JBits: A Java-based Interface for Reconfigurable Computing**. 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD). Julho 2000.

[MÊS 01] Mesquita,D. **Contribuições para reconfiguração parcial, remota e dinâmica de FPGA**s (in portuguese). Dissertação de Mestrado at PUC, 2001.

[ROX 00] 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.

[XIL 01] **Virtex series configuration architecture user guide**. http://www.xilinx.com/xapp/xapp151.pdf (2001).