PaDReH - A Framework for the Design and Implementation of Dynamically and Partially Reconfigurable Systems

Ewerson Carvalho ecarvalho@inf.pucrs.br
Ney Calazans ecarvalho@inf.pucrs.br
Eduardo Brião briao@inf.pucrs.br
Fernando Moraes moraes@inf.pucrs.br
Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS)
Av. Ipiranga, 6681 - Prédio 30 / Bloco 4 - 90619-900 - Porto Alegre – RS – BRASIL

ABSTRACT
Dynamically and Partially Reconfigurable Systems (DRSs) are those where any portion of the hardware behavior can be altered at application execution time. These systems have the potential to provide hardware with flexibility similar to that of software, while leading to better performance and smaller system size. However, the widespread acceptance of DRSs depends on adequate support to design and implement them. This work proposes a framework for DRS design and implementation named PADReH. The approach is compared to other propositions available in the literature. The first steps of the framework implementation are described, involving methods and tools to control the hardware reconfiguration process and the generation of partial bitstreams. The main contribution of the work is to provide means to systematically reduce the lack of support currently hampering the adoption of DRSs as a mainstream technology.

Categories and Subject Descriptors
B.7.1 [**Integrated Circuits**]: Types and Design Styles – *advanced architectures, algorithms implemented in hardware, VLSI (very large scale integration).*

General Terms
Design, Experimentation, Theory.

Keywords
Dynamically and partially reconfigurable systems; reconfiguration control, partial bitstream generation, run-time reconfiguration.

1. INTRODUCTION
It is possible to notice an increasing interest on reconfigurable computing [1]. The potential flexibility provided by reconfigurable hardware has the potential to increase the lifetime of products. Similar to software systems that constantly receive updates, hardware implemented with reconfigurable devices can put this strategy to good use to preserve product utility for longer time. In addition, the time available to execute design flow continually decreases because of market pressures. The use of reconfigurable technology, coupled to the massive reuse of intellectual property can reduce System-on-Chip (SoC) design time [2], decreasing the time-to-market of technological products.

One attractive feature of using reconfigurable computing is the possibility to implement a whole system in less silicon than its nominal minimal requirement, developing the concept of virtual hardware [3]. The use of DRS design techniques has potential to save resources while reducing the system area overhead. This happens because they allow that parts of the systems not needed in some time interval be removed from the hardware to make room for another part of the system, required at that same interval. On the other hand, potential drawbacks of DRSs are the performance penalty induced by the often long reconfiguration times and the area overhead to implement the hardware responsible for controlling the reconfiguration process.

The deployment of DRSs however requires support that is not yet available [1]. This support comprises tools to enable the design and verification of DRSs and adequate infrastructure for enabling the design and runtime control of these systems. The above items must be integrated through the use of adequate methods that take into account the specificities of DRSs. Such methods are not themselves available in a widespread way. The main objective of the present work is to propose and describe PADReH, a framework to provide support for the hardware implementation of DRSs.

The rest of this paper is organized as follows. Section 2 presents a survey of previous DRS frameworks propositions. Section 3 describes the main features of the PADReH framework, while Sections 4 and 5 discuss the implementation of parts of the framework. Section 6 discusses DRS applications and presents a case study employed in the validation of the proposed tools and infrastructure. Next, Section 7 describes practical results of using the framework for implementing the case study of Section 6, while Section 8 presents some conclusions and directions for future work.

2. RELATED WORKS
Several approaches have been proposed to organize the design, implementation and maintenance of reconfigurable systems. This Section reviews several relevant propositions in this theme.

The Brass project proposes SCORE (Stream Computations for Reconfigurable Execution) [4], a computation model based on the organization of reconfigurable systems around the virtualization of three main hardware concepts: paged reconfigurable hardware, page communication through the use of streams, and storage.

The Janus framework [5] targets the development of reconfigurable systems composed by stages implemented either in

SBCCI'04, September 7-11, 2004, Pernambuco, Brazil.

software and hardware and based on JHDL, a set of classes built upon Java.

Edwards and Green [6] present an integrated run-time environment to support partially and dynamically reconfigurable systems based on the discontinued Xilinx XC6200 FPGA family. They propose FSS (FPGA Support System), a system to support loading and unloading of hardware tasks on FPGAs. The software applications are developed in C++ and hardware tasks are described in VHDL.

The Model-Integrated Development Environment for Adaptive Computing (MIDE) project [7] has as goal to develop high-level system design tools for implementing dynamically reconfigurable systems using adaptive computing technology. MIDE provides tools that allow designers to construct and exercise graphical models of the systems, used to create executable computational structures (software and hardware), and a runtime environment. The system is aimed at embedded systems of weapons like missile guiding systems and uses DSP processors coupled to Virtex Xilinx FPGAs.

The design environment CHAMPION [8] provides automatic retiming to match paths through the dataflow and performs multi-FPGA design partitioning, without knowledge of the specific architecture implementation.

Einsenring et al. [9] proposes a run-time reconfiguration of FPGA computing resources, where system behavior and architecture are represented as a *problem graph*, and an *architecture graph*, respectively. Tool support is provided to map the nodes in the problem graph to nodes in the architecture graph, assign problem graph nodes to FPGA configurations, and schedule the execution order of problem graph objects and configurations.

The Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems (SPARCS) [10] takes a design specification at the behavioral level in the form of a task graph, divides and schedules the tasks on the target architecture and maps the tasks to individual FPGAs in multi-FPGA systems.

3. PADReH FRAMEWORK FOR DRS

PADReH is a framework to design and implement DRSs. This framework is intended to enable obtaining advantages from the use of dynamic and partial reconfigurable hardware technology. The general characteristics of this framework are as follows.

First, PADReH is initially addressed to deal only with the DRS hardware design flow only. This is the same decision followed by SPARCS, Einsenring et al. approach, and CHAMPION, but distinct from the choice in Janus and MIDE. The reasoning behind the decision is that most of the complexity of DRSs lies on the hardware side, and it is necessary to extensively experiment with hardware concepts before delving into software implications of DRS.

Second, PADReH is targeted to support partial and dynamic reconfiguration of single FPGAs. This is justified by the fact that current FPGAs are already capable of embedding entire complex systems, and that FPGA advances are fast enough for this to stay valid in the long term. This choice is the same for the systems MIDE and the one proposed by Edwards et al., and addressing it is planned by Einsenring et al. The opposite approach, suggested by CHAMPION, SPARCS and currently supported by Einsenring et al., DRSs implemented in multiple FPGAs, is useful for low-end applications. This occurs once multiple FPGAs typically present reduced performance and cost[1], while single FPGA DRSs are adequate for high-end applications.

Third, regarding the underlying implementation technologies, PADReH advocates the use of higher level abstractions for design capture and validation through the use of languages like SystemC. Also, initial development of back-end DRS support employs Xilinx Virtex families FPGAs devices (Virtex, Virtex2 and Virtex2-Pro). The justification for these choices is the emerging of SystemC as a de facto standard for addressing hardware modeling at abstraction levels above RTL, and the fact that Virtex FPGA families are the only commercial devices to support System-on-Chip integration capability. These choices can be compared to the use of Khoros by the CHAMPION system, JAVA by Janus, and C++/VHDL in the approach of Edwards et al. On the support device side, most reviewed systems claim to use or to be considering the use of Virtex families.

3.1 The PADReH System Structure

The PADReH system is composed by three module sets, as depicted in Figure 1.

The first module set is named *Design Capture and Functional Validation*. It is responsible for the description and validation of DRSs at high abstraction levels and translation from these to the *Register Transfer Level* (RTL) of abstraction.

Next, the *Partitioning and Scheduling* module set is responsible for the generation of files that describe the DRS behavior. These files are usually represented in a hardware description language (HDL). The same files are transmitted to the module set *Physical Synthesis and Reconfiguration Infrastructure*. This module set is responsible for the generation of configuration files implemented as total and partial bitstreams, along with the spatial and temporal system partitioning, as defined in the second module set. It is also responsible for inserting the parameterized configuration controller module in the system, according to the specific DRS characteristics. The generation of the physical interconnection implementation (e.g. bus or network-on-chip) among cores of the DRS is also performed in this module set.

The shaded area in Figure 1 represents the parts of the framework that are currently supported through the use of automated methods and tools. Other parts of PADReH have been specified and are currently conducted by combinations of manual application of the method and/or with the help of commercial tools. The next two Sections describe respectively the methods developed for bitstreams generation and for runtime reconfiguration control of DRSs.

[1] For example, the cost of ten million-gate FPGAs is typically much smaller than the cost of a single 10-million-gate FPGA.

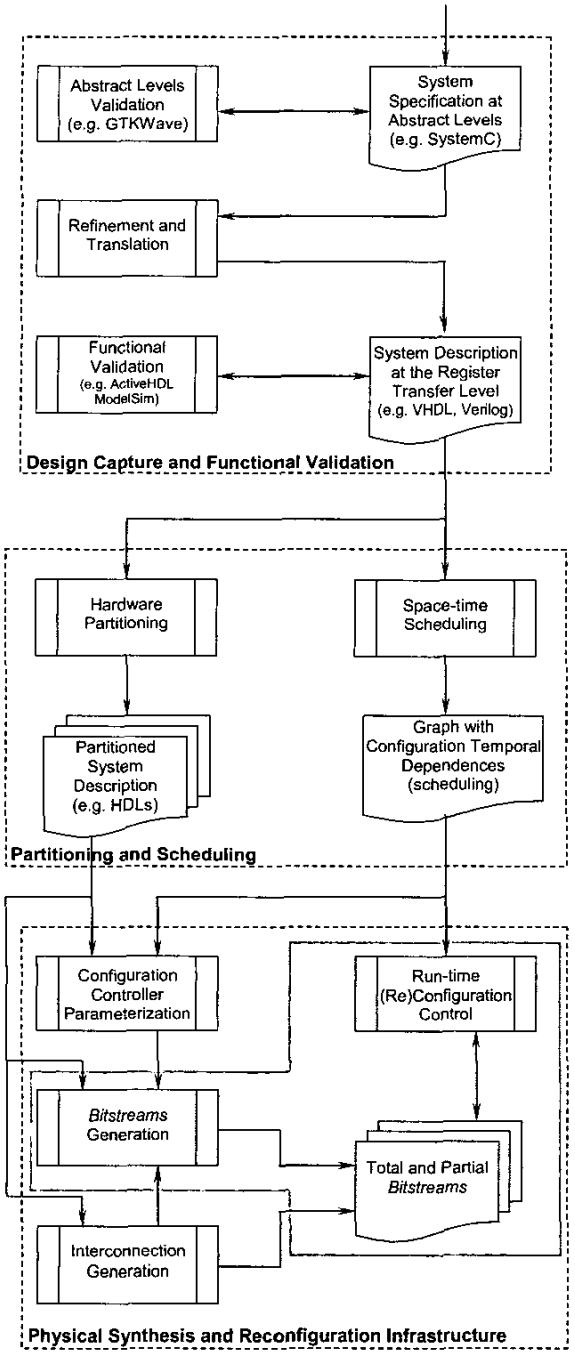

Figure 1 - PADReH framework design, verification and implementation flow for DRS.

4. BITSTREAMS GENERATION

In most FPGA devices, the entire chip is configured using one complete set of configuration data, called bitstream. Bitstreams contain the information required to set the state of FPGA configurable resources (e.g. switches and lookup tables) thus implementing the designed circuit.

To partially reconfigure an FPGA, it is necessary to create *partial* bitstreams, and to have available FPGA architectures that explicitly support the partial loading of configuration data contained in these files. Partial bitstreams take less time to be loaded in the FPGA. Another potential advantage is that, depending on the FPGA architecture, parts of the device not suffering the reconfiguration process may continue to operate normally during the partial reconfiguration.

4.1 Bitstream Manipulation Approaches

The generation of partial bitstreams is a crucial task for the kind of DRS addressed in this work. The production of partial bitstreams with current FPGA tools is basically a manual, complex and error-prone process. Some tools and techniques to automate the generation of partial bitstreams can already be found on the research literature. Examples are JbitsDiff [11], Jbits [12], PARBIT [13] and JPG [14]. Most of these tools and techniques allow *difference based* manipulations where just very small, localized portions of the FPGA are changed at any given moment [15] (in this sense, PARBIT is an exception). Through of using these techniques, it is difficult or even unfeasible to implement complex dynamic IP cores (e.g. image filters, processors, etc). The more complex problem of *module based* partial reconfiguration is much less exploited in the literature. Module based reconfiguration allows substituting arbitrarily sized regions of the FPGA and imply the need to precisely define an interface between reconfigurable areas and the rest of the device, formed by either other reconfigurable areas or fixed areas. The PADReH framework addresses module based partial reconfiguration.

4.2 IP Cores Communication in DRS

Some reconfigurable IP cores communication have been proposed. In the context of initial development of the PADReH framework, Palma et al. suggested a method to generate the interconnection among partial bitstreams using a bus-based structure [16]. The method is partially automated, but suffered from limitations due to difficulty to more precisely control the routing in Xilinx FPGA designs. Another technique for interconnecting dynamically replaceable cores in FPGAs has been proposed by Xilinx itself [15]. This technique was adopted and adapted to the PADReH framework needs.

4.3 PADReH Bitstreams Generation

The technique proposed in [15] is based on the Xilinx Modular Design flow. This design flow establishes a set of steps to generate partial bitstreams. Besides, it allows implementing dynamic IP cores (partial bitstreams) at a more abstract level than other bitstream generation tools. The designer may start from HDL descriptions of reconfigurable modules, elaborate some flooplanning constraints and from this set of data automate the generation of partial bitstreams. In this technique, for IP cores that communicate with each other, a special Xilinx library component called *bus macro*. This component enables the interconnection of signals across the boundary of reconfigurable and fixed areas or between a pair of reconfigurable areas. The bus macro is composed by eight tristate buffers that create a bidirectional 4-wire interface between modules. Figure 2 shows the structure of a bus macro. Note that each side of the bus macro may control the flow of data in the interface. A control protocol must be agreed upon by both sides to avoid the possibility of bus conflicts.

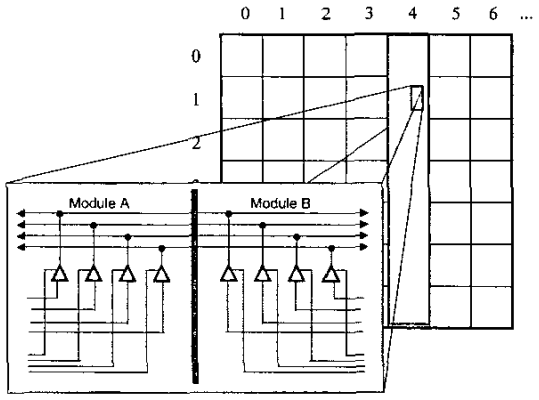

Figure 2 - Bus macro with typical placement in the FPGA.

For each DRS implementation, the FPGA must be initialized with a total bitstream, followed by successive reconfigurations with partial bitstreams. The next Section describes how the PADReH framework treats the configuration/reconfiguration process in more detail.

Although the Modular Design flow provides the necessary means to enable module based reconfiguration, the flow is not as automated as it could be. This makes the whole process error-prone. Four major issues that could be enhanced in the flow were identified:

- Several specific logic synthesis scripts must be produced in the flow. Xilinx provides only hints on how these must be written and formatted;
- One physical synthesis complex script must be generated for each partial and total bitstream manipulated during the flow. Again, these are not fully specified by the FPGA vendor;
- Bus macros must be explicitly inserted by the DRS designer in HDL files;
- A set of constraints must be manually generated and inserted in a constraints file, in order to correctly generate partial and total bitstreams.

To reduce the risk of errors during the application of the Modular Design flow, a tool named MDLauncher (i.e. Modular Design Launcher) has been proposed and developed [17]. MDLauncher reduces design time and decreases the amount of errors during design and implementation of DRSs. The user creates the top module and the partial modules and input these using the MDLauncher GUI. The tool will automatically perform all steps of Modular Design flow, solving two of the issues mentioned above through scripts template personalization. MDLauncher is integrated with the Leonardo Spectrum tool, used for logic synthesis and Xilinx ISE tool flow for physical synthesis.

5. CONFIGURATION CONTROL

A *configuration controller* commands which reconfigurable IP core(s) must be inserted on the reconfigurable device at any moment, and which must be removed. It executes tasks similar to those of a loader of an operating system. This module is responsible for loading configurations to execute on the reconfigurable hardware, according to a defined task scheduling.

The PADReH framework supports the control of the configuration process by using a fixed hardware module that automates the management of the partial bitstreams described in the previous Section. This hardware module, named RSCM, must be present in every DRS implemented with the framework.

5.1 The RSCM Controller

The *Reconfigurable System Configuration Manager* (RSCM) is really a model to implement configuration controllers [18]. Its general structure is detailed in Figure 3.

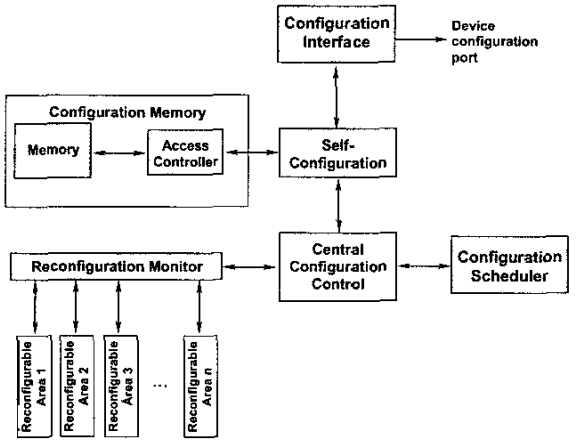

Figure 3 - RSCM model and its usage in the implementation of DRSs. The gray modules are components of the RSCM controller. The other modules in picture are part of the dynamic reconfigurable system. The Memory is only module outside of the FPGA.

The *Configuration Memory* (CM) stores all partial bitstreams used at runtime by the system. Considering the large amount of space to store bitstreams, and the scarcity of storage in current FPGAs, the *Configuration Memory* is a partially implemented outside the reconfigurable device.

The *Self-Configuration* (SC) module controls the configuration process. It has an interface with the CM module, to send control signals requesting configuration data. The *Configuration Interface* (CI) module formats configuration data to the FPGA. The *Central Configuration Control* (CCC) interface receives requests to start the configuration process, and provides results of the process in the form of status signals. The SC module may contain logic to control the relocation of partial bitstreams. CI is responsible for receiving configuration data from SC and sending it to the FPGA configuration port.

The Reconfiguration Monitor (RM) detects situations where reconfigurations need to be performed, the so-called *reconfiguration events*, and notifies CCC, which acts appropriately.

The *Central Configuration Control* (CCC) manages all control flow between other modules of the RSCM system. It applies the configuration scheduling stored on the *Configuration Scheduler* (CS) module. CCC receives requests from RM and requests services to the CS and SC modules.

The *Configuration Scheduler* (CS) module is responsible to determine which bitstream is the next to be configured. This

module receives service requests from the CCC. It stores a data structure with information about configurations dependence, called *Table of Dependencies and Descriptors* (TDD).

In this context of the PADReH framework, the RSCM model is implemented in hardware, but since the model is generic, it could as well be implemented in software or as mixed hardware/software versions. Since the implemented controller is part of the hardware and lies inside the reconfigurable device containing the rest of the system, the device is capable of performing its own reconfiguration without resource to external controlling devices. More implementation details may be obtained in [18].

6. APPLICATIONS IMPLEMENTED ON DRSS

Several application fields may benefit from the use of DRSs. The authors are currently investigating the use of these systems in network intrusion detection applications. Network intrusion detection requires that a huge amount of data processing be performed using a set of rules that are changed as a function of the network instantaneous traffic pattern. Rapid switching of rules and very high throughput prevent the use of software-only solutions. This is an ongoing work.

Possibly, the most intuitive form of DRS is one where the instruction set of a processor is dynamically augmented through the use of reconfigurable hardware. Such components are generically known as reconfigurable processors. As a proof of concept for the PADReH framework, the rest of this Section presents a system named R8NR, implemented by the authors and composed by a processor with N attached reconfigurable coprocessors.

6.1 R8NR – A Reconfigurable Processor

The R8R processor is based on the R8 processor, a 16-bit load-store 40-instruction RISC-like processor [19]. The original R8 processor was transformed into the R8R processor by the addition of five new instructions intended to give support to the use of partially reconfigurable coprocessors. The R8R processor was wrapped to provide communication with the local memory, the system bus, the RSCM and the reconfigurable regions.

The coprocessors are configured on demand, under control of the software that executes on the R8R processor. During the execution of the system, the R8R selects, at each moment, one specific coprocessor with which it operates. This selection is sent to the RSCM controller that according to the allocation state of reconfigurable areas verifies if the coprocessor is already present in the hardware, reconfiguring some unselected area, if needed. After this, the RSCM notifies the processor that the selected coprocessor is ready. From now on, the software can request coprocessor services. The signal exchange protocol that implements the R8NR inner working is explained next.

For this case study, three coprocessors were implemented. The first, *SQRT*, computes the square root. *MULTI* coprocessor executes a multiplication and *DIV* coprocessor executes a division.

7. CASE STUDY INITIAL QUANTITATIVE RESULTS

The system described in Section 6 has been completely prototyped and is operational in two versions, with one (R81R) and two reconfigurable regions (R82R), respectively. A V2MB1000 prototyping platform from Insight-Memec was used. This platform contains a million-gate XC2V1000 Xilinx FPGA, memory and I/O resources.

One of the gains of using the approach proposed here is that using a configuration controller internal to the FPGA device leads to a better performance in configuration time. For example, using an implementation of the RSCM running at 24MHz, the time to reconfigure a bitstream with *Ncw* words is given by: $Tr = 884ns + (Ncw \times 748ns)$.

This leads to a total time to reconfigure a million-gate FPGA in 95,43 ms and 10 ms to reconfigure each coprocessor mentioned in the previous Section.

7.1 Coprocessor Execution Time

To compare execution times, software implementations of each coprocessor were used. The results are illustrated in the graph of Figure 4. For the multiplication coprocessor, the execution of more than 750 consecutive operations will execute faster in hardware than in software, considering the reconfiguration time overhead. For division and square root the respective break-even points occur for 260 and 200 operations. The determination of this break-even point is important to establish when using DRSs becomes advantageous. Example applications where these results can be applied are digital filters where a great number of multiply-accumulate operations execute over a data set.

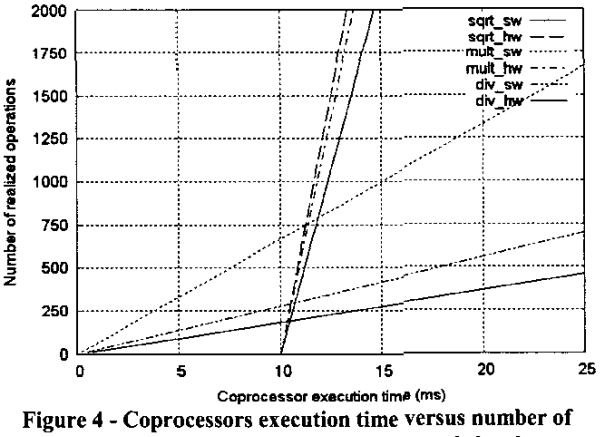

Figure 4 - Coprocessors execution time versus number of performed operations. The _hw suffix regards hardware implementations, while _sw regards software implementations. Each partial bitstream is configured in approximately 10ms by RSCM.

7.2 Area Consumption

The preliminary version of the PADReH framework was used to design and prototype the R81R and the R82R case studies. The more complex system, R82R occupied 14.89% of the million-gate XC2V1000 FPGA or less than 150,000 equivalent gates. The

RSCM accounts for 7,5% of these gates only, or 1,13% of the FPGA resources.

8. CONCLUSIONS AND FUTURE WORK

This work proposed a framework to support the design, validation and implementation of Dynamically and Partially Reconfigurable Systems, DRSs. PADReH employs a set of state-of-the-art technologies to enable the implementation of DRSs. These include the use of modern enabling VLSI target devices, higher level recently proposed abstraction levels for design capture, and a set of methods and tools to automate the process.

The framework is currently under implementation using a bottom-up approach. The infrastructure to implement DRSs is already operational as demonstrated by the discussion of the case studies design, implementation and prototyping.

Future works include the refinement of the implementation infrastructure, comprising completing and improving the Physical Synthesis and Reconfiguration Infrastructure module set of Figure 1, and addressing the upper abstraction levels of the framework.

One immediate work is to enhance the bitstream generation process, by automating the issues identified in Section and not yet addressed. In the case of the reconfiguration control process, it is necessary to improve RSCM performance and provide support in it for the relocation of bitstreams.

9. ACKNOWLEDGMENTS

Work partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, under scholarship grants 130900/2002-8 and 307655/2003-2.

10. REFERENCES

[1] Zhang, X., and Ng, K. W. A review of high-level synthesis for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems*, v. 24, p.199-211. 2000.

[2] Bergamaschi R., and Lee, W. Designing system-on-chip using cores. *In DAC'00*, pp. 420-425. USA. 2000.

[3] DeHon, A. Comparing Computing Machines. *In Proc. SPIE Configurable Computing: Technology and Applications*. v. 3526, pp. 124-133, 1998.

[4] Caspi, E., DeHon, A., and Wawrzynek, J. A streaming multithreaded model, *In: Third Workshop on Media and Stream Processors*, 2001.

[5] Lehn, D., Hudson, R., and Athanas, P. Framework for architecture-independent run-time reconfigurable applications, *In: SPIE Proceedings*, November, 2000.

[6] Edwards, M., and Green, P. Run-time support for dynamically reconfigurable computing systems. *Journal of Systems Architecture*, v. 49, pp. 267-281. 2003

[7] Bapty, T., Neema, S., Scott, J., Sztipanovits, J., and Asaad, S. Model-integrated tools for the design of dynamically reconfigurable systems, Technical Report #ISIS-99-01, ISIS, Vanderbilt University, 2000.

[8] Natarajan, S., Levine, B., Tan, C., Newport, D., and Bouldin, D. Automatic Mapping of Khoros-based Applications to Adaptive Computing Systems. *In: MAPLD'99*, pp. 101-107, 1999.

[9] Eisenring, M., and Platzner, M. A framework for run-time reconfigurable systems, *Journal of Supercomputing* v. 21, pp. 145-159, 2002.

[10] Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul, M., and Vemuri, R. An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. *In: RAW'98*, 1998.

[11] James-Roxby, P.; Guccione, S. Automated extraction of run-time parameterisable cores from programmable device configurations. *In: FCCM'00*. pp. 153-161, 2000.

[12] Xilinx, Inc. The JBits 3.0 SDK for Virtex-II. 2003. Available at http://www.xilinx.com/labs/downloads/ jbits/index.htm

[13] Horta, E., Lockwood, J., and Kofuji, S. Using PARBIT to implement Partial Run-time Reconfigurable Systems. *In: FPL'02*, pp 182-191, 2002.

[14] Raghavan, A., and Sutton, P. JPG - a partial bitstream generation tool to support partial reconfiguration in Virtex FPGAs. *In: IPDPS'02*, pp. 155-160, 2002.

[15] Xilinx, Inc. Two Flows for Partial Reconfiguration: Module Based or Difference Based. *Xilinx Application Note* XAPP290, V1.1. 2003.

[16] Palma, J., Mello, A., Möller, L., Moraes, F. and Calazans, N. Core Communication Interface for FPGAs. *In SBCCI'02*, pp. 183-188, 2002.

[17] Brião, E. Partial and Dynamic Reconfiguration for Intellectual Property Cores, *MSc Dissertation*, PUCRS-PPGCC–FACIN, Porto Alegre, Brazil, 2004. (In Portuguese)

[18] Carvalho, E. RSCM: Configuration Control for Reconfigurable Hardware Systems, *MSc Dissertation*, PUCRS-PPGCC–FACIN, Porto Alegre, Brazil, 2004. (In Portuguese)

[19] Moraes, F., and Calazans, N. R8 Processor Architecture and Organization Specification and Design Guidelines. 2003. Available at http://www.inf.pucrs.br/~gaph/Projects/R8 /public/R8_arq_spec_eng.pdf