Remote and Partial Reconfiguration of FPGAs: tools and trends

Daniel Mesquita[1], Fernando Moraes[2], José Palma[2], Leandro Möller[2], Ney Calazans[2]

[1]Université Montpellier II, LIRMM
161, Rue Ada – 34392 – Montpellier – France
mesquita@lirmm.fr

[2]Pontifícia Universidade Católica do Rio Grande do Sul – FACIN
Avenida Ipiranga, 6681 – 90619-900 – Porto Alegre, RS – Brasil
{moraes, jpalma, moller, calazans}@lirmm.fr

Abstract

This work describes the implementation of digital reconfigurable systems (DRS) using commercial FPGA devices. This paper has three main goals. The first one is to present the trend of DRS, highlighting the problems and solutions of each DRS generation. The second goal is to present in detail the configuration architecture of a commercial FPGA family allowing DRS implementation. The last goal is to present a set of tools for remote and partial reconfiguration developed for this 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.

1. Introduction

Reconfigurable computing has been growing in the past two decades [1]. Research of FPGA-based systems has demonstrated its efficiency over GPP (General Purpose Processor) and software-based systems in several applications.

Many of the systems designated as reconfigurable architectures can only be statically configured. Static reconfiguration means to completely configure the device before system execution. If a new configuration is required, it is necessary to stop system execution and reconfigure the device all over again.

A dynamic reconfigurable device (or system) allows that part of it be modified while the rest of the device (or system) continues to operate. Dynamic reconfigurable systems are quite often coarse-grain architectures, not using off-the-shelf components.

This work addresses partially reconfigurable systems using commercial FPGAs. Currently, only two FPGA vendors support partial and dynamic reconfiguration. One of them, Atmel, produces the FPSLIC (*Field Programmable System Level Integrated Circuit*), a device including a GPP, memory and programmable logic in the same integrated circuit. FPSLIC supports partial and dynamic reconfiguration through context switching [2]. The second one, Xilinx, offers the Virtex family, which also supports partial and dynamic reconfiguration. Reconfiguration is possible because internal configuration elements of this device can be individually addressed [3].

The Virtex family was chosen due to its widespread availability in the market. Using the features of the Virtex device, a set of tools for remote reconfiguration (partial or complete) was developed. Remote reconfiguration allows to upgrade or to modify a system from a distant location, by sending a partial or complete bitstream via Internet, radio or any other transmission medium. Partial reconfiguration can be static, if the system halts, or dynamic, if the rest of the system remains operating.

This paper is organized as follows. The evolution of digital reconfigurable systems and the state-of-the-art on CAD for DRS are presented in Section 2. Section 3 details the Virtex internal configuration architecture, showing how to address internal elements to attain partial reconfiguration. Section 4 presents the proposed tools for partial and remote reconfiguration. Finally, Section 5 provides some final remarks and presents current and future works on the subject.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

2. DRS Trends and CAD for DRS

2.1 DRS Trends

The evolution and trends of DRS is shown in Figure 1. The first generation comprises systems aiming to increase performance over GPPs, using off-the-shelf FPGAs [4]. Typical applications were cryptography, pattern matching and naturally parallel algorithms. Systems such as DECPERLE [5], PRISM [6], SPLASH [7] are examples of this first generation. Modern platforms such as Transmogrifier-2 [8], RPM-2 [9] and SPYDER [10] are also examples of systems belonging to the first generation (Figure 1- botton). These systems are typically composed by a GPP to execute sequential operations and FPGAs to exploit parallelism.

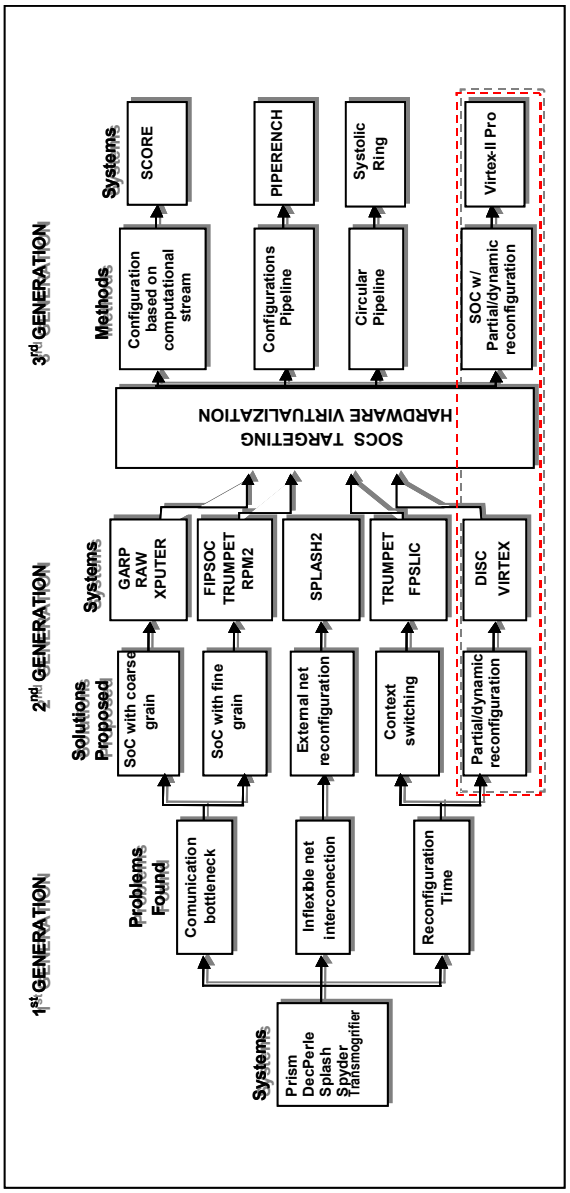


Figure 1 - Trends of reconfigurable architectures

Common problems are observed in these systems: (*i*) the communication bottleneck between FPGA (hardware) and GPP (software), since they are usually connected by an external bus; (*ii*) long time spent to configure devices, demanding an initial configuration to be done before system starts; (*iii*) no support to partial and dynamic reconfiguration (exception to the devices from National Clay and Algotronix); (*iv*) fixed external network.

The evolution in complexity measured by equivalent gates per device made possible to implement complete systems on a single integrated circuit, namely SOC – *System-on-a-Chip*. SOCs merge processors, memory and programmable logic. As a consequence, the bottleneck between GPP and FPGA was minimized.

The granularity concept accounts for the complexity of the minimal processing element of the programmable device. Fine-grain devices typically contain LUTs as processing elements, and coarse-grain devices typically contain ALU or small processors. Examples of fine-grain SOCs are FIPSOC [11] and TRUMPET [12], and coarse-grain SOCs are GARP [13] and RAW [14]. These systems represent examples of the second DRS generation (Figure 1– middle).

Devices and systems allowing dynamic reconfiguration are another characteristic of the second generation. Dynamic reconfiguration can be achieved by context switching with DPGAs [15], or by partially reconfigurable devices, like VIRTEX. DISC and FIREFLY [10] systems also belong to this second generation.

The workload of DRS is moving towards dataflow-based algorithms used in multimedia applications. These applications require complex operators, such as multipliers, and aggressive techniques to increase the output data throughput. Pipelined architectures as PIPERENCH [17] and Systolic Ring [18] are examples of such systems.

DRS are device dependent, i.e. a system synthesized to a given device can only be used in this device. Even for devices of the same family, the system must be resynthesized. Hardware *virtualization* is proposed by SCORE [19] to minimize this problem.

Even the industry believes in this trends. For exemple, recently the Xilinx Inc launch the Virtex II-Pro, a SOC with more than four PowerPC processors embedded with the CoreConnect technology, programmable logic based on Virtex II FPGAs, with memory and multipliers on-chip [26]. This trend is a merge between two features viewed in the second generation: SOCs with fine grain programmable logic and dynamic reconfiguration.

SOCs targeted to data-flow processing (multimedia)

with dynamic reconfiguration and hardware virtualization are features of DRS third generation (Figure 1-top). Our work focuses on the trends assigned on the hatched box. There is a lack of methodologies and CAD tools to take advantage of the hardware provided by industry, and we try to fill this gap

2.2 Related Work on CAD for DRS

Hardware cores can be classified as hard, firm or soft. Soft cores are often described using HDLs. A synthesized core, in a netlist format such as EDIF, is an example of firm core description. Hard cores are cores ready to be downloaded (FPGA bitstream) or in a mask set format (ASICs).

Soft and firm cores are frequently used early in the design flow, described in HDL or EDIF languages. However, if the goal is to allow IP reuse of *completely synthesized cores*, they should be used later in the design flow. In the case of reconfigurable devices, these cores are bitstreams ready to be used to configure the FPGA. Therefore, the function of CAD tools for digital reconfigurable systems is to manipulate hard cores, inserting them into bitstreams of partially reconfigurable FPGAs.

Some tools were written to allow core parameterization, targeting the Xilinx XC6200 FPGA family, now discontinued. Luk et al., in [20], describe cores using a proprietary language and translates these cores automatically into VHDL. FPGA vendors offer similar tools to core parameterization in their design environment. These are examples of core manipulation early in the design flow, not developed for reconfigurable systems.

James-Roxby et al., in [21], describe a tool called *JbitsDiff*, that share some features with one tool proposed here. The basis of this tool is the *Jbits* class library [22]. The user generates a core using a standard design flow, defining its bounding-box with a floorplanning tool. *JbitsDiff* is used to extract the generated core, inserting it into the user bitstream. If the core communicates only with the external world, this tool can be efficiently used. However, if the core should communicate with other cores inside the FPGA, a connection structure must be provided.

The method presented by Dyer, in [23], defines the routing between cores using a structure called *virtual socket*, which defines a border between static and reconfigurable parts. This interface is built from feed-through routed CLBs. The virtual socket is manually placed and routed to guarantee connection between cores.

Recently, Xilinx announced the *Modular Design* tool [24], enabling partial reconfiguration in Virtex devices. The heart of this tool is the *bus macro*, which connects two vertical adjacent cores. Each bus macro provides four bits of inter-core communication. The bus macro is a pre-synthesized bitstream, inserted between two cores, with fixed routing resources. The communication is done through tri-state buffers.

Another approach to partial and dynamic reconfiguration is described in [25]. The PARBIT tool has been developed to transform and restructure bitstreams to implement dynamically loadable hardware modules. To do this, the PARBIT utilizes the original bitstream, a target bitstream and parameters given by user. These parameters include the block coordinates of the logic implemented on a source FPGA, the coordinates of the area for a partially programmed target FPGA and programming options. This tool works over Virtex-E FPGA family.

The presented approaches have interesting advances, but there is some gaps to be filled. The Dyer work extends the JBits, providing the class JBitsCopy to merge cores into FPGAs. But this approach does not address the problem of dynamic reconfiguration.

The main problem of the PARBIT and the method presented in [25] is that they requires a lot of user interactions with the vendor's CAD tool to put the routing resources on the right place, and there is necessary to take care that the routes for de cores do not pass trough the partially reconfigured area, and vice-versa.

The tools JBitsDiff, Modular Design and PARBIT allow the connection between cores inside an FPGA. But a more structured approach to connect cores (using e.g. a standard bus) is required to allow effective implementation of DRS. This work proposes such a structured approach, showing that it is possible to allow IP reuse of synthesized blocks into commercial FPGAs, with dynamic reconfiguration.

3. Virtex Internal Architecture

The main internal components of a Virtex FPGA are CLBs (*Configurable Logic Blocks*), IOBs (*Input Output Blocks*), memory blocks, clock resources and configurable routing. Only the bitstream structure and the equations to access data bits into CLBs are presented in this Section. More details can be found in [3]. It is important to understand the bitstream structure to develop tools aiming core manipulation, i.e., dynamic replacement of cores. The direct access to CLBs information is necessary since CLBs can be configured as memory blocks named LUTRAM, storing circuit parameters.

Virtex devices are organized as bi-dimensional arrays of bits. A single column of bits is named *frame*. One frame corresponds to one atomic reconfiguration unit, i.e. the smallest portion that can be read from (or written to) the FPGA configuration memory. Sets of consecutive frames compose CLB, Block Select RAM, IOB and

Clock columns. Figure 2 illustrates this organization.

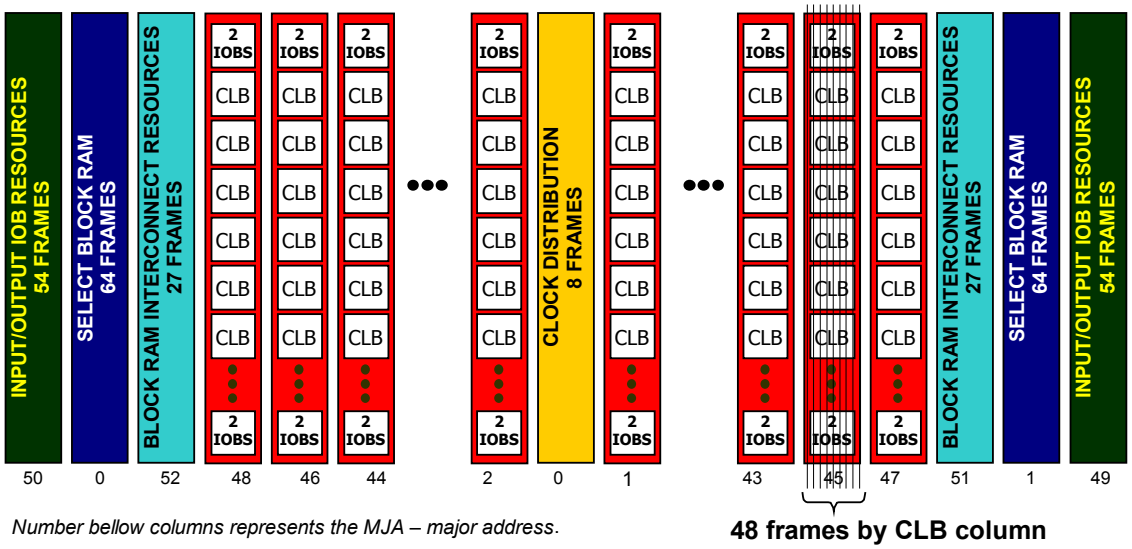


Figure 2 - Virtex architecture (XCV300 device).

A Virtex device can be partially reconfigured since *frames* can be read or written individually. Note that it is not possible to configure a single CLB, since the *frames* belonging to a given CLB are common to all other CLBs in the same column. So, if a modification to a single CLB is required, all *frames* belonging to the same column must be read (operation called *read-back*), and the modification is inserted over the read *frames*. In practice, this feature renders the configuration structure of the Virtex device as an uni-dimensional array of columns.

Each CLB has two slices, named 'S0' and 'S1'. Each slice contains two LUTs, named 'F' and 'G', two flip-flops and carry resources. To address a given LUT, a quadruple is used, for example R8C9.S1.G, which means the LUT is the G LUT at row 8, column 9, slice 1.

3.1 Element Addressing

To partially reconfigure a device it is necessary to address individual elements inside the configuration file, called *bitstream*. The following equations are used to address bits inside LUTs [3]:

$\text{if}\left(\text{CLB}_{\text{col}} \le \frac{Chip_{col}}{2}\right)\ \text{then}\quad \text{MJA} = Chip_{col} - (\text{CLB}_{\text{col}} * 2) + 2\quad \text{else}\quad \text{MJA} = (\text{CLB}_{\text{col}} * 2) - Chip_{col} - 1$
$MNA = lut\_bit + wd - slice * (2*lut\_bit + 17)$
$fm\_bit\_idx = 3 + 18*CLB_{ROW} - FG + RW*32$
$fm\_st\_wd = FL * (8 + (MJA-1)*48 + MNA) + RW*FL$
$fm\_wd = abs(fm\_bit\_idx / 32)$
$fm\_wd\_bit\_idx = 31 + 32*fm\_wd - fm\_bit\_idx$

Where:

- **MJA** - *Major Address*: represents the column address. $Chip_{col}$ is the number of columns available in the device.
- **MNA** - *Minor Address*: identifies in which frame the *lut_bit* is placed. MNA assumes values between 0 and 47. "*wd*" is the number of bits per word (32) and "*slice*" is the slice number.
- **fm_bit_idx** – *frame bit index*: indicates the start position of the CLB being addressed. Constant 18 multiplies CLB_{row} because each CLB requires 18 bits per frame. "FG" is equal to 1 if the desired bit is in a G-Lut, and 0 if it is in an F-Lut. "RW" is equal to 0 when writing data to the FPGA and 1 when reading data from the FPGA (read-back operation).
- **fm_st_wd** – *frame starting word* in the bitstream (file containing 32-bit words). "FL" designates the frame length, i.e., the number of 32-bit words needed to store a complete frame. "8" is the number of clock columns.
- **fm_wd** – indicates, in the bitstream, which word contains the bit we are looking for.
- **fm_wd_bit_idx** – designates the bit inside *fm_word* containing the information we are looking for.

For example, suppose we want to change the 14th bit of an F-LUT, placed at slice 0 of row 1 column 1 (R1C1.S0.F), using the device XCV100, which has $\text{Chip}_{\text{cols}}$=30, FL=14. Applying the above equations, we obtain: *MJA*=30, *MNA*=46, *fm_bit_idx*=21, *fm_st_wd*= 20.244, *fm_wd*=0, *fm_wd_bit_idx*=10. These results mean that the 10th bit (*fm_wd_bit_idx*) of the bitstream word 20.244 (*fm_st_wd* + *f_wd*) is the location of the bit we want to change. Thus, changing this bit and recomputing the bitstream CRC, we are able to reconfigure the FPGA.

4. Tools for Partial and Remote Reconfiguration

This Section presents a pair of tools for partial and remote reconfiguration. The first one was developed

using JBits [22]. The goal of this tool is to help the circuit designer to create an interface to customize parameters of the circuit being designed. The second tool uses the equations defined in the previous Section to manipulate cores, allowing a more powerful form of partial reconfiguration.

4.1 Circuit Customization Tool

Usually, a circuit has a set of parameters defining its behavior, being loaded from an external ROM. The function of this 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. LUTRAM), been 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.

Remote configuration/reconfiguration permits to fix design errors, to change circuit parameters and/or to upgrade the circuit functions without customer knowledge. The implemented tool employs the client-server paradigm to remotely modify the hardware. The server communicates with client(s) through sockets, receiving values, generating new configuration files from these, and downloading them into the FPGA.

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

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, and using the interface may change the parameters at will, using the interface. Note that partial reconfiguration is used, changing only the FPGA columns containing the specified parameter memory blocks.

The ***software developer*** implements a software layer hiding FPGA architecture details. This software layer is implemented as an *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. Figure3a 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 name. The parameters "ip" and "port" (lines 8 and 9) specify the server address and IP service. This is necessary to remotely access the host connected to the FPGA. The parameter *nbsignals* 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. Line 13 of Figure3a 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.

Finally, the ***circuit user*** receives the bitstream and the HTML description. The resulting reconfiguration page is presented in Figure3b. 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.

A second version of this tool is under implementation, replacing Jbits by the Virtex address equations, resulting in an open source code. An important comment is that this tool is addressed to the same goal as the small bit manipulations proposed in [24], but offering a much higher degree of abstraction to its user.

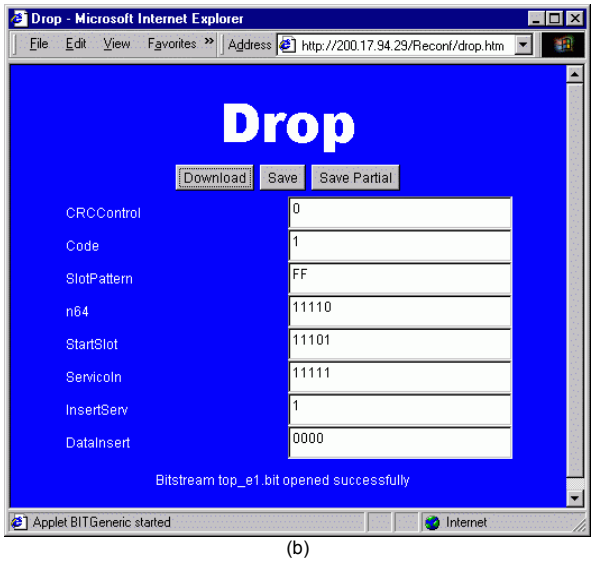
4.2 Core Unifier Tool

Core reconfiguration requires a communication structure between modules inside the FPGA, to allow the practical use of partial reconfiguration. This structure can be viewed as a bus to which application cores are connected.

A fixed core, named *controller*, is initially downloaded into the FPGA. Other cores, named *slave cores*, can be downloaded at run time. The controller is in charge of communication with the external world (I/O pins of the device) and of communication among slave cores. Each slave core communicates with the controller through *virtual pins*

```
1.  <html>
2.  <head> <title> Drop </title>    </head>
3.  <body bgcolor=blue>
4.  <center>
5.  <font color=white size=7 face="Arial Black"> Drop </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="nbsignals" 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>
```

(a)



(b)

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

Some limitations are imposed by the FPGA architecture: (*i*) it is not possible to restrain the CLB usage only to tri-state buffers and routing; (*ii*) it is not possible to define wire positions; (*iii*) only two tri-state buffers are available per CLB. To overcome these limitations, a communication interface with two tri-state buffer layers with common routing wires is implemented, presented in Figure 4. 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.

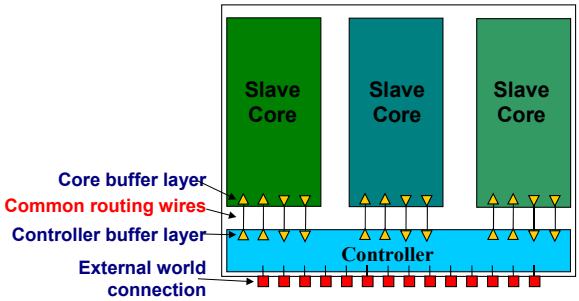


Figure 4 - Communication interface with two tri-state buffer layers and a common routing wire.

The controller contains three cores: *communication bus*, connecting the slave cores; *arbiter*, granting the data line to a given slave core; *master core*, responsible for the communication with the external world. The *communication bus* employs a serial communication protocol to interact with the slave cores. The serial protocol was chosen due to the limited number of tri-state buffers in Virtex FPGAs and to simplify prototyping.

To implement the proposed structure, a CAD tool named *core unifier* was developed. The tool works 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*.
3. The tool creates a partial bitstream, containing the modified area. Partial reconfiguration is then executed, inserting a new core into the FPGA.

This procedure is illustrated in Figure 5.

This tool was implemented using the Virtex address equations (Section 3.1). The approach gives to the user most features found in *JBitsDiff*. However, it provides a structured form to interconnect cores, together with the possibility to dynamically replace cores in the FPGA.

Figure 6 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 and it is different for distinct 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 LUT values can be viewed in an auxiliary window, selecting the CLB with the mouse.

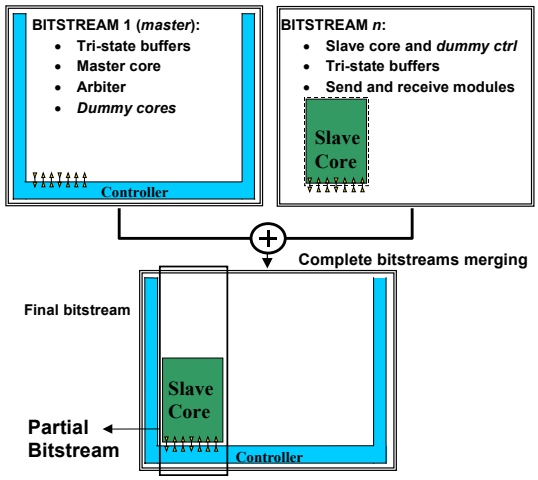


Figure 5 - Bitstream merging procedure.

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. The user may have several hard cores stored in memory. As a function of

some execution scheduling these may be partially downloaded into the FPGA.

Three main problems exist in this approach, all related to the current state of commercial FPGA devices and associated CAD tools: (*i*) 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 [23] and in [24] to verify that FPGA vendor tools must evolve to better support partial and dynamic reconfiguration.

At present, the tool creates complete and partial bitstreams. Complete and partial bitstream download with core insertion was achieved successfully. A subset of *JBits* classes, collectively known as *JRoute* is being investigated to have access to Virtex FPGA routing architecture manipulations.

5. Conclusions and Future Work

The first contribution of this work is the analysis of DRS trends. The goals, solutions and remaining problems of each DRS generation were highlighted. Also, this study identified the future directions of configurable architectures.

The second contribution is the development of tools for remote, partial and dynamic reconfiguration. Multiple advantages are offered by this tool-set. First, remote reconfiguration is enabled. It is possible to exploit this feature to update and/or fix hardware cores in the field. Second, parameter reconfiguration can be used to customize a circuit, avoiding extra devices as external microcontrollers and ROMs, and saving internal control logic in the FPGA. Third, virtual hardware is feasible in off-the-shelf FPGA devices, even if the routing model imposes hard constraints.

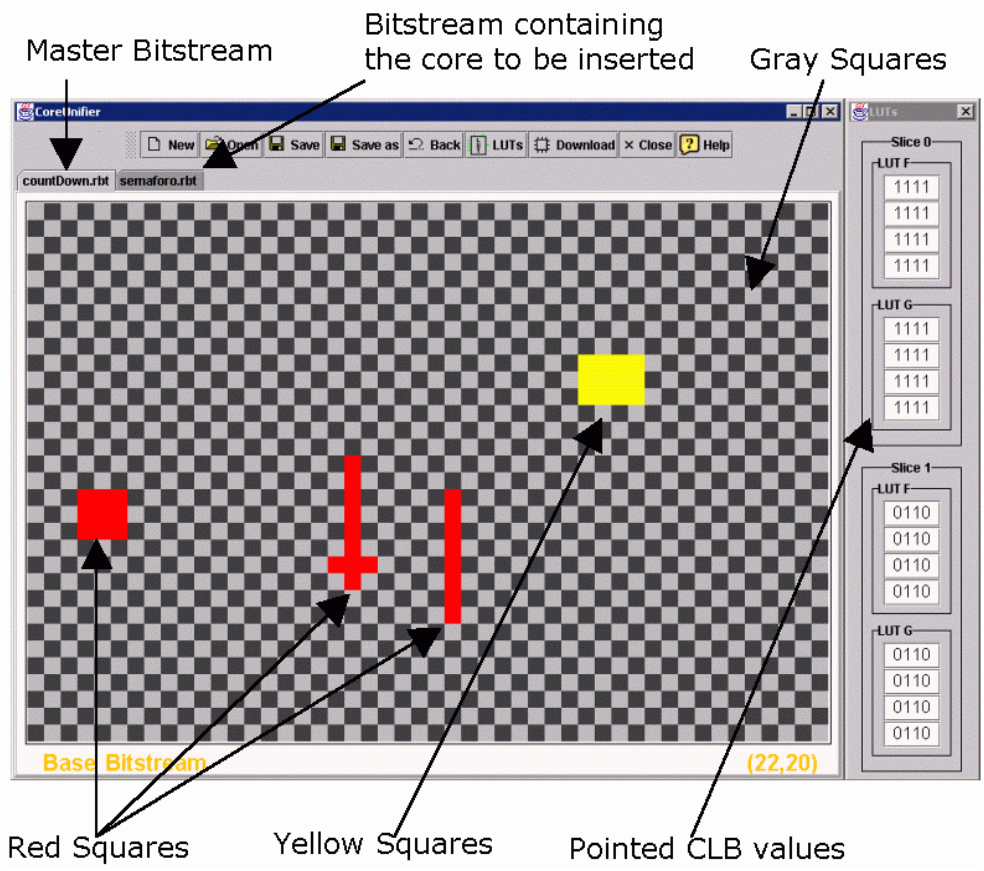


Figure 6 – Core unifier tool main window.

As suggestions for future work, the following can enumerated: (*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.

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

6. References

1 Hartenstein, R. **A decade of reconfigurable computing: a visionary retrospective**. In: Design, Automation and Test in Europe, pp. 642 –649, 2001.

2 ATMEL. **Field Programmable System Level Integrated Circuits (FPSLIC)** (Aug. 2002). http://www.atmel.com/atmel/products/prod39.htm

3 XILINX. **Virtex series configuration architecture user guide**. Application Note nb. 151, http://www.xilinx.com/xapp/xapp151.pdf (March 2000)

4 Page, I. **Reconfigurable processor architectures**. Elsevier Microprocessors & Microsystems, v.20, p.185–196, 1996.

5 Vuillemin, J.E.; et al. **Programmable active memories: reconfigurable systems come of age**. IEEE Transactions on VLSI Systems, vol 4(1), pp. 56-69, 1996.

6 Athanas, P.; Silverman, H.F. **Processor reconfiguration through instruction-set metamorphosis**. Computer, vol 26(3), pp. 11-18, 1993.

7 Gokhale, M. **SPLASH, A Reconfigurable Linear Logic Array**. In: International Conference on Parallel Processing. pp. 219-314, 1990.

8 Lewis, D.M.; Galloway, D.R.; Van Ierssel, M.; Rose, J.; Chow, P. **The Transmogrifier-2: a 1 million gate rapid-prototyping system**. IEEE Transactions on VLSI Systems, vol 6(2), pp 188-198, 1998.

9 Dubois, M.; Jaeheon Jeong; Yong Ho Song; Moga, A. **Rapid hardware prototyping on RPM-2**. IEEE Design & Test of Computers, vol 15(3), pp. 112 -118, 1998.

10 Sanchez, E.; et al. **Static and dynamic configurable systems**. IEEE Transactions on Computers, vol 48(6), pp. 556-564, 1999.

11 SIDSA. **FIPSOC mixed signal system-on-chip**. http://www.sidsa.com/FIPSOC/fipsoc.html (Aug. 2002).

12 Perissakis, S.; Joo, Y.; Ahn, J.; Dellon, A.; Wawraynek, J. **Embedded DRAM for a reconfigurable array**. In: Symposium on VLSI Circuits, pp. 145-148, 1999.

13 Callahan, T.J.; Hauser, J.R.; Wawrzynek, J. **The Garp architecture and C compiler**. Computer, vol. 33(4), pp. 62-69, 2000.

14 Waingold, E.; et al. **Baring it all to software: Raw machines**. Computer, vol. 30(9), pp. 86-93, 1997.

15 DeHon, **A. DPGA-coupled microprocessors: commodity ICs for the early 21st Centur**y. In: FPGAs for Custom Computing Machines, pp. 31 –39, 1994.

16 Wirthlin, M.; Hutchings, B. **DISC: The dynamic instruction set computer**. In: Proceedings of the SPIE - FPGA for Fast Board Development and Reconfigurable Computing, pp. 92-103, 1995.

17 Goldstein, S.C.; et al. **PipeRench: a reconfigurable architecture and compiler**. Computer, vol 33(4), pp. 70-77, 2000.

18 Sassatelli, G.; et al. **Highly scalable dynamically reconfigurable systolic ring-architecture for DSP applications**. In: DATE, pp. 553-558, 2002.

19 Caspi, E.; et al. **Stream Computations Organized for Reconfigurable Execution (SCORE): Introduction and Tutorial**. In: Field Programmable Logic and Applications (FPL'2000), Villach, Austria.. pp 605-614. 2000

20 Luk, W.; Mckeever, S. **Pebble: a language for parameterized and reconfigurable hardware design**. In: Field Programmable Logic and Applications (FPL'1998), pp 9-18, 1998.

21 James-Roxby, P.; Guccione, S.A. **Automated extraction of run-time parameterisable cores from programmable device configurations**. In: Field-Programmable Custom Computing Machines, pp. 153-161, 2000.

22 XILINX. **The Jbits 2.8 SDK for Virtex**. ftp://customer:xilinx@ftp.xilinx.com/download/JBits2_8.exe (Sept. 2001).

23 Dyer, M.; Plessl, C.; Platzner Marco. **Partially Reconfigurable Cores for Xilinx Virtex**. In: Field Programmable Logic and Applications (FPL'2002), pp 292-301, 2002.

24 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).

25 Horta, E. L.; Locwood, J. W.; Kofuji, S. T. **Using PARBIT to implement Partial Run-time Reconfigurable Systems**. In: Field Programmable Logic and Applications (FPL'2002), pp 182-191, 2002.

26 XILINX. **Virtex II-Pro Platform FPGA Complete DataSheet**. Application Note nb. 083 http://direct.xilinx.com/bvdocs/publications/ds083.pdf. (January 2003).