Reconfiguration Control for Dynamically Reconfigurable Systems

Ewerson Carvalho, Ney Calazans, Fernando Moraes and Daniel Mesquita

Abstract— **Dynamically Reconfigurable Systems (DRS), those where the hardware can be changed at runtime, have the potential to enhance hardware flexibility to a degree similar to that of software. At the same time, they may lead to better performance and a smaller system size. However, the widespread acceptance of DRS depends on adequate support to design and implement them. A framework for the design, verification and implementation of DRS named PaDReH has been proposed by the authors as one step forward to reduce this lack of support. One of the main problems for enabling reconfigurable systems is the unavailability of efficient methods to control the hardware reconfiguration process. The main contribution of this paper is the proposition of a configuration controller totally built in hardware. This is different from previous approaches, where software implementations dominate. The proposed controller has been implemented and validated in VirtexII Xilinx FPGAs. The controller has been designed, validated and prototyped successfully.**

Index Terms— **Dynamically reconfigurable systems; reconfiguration control, Reconfigurable System Configuration Manager, run-time reconfiguration.**

I. INTRODUCTION

ALONG the previous decade, it is possible to notice a considerable increase of the 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 a longer time.

The time available to execute systems design flow continually decreases, because of market pressures. The use of configurable technology coupled to the massive reuse of intellectual property can make System-on-Chip (SoC) design quicker [2], adapting the design flow to current and future time-to-market restrictions.

An attractive factor to the use of 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 Run-Time Reconfiguration (RTR) techniques has potential to save resources while reducing the system area overhead. This happens because it allows that parts of the system 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 RTR are the performance penalty, induced by long reconfiguration times and the area overhead to implement the hardware responsible for controlling the reconfiguration process.

The deployment of RTR however requires extensive support that is not yet available [1]. This support is composed by tools to enable the use of RTR and infrastructure to implement Dynamically Reconfigurable Systems (DRS). Table I presents a summary of the main features still requiring non-existent support to enable that DRS develop its potential to become a mainstream technology.

TABLE I
MAIN FEATURES REQUIRING STRONGER SUPPORT TO ENABLE DRS.

Tools	- DRS Design; - DRS Verification;
Infrastructure of reconfiguration	- DRS-enabling devices; - Modules to control dynamic reconfiguration - Communication interface.

A DRS is seen here as being composed by a set of fixed modules and a set of reconfigurable modules. Each of the modules is a complex hardware piece, implemented by tens or hundreds of thousand equivalent logic gates. Each of the modules can then be thought as an Intellectual Property Core (or IP core) [2]. The implementation of DRSs usually assumes the availability of an infrastructure composed by specific modules for system control and operation. Among these modules, one is responsible to manage the system reconfiguration process, the *configuration controller*. This module commands which reconfigurable IP core(s) must be inserted on the reconfigurable device at any moment, and which must be removed. The main goal of the present work is to deal with this lack of infrastructure for DRS.

The rest of this paper is organized as follows. Section II presents PaDReH, a framework for the design, verification and

Work partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, under scholarship grants 130900/2002-8, 307655/2003-2, and CAPES BEX 0276/02 -2.

E. Carvalho holds an M.Sc. degree obtained at the PPGCC/FACIN/PUCRS in 2004. He is Research Assistant at PUCRS/Brazil since April, 2004. (e-mail: ecarvalho@inf.pucrs.br)

N. Calazans works at the PUCRS/Brazil since 1986. He is a professor since 1999. Prof. Calazans is the head of the Hardware Design Support Group (GAPH) at PUCRS. (e-mail: calazans@inf.pucrs.br)

F. Moraes works at the PUCRS/Brazil since 1996. He is a professor since August 2003. Prof. Moraes is a member of the Hardware Design Support Group (GAPH) at the PUCRS. (e-mail: moraes@inf.pucrs.br)

D. Mesquita is a Ph.D. student at the LIRMM-Montpellier, France. He holds an MSc form the PPGCC/FACIN/PUCRS, obtained in 2001. (e-mail: mesquita@lirmm.fr)

implementation of DRS. Section III contains a brief review on the state of the art for partial and dynamic reconfiguration techniques. The state of the art on configuration controller models and implementations is presented in Section IV. Next, a model of configuration controller is proposed, in Section V. Section VI describes a case study employed to validate the configuration controller model. A first set of implementation results using the case study appears in Section VII. Finally, Section VIII presents conclusions and future work.

II. PADREH – A FRAMEWORK FOR DRS

The present work is part of a framework to design and implement DRSs, named *Partial and Dynamic Reconfiguration of Hardware* - PaDReH, to enable the design of complex reconfigurable system. This framework helps in obtaining advantages from the use of dynamic and partial reconfigurable hardware technology. The PaDReH framework is composed by three main modules, as depicted on Fig. 1.

The first module of the framework, named *Design Capture and Validation*, comprises the description and validation of DRSs at the Transaction Level (TL) of abstraction and translation to the Register Transfer Level (RTL) of abstraction.

Next there is the *Partitioning and Scheduling* module, 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 Physical Synthesis and Reconfiguration Infrastructure module.

The last module of the PaDReH framework, named *Physical Synthesis and Reconfiguration Infrastructure*, is responsible for the generation of configuration files implemented as total and partial bitstreams, according to the partitioning defined in the previous module. This module 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. More details about PaDReH can be obtained in the original proposition of the environment, recently published in [4]. This reference also includes a comparison with other frameworks proposed recently in the literature of DRSs.

The Configuration Control module, proposed and implemented in this work, is part of the Physical Synthesis and Reconfiguration Infrastructure module as illustrated in Fig. 1.

III. DYNAMIC AND PARTIAL RECONFIGURATION

Several approaches were proposed to enable the use of dynamic and partial reconfiguration as revised in [1][5][6]. This Section is intended as a specific discussion on three topics relevant to this work: (*i*) commercial devices enabling DRS design and implementation; (*ii*) tools to generate partial bitstreams; (*iii*) methods used to interconnect IP cores in DRS.

A. Hardware devices enabling RTR

There are still very few commercially available semiconductor devices that enable the use of dynamic and partial reconfiguration. Atmel Inc. produces two series of partially reconfigurable devices: AT6000 and AT40k, but these allow implementing only small circuits, typically smaller than a hundred thousand logic gates. Another vendor, Xilinx Inc. commercializes four FPGA device series supporting dynamic and partial reconfiguration: Virtex, VirtexE, VirtexII and Virtex-II Pro series. Xilinx devices can reach up to 10 million equivalent gates, justifying their choice in this work.

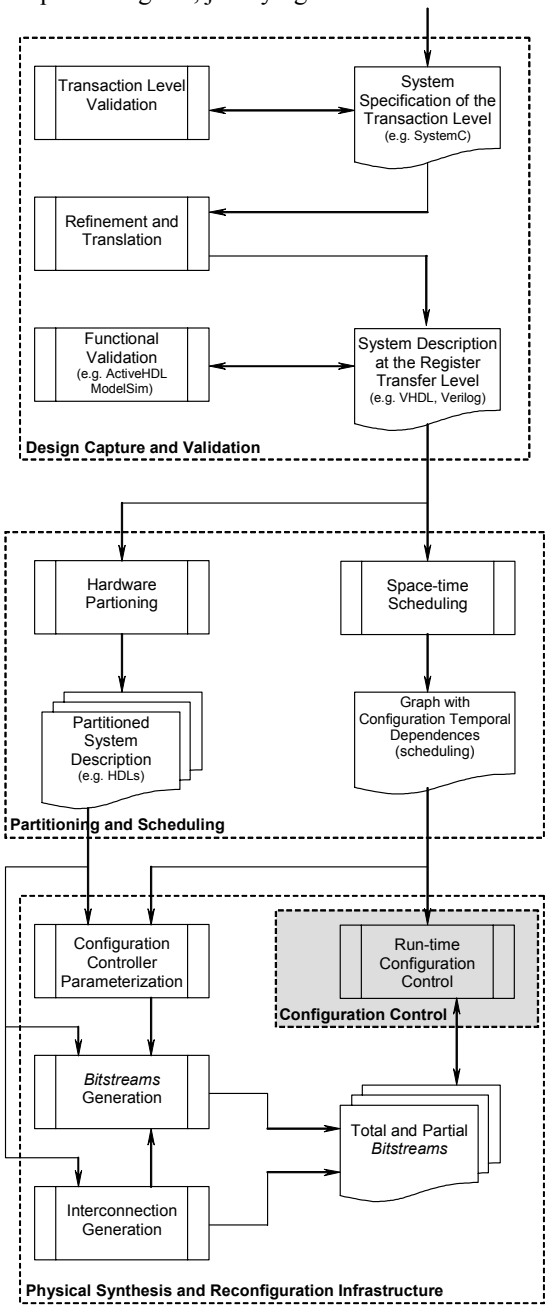

Fig. 1. The DRS PaDReH framework general structure.

B. Reconfiguration files manipulation

Generating partial reconfiguration files, also called *partial bitstreams*, is a crucial task for DRS development. The

production of partial bitstreams with current FPGA tools is a basically manual, complex and error-prone process. Some tools and techniques have already been proposed to automate the generation of partial bitstreams. Examples are JbitsDiff [7], Jbits [8], PARBIT [9] and JPG [10]. These tools allow *difference based* manipulations [11], where just localized portions of the total bitstream are changed in a given moment.

C. IP core interconnection schemes in DRS

Palma [12] suggests a method to generate the interconnection among partial bitstreams representing arbitrarily complex IP cores using a bus-based structure. The method is partially automated, but is limited by the difficulties to fine control the routing in Xilinx FPGA designs. Another technique for interconnecting dynamically replaceable cores in FPGAs has been recently proposed by Xilinx, called *module based* manipulations [11]. This technique, based on the Xilinx Modular Design flow, establishes a set of steps to generate partial bitstreams. Operating details of it have been provided briefly in [13] and extensively in [14].

IV. Configuration Controllers

As previously mentioned, one of the main problems for enabling DRSs is the unavailability of efficient methods to control the hardware reconfiguration process. A configuration controller commands which reconfigurable IP core(s) must be inserted on the reconfigurable device at any moment, and which may/must be removed. It executes tasks similar to those of a loader in an operating system, being responsible for loading configurations to execute on the reconfigurable hardware, according to a defined task scheduling. This Section reviews models and implementations of configuration controllers available in the literature.

Table II compares some of the most relevant configuration controller models and implementations proposed to date. The last column in the Table shows the characteristics of the configuration controller proposed here, detailed in Section V.

The most sophisticated model, proposed by Lysaght et al. [15] allows the use of advanced scheduling strategies, preemption, and allows the manipulation of compacted or encrypted data. The simplest model, proposed by Shirazi et al. [16] presents a generic configuration controller where relocation of configurations is not employed. On the other hand, Burns et al. [17] describe the structure of a DRS configuration controller, which has a sub module specific to relocate configurations. A discontinued device series, Virtex XC6200, is the target of both models. Curd [18] describes a configuration controller implementation for Virtex-II Pro devices. This configuration controller is implemented in software. Code executes in the Virtex-II Pro embedded PowerPC processor. It reads configuration data from RAMs and sends them to the Internal Configuration Access Port (ICAP), an internal port to reconfigure the FPGA, available in Virtex-II and Virtex-II Pro series FPGAs. Blodget et al. [19] also implemented a software configuration controller. However, this controller executes on the Xilinx MicroBlaze soft processor. Both implementations are based on specific models, different from others considered in this Section.

V. RSCM - A Proposal of Configuration Controller

The *Reconfigurable System Configuration Manager* (RSCM) is a model of configuration controller. The RSCM general structure, detailed in Fig. 2, comprises 6 modules: Configuration Memory, Self-Configuration, Configuration Interface, Reconfiguration Monitor, Configuration Scheduler and Central Configuration Control. The *Configuration Memory* (CM) stores all partial bitstreams used at runtime by the system. A partial bitstream generated by the technique employed here has at least 4Kbytes.

The PaDReH framework and the RSCM are designed to substitute arbitrarily large IP cores from the reconfigurable device. Consequently, the average size of partial bitstreams is typically much larger than the minimum, although typically much smaller than a total bitstream. For example, for a 1-million-gate Virtex-II device, a total bitstream is about 512Kbytes, but this FPGA contains only about 90Kbytes of internal RAM, distributed in several blocks. Consider the relatively large amount of memory needed to store even a medium amount of partial bitstreams, and the scarcity of memory in current FPGAs. These are the reasons why the Configuration Memory is the only module partially implemented outside the reconfigurable device. The internal part of the CM module is responsible to control access to an external memory. External memory bandwidth is not currently a problem, since present FPGAs reconfiguration technology is rather restrictive. For example, while synchronous SDRAMs

TABLE II - Features comparison for configuration controller models and implementations.

Features	***Models***			***Implementations***		***Propose***
	Shirazi	***Burns***	***Lysaght***	***Curd***	***Blodget***	***RSCM***
Hardware/Software	NA*	NA	NA	Software	Software	Hardware
Target device	XC6200	XC6200	XC6200	Virtex-II Pro	Virtex-II	Virtex-II
Scheduler type	Static	Static	Dynamic	No	No	Yes
Preemption Support	No	No	Yes	No	No	No
Relocation Support	No	Yes	Yes	No	No	Planned
Configuration storage	No	Yes	Yes	Yes	No	Yes
Configuration decoder	No	No	Yes	No	No	No
Controller Location	NA	NA	NA	FPGA	FPGA	FPGA
Publication date	1998	1997	1999	2003	2003	2004

*NA means Not Applicable

may easily operate at an access rate of more than 100MHz, Virtex-II FPGAs must be reconfigured using a clock that is below 33MHz [20].

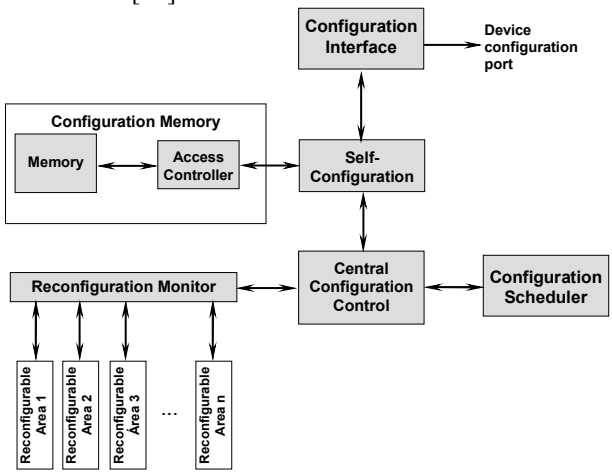

Fig. 2. DRS reconfiguration architecture using RSCM. The gray modules are components of the configuration controller. The other modules in picture are part of the rest of the DRS. Memory is the only module outside of the FPGA.

The *Self-Configuration* (SC) module controls the configuration process. It has an interface with the CM, to send control signals requesting configuration data. The interface with the Configuration Interface (CI) module allows sending configuration data to the FPGA. The Central Configuration Control (CCC) interface allows receiving requests to start the configuration process and providing results in the form of status signals. SC module can contain logic to control configuration relocation. CI is responsible for receiving configuration data from SC module 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 the RM and requests services to the CS and SC modules. Information about the device allocation status is stored into the *Table of Allocation of Resources* (TAR).

The *Configuration Scheduler* (CS) module is responsible to determine which configuration 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).

This work describes a version of RSCM implemented in hardware, but the model is generic. It may as well be implemented in software or 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 using external controlling devices.

Task context switching and communication between reconfigurable regions not necessarily present at the same moment in the DRS are not directly addressed by RSCM. Although these are important issues in DRS operation, it would be very complex to address them generically. RSCM assumes they are solved by each pair of communicating tasks in ad hoc ways, or through an external fixed processor running software, to provide maximum communication flexibility adapted to the set of requirements of each specific application.

More details about the operation of each RSCM module can be found in [22].

VI. Case Study: R8NR

Possibly, the most intuitive form of DRS is one where the instruction set of a processor is dynamically incremented through the use of reconfigurable hardware. Such components are generically known as reconfigurable processors. As a proof of concept for the RSCM controller, this Section proposes one such system, a processor with attached reconfigurable coprocessors, named R8NR. It should be pointed out that this implementation allows only a partial validation of the RSCM, since the case study does not encompass functionalities requiring the Reconfiguration Monitor or the Configuration Scheduler. Nonetheless, all modules of the RSCM have been individually and collectively validated in hardware through the use of small applications, such as Boolean operations on external switches values and display of results on leds, as described in detail in [22].

A. R8NR Structure

The main module of the R8NR is R8R, a processor based on the R8 processor, a 16-bit load-store 40-instruction RISC-like processor [23]. R8 was transformed into R8R by the addition of 5 new instructions, intended to give support to the use of partially reconfigurable coprocessors. R8R was wrapped to provide communication with the local memory, the system bus, the RSCM and the reconfigurable regions. The interface to the reconfigurable regions comprises a set of signals connected to bus macros.

Fig. 3 displays the general organization of the R8NR system implementation. The fixed part is a complete computational system, comprising the R8R processor, its local memory containing instructions/data and the RSCM. Additionally, not shown in the picture but implemented, there is a system bus controlled by an arbiter and peripherals to interface to a host computer. The RSCM acts as a slave of the R8R processor or the host computer. The host computer typically fills/alters the configuration memory before execution starts.

The case study structure follows a DRS implementation model proposed by the authors, called reconfigurable Coprocessor System or RCS [24], and was used to validate it. RCS is a model intermediate between the Application-Specific Instruction Set Processor model (ASIP) and the Reconfigurable Instruction Set Processor model (RISP).

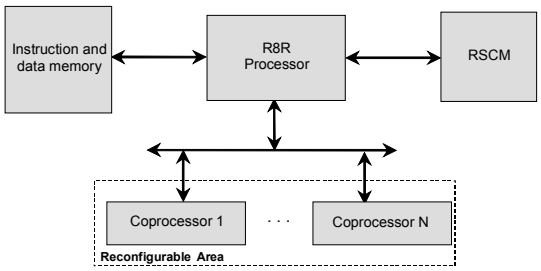

Fig. 3. R8R processor structure.

B. R8NR Behavior

The normal operation of the RSCM module is to wait for the R8R processor to produce coprocessor reconfiguration requests using the `rec_reconf` signal, while informing the specific coprocessor identifier in the `IOaddress` lines. If the coprocessor is already in place (an information internally stored in the RSCM), the `ack_remove` signal is immediately asserted, which releases the processor to resume instruction execution. If the coprocessor is not configured in some reconfigurable region, the reconfiguration process is fired. The RSCM is responsible to locate the configuration memory area where lies the coprocessor bitstream corresponding to the identifier. This bitstream is read from memory and sent, word by word, to the Physical Configuration Interface. For Virtex-II devices, this corresponds to the ICAP module. In this case, only after the reconfiguration process is over the `ack_remove` signal is asserted. The `req_remove` signal exists to allow the R8R processor to invalidate some reconfigurable coprocessor. This is useful to help the RSCM to better choose which the most adequate region to reconfigure next is.

The coprocessors are configured on demand, under control of the software that executes on the R8R processor. During 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, which 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.

For this case study, three coprocessors were implemented. The first, named *SQRT* coprocessor computes the square root of a 32-bit value and presents a 16-bit value as response. The *MULTI* coprocessor executes a multiplication of two 16-bit values and presents a 32-bit response. The *DIV* coprocessor executes a division of two 16-bit values and presents 32 bits response (quotient-remainder).

VII. RESULTS

The system described in Section VI 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.

The *reconfiguration time* (*Tr*) is composed by the sum of initialization and transmission times (Eq. 1). The *initialization time* (*Ti*) is the time to read and process the first configuration word from memory. The *transmission time* (*Tt*) is the time spent to send all other configuration words to the device configuration interface. *Tt* is the product of the *number of configuration words* (*Ncw*) of the *bitstream* and the *time to send each word* to the configuration interface (*Tw*). After replacing this into Eq. 1, Eq. 2 is generated.

$$Tr = Ti + Tt \quad (1)$$

$$Tr = Ti + (Ncw \times Tw) \quad (2)$$

An experiment was conducted to compare the partial and total reconfiguration times, using the S*elf-Configuration* module of the RSCM system (Eq. 3) and a Xilinx Multilinx USB configuration cable (Eq. 4). The cable reconfiguration uses the Xilinx Impact software. The latter is much slower due to the fact that the reconfiguration based on Impact is executed in the host computer.

$$Tr = 884ns + (Ncw \times 748ns) \quad (3)$$

$$Tr = 160ms + (Ncw \times 8{,}44\mu s) \quad (4)$$

Fig. 4 presents a plot comparing the reconfiguration times according to Eq. 3 and Eq. 4, as a function of the bitstream size (i.e. number of configuration words).

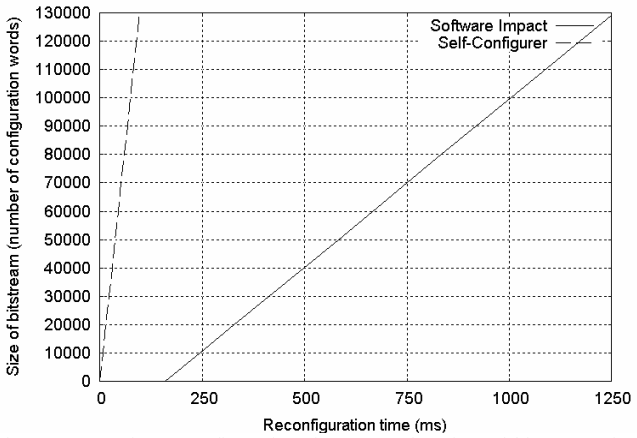

Fig. 4. Comparing reconfiguration times as a function of bitstream size. A total bitstream to configure a million-gate XC2V1000 Virtex-II device contains 127,581 32-bit words.

To compare execution times, software implementations of each coprocessor were used. The results are illustrated in Fig. 5. For the multiplication case, the execution of more than 750 consecutive multiplications will execute faster in hardware, even considering the reconfiguration time. For division and square root the respective break-even points occur for 260 and 200 operations, respectively. The determination of this break-even point is important to establish the advantage of using DRSs.

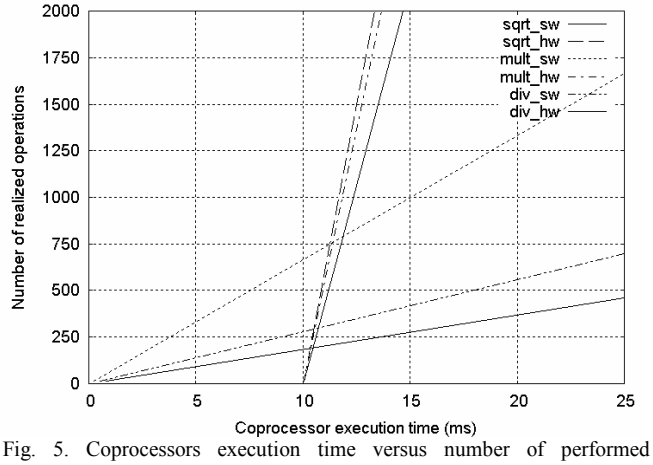

Fig. 5. Coprocessors execution time versus number of performed operations. The _hw suffix regards hardware implementations, while _sw regards software implementations. All hardware coprocessors were synthesized in the same reconfigurable area, corresponding to bitstreams of exactly the same size. Each bitstream is configured in approximately 10ms by RSCM.

Example applications where the above results can be applied are digital filters where a great number of multiply-accumulate operations are executed over a data set.

Table III presents area consumption data for the RSCM controller in a XC2V1000 Virtex-II Xilinx device. Data were obtained from logical synthesis using Leonardo Spectrum.

TABLE III
AREA CONSUMPTION DATA FOR RSCM IN A MILLION-GATE FPGA. THE CONFIGURATION INTERFACE MODULE EMPLOYS A BUILT-IN FPGA MODULE, JUSTIFYING ITS NULL AREA CONSUMPTION.

Module	**LUTs**	**%LUTs**
Configuration Interface	0	0.00 %
Reconfiguration Monitor	16	0.16 %
Configuration Scheduler	83	0.81 %
Central Configuration Control	126	1.23 %
Self-Configuration	268	2.62 %
RSCM (Total)	493	4.81 %

VIII. CONCLUSIONS AND FUTURE WORK

This work proposed a model and an associated hardware implementation of a configuration controller for DRSs named RSCM. This controller was completely prototyped in hardware and a proof-of-concept reconfigurable processor case study was employed to demonstrate its efficacy. The RSCM controller presents a small area overhead for medium to large devices (less than 5% of a million-gate FPGA). Execution time quantitative results indicate that the RSCM controller can be used to enable the construction of DRS applications that present performance gains with regard to software only implementations. Much greater improvement on the efficiency of configuration controllers for dynamic reconfiguration can be obtained if e.g. FPGA vendors make these available as optimized standard cells inside their devices. Also, the percentage of area occupied by the configuration controller is further reduced if bigger devices are used.

REFERENCES

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

[2] R. Bergamaschi and W. Lee. Designing system-on-chip using cores. In 37th DAC, p.420-425. USA. 2000.

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

[4] E. Carvalho, N. Calazans, E. Brião, F. Moraes. PaDReH - A Framework for the Design and Implementation of Dynamically and Partially Reconfigurable Systems. In: SBCCI, 2004.

[5] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, v. 34, no. 2, June 2002, pp. 171-210.

[6] E. Sanchez, M. Sipper, J-O. Haenni, J-L. Beuchat, A. Stauffer and A. Perez-Uribe. Static and Dynamic Configurable Systems. IEEE Transactions on Computers. v. 48, no. 6, p.556-564. 1999.

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

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

[9] E. Horta, J. Lockwood, S.T. Kofuji. Using PARBIT to implement Partial Run-time Reconfigurable Systems. In: FPL'02, Lecture Notes in Computer Science, pp 182-191, 2002.

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

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

[12] J. Palma, A. Mello, L. Möller, F. Moraes and N. Calazans. Core Communication Interface for FPGAs. In: SBCCI´2002. Brazil. 2002.

[13] F. Moraes, N. Calazans, L. Möller, E. Brião and E. Carvalho. Dynamic and Partial Reconfiguration in FPGA SoCs: Requirements and Tools. In: Reconfigurable Computing, organized by Wolfgang Rosenstiel. *Chapter approved to be published in Kluwer Academic Press Book*. In print, 2004.

[14] E. Brião and N. Calazans. Tutorials on Partial and Dynamic Reconfiguration using the Modular Design Design Flow on the Insight V2MB100 Platform. PPGCC-PUCRS Technical Report Series, TR033, 93 pages. October, 2003. (in Portuguese)

[15] D. Robinson and P. Lysaght. Modeling and Synthesis of Configuration Controllers for Dynamically Reconfigurable Logic Systems using the DCS CAD Framework. In FPL´99, Lecture Notes in Computer Science v. 1673. UK. 1999.

[16] N. Shirazi, W. Luk and P. Cheung. Run-Time Management of Dynamically Reconfigurable Designs. In FPL'98, Lecture Notes in Computer Science, Heidelberg: Springer-Verlag, v. 1482. Estonia. 1998.

[17] J. Burns, A. Donlin, J. Hogg, S. Singh and M. Wit. A Dynamic Reconfiguration Run-Time System. In FCCM'97, p.66-75. USA. 1997.

[18] D. Curd. Dynamic Reconfiguration of RocketIO MGT Attributes. Xilinx Application Note XAPP660, V2.1. November 2003.

[19] B. Blodget, S. McMillan and P. Lysaght. A Lightweight Approach for Embedded Reconfiguration of FPGAs. In: DATE´03, p.399-400. Germany. 2003.

[20] Xilinx, Inc. Virtex-II Platform FPGA Handbook. UG002 (v1.3). 3 December 2001.

[21] E. Carvalho, F. Möller, F. Moraes and N. Calazans. Design Frameworks and Configuration Controllers for Dynamic and Partial Reconfiguration. PPGCC-PUCRS Technical Report Series, TR042, 17 pages. June, 2004.

[22] E. Carvalho. RSCM - A Configuration Controller for Reconfigurable Hardware Systems. MSc Dissertation, PPGCC - FACIN - PUCRS, Porto Alegre, Brazil. March 2004. 150 pages. (In Portuguese)

[23] F. Moraes and N. Calazans. 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

[24] L. Möller, N. Calazans, F. Moraes, E. Brião, E. Carvalho and D. Camozzato. FiPRe: An Implementation Model to Enable Self-Reconfigurable Applications. In: FPL´04. Lecture Notes in Computer Science. Belgium. 2004.