Controlling Configurations on Dynamic Reconfigurable Systems

Ewerson Carvalho, Eduardo Brião, Leandro Möller, Frederico Möller, Fernando Moraes, Ney Calazans
{ecarvalho, briao, moller, frebm, moraes, calazans}@inf.pucrs.br

Abstract

Dynamically Reconfigurable Systems (DRSs) have the potential to provide hardware with flexibility similar to that of software. At the same time, they may lead to better performance and smaller system size. However, there is a clear lack of support devices, tools and design flows adequate for such systems. One of the main problems for enabling dynamically reconfigurable systems is the unavailability of efficient methods to control the hardware configuration process. The main contribution of this work is the proposal 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 Virtex-II Xilinx FPGAs.

1. Introduction

Along the previous decade, it is possible to notice a considerable increase of the interest on reconfigurable computing [ZHA00]. The potential flexibility provided by reconfigurable hardware is relevant and 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. Therefore, the use of configurable technology coupled to the massive reuse of intellectual property can accelerate System-on-Chip (SoC) design time [BER00], decreasing the time-to-market of technological products.

An 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 [DEH98]. The use of Dynamically Reconfigurable Systems (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 DRS 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 implementation of DRS 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. The main objective of the present work is to propose a solution to this lack of infrastructure for DRS, evaluating the relative costs of the approach.

The rest of this paper is organized as follows. Section 2 contains a brief review of the state of the art on DRSs. The state of the art on configuration controller models and implementation is presented in Section 3. From there, a model of configuration controller is proposed, in Section 4. Next, a case study used to validate the proposed model is presented in Section 5. A set of implementation results using the case study is analyzed in Section 6, while Section 7 presents some conclusions.

2. Dynamic and Partial Reconfiguration

Several approaches were proposed to enable the use of dynamic and partial reconfiguration as revised in [COM02][SAN99][ZHA00]. 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 DRSs.

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. Another vendor, Xilinx Inc. commercializes four FPGA device series supporting dynamic and partial reconfiguration: Virtex, VirtexE, Virtex-II and Virtex-II Pro series. Xilinx devices can reach up to 10 million equivalent gates, justifying its choice for use in this work.

The generation of partial reconfiguration files, also called partial bitstreams, is a crucial task for DRS development. The production of partial bitstreams with current FPGA tools is basically a manual, complex and error-prone process. However, some tools and techniques to automate the generation of partial bitstreams can already be found on the research literature as JbitsDiff [JAM00], Jbits [XIL03], PARBIT [HOR02] and JPG [RAG02]. These tools allow difference based manipulations where just very small, localized portions are changed at a given moment.

Palma [PAL02] 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 [XIL03a]. This technique, based on the Xilinx Modular Design flow establishes a set of steps to generate partial bitstreams.

3. Configurations Controllers

One of the main problems for enabling reconfigurable systems 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 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. This Section reviews models and implementations of configuration controllers.

Tab.1 compares all configuration controller models and implementations that could be found in the available literature. The last column in the Table shows the characteristics of the configuration controller proposed here, detailed in Section 4. The most sophisticated model, proposed by Lysaght et al. [ROB99] 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. [SHI98] presumes a generic configuration controller where relocation of configurations is not employed. On the other hand, Burns et al. [BUR97] describe the structure of a DRS configuration controller, which has a sub module specific to relocate configurations.

Tab.1- Feature comparison of configuration controller models and implementations.

<table>
<thead>
<tr>
<th>Features</th>
<th>Shirazi</th>
<th>Burns</th>
<th>Lysaght</th>
<th>Implementations</th>
<th>Curd</th>
<th>Blodget</th>
<th>RSCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware/Software</td>
<td>NA*</td>
<td>NA</td>
<td>NA</td>
<td>Software</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Target device</td>
<td>XC6200</td>
<td>XC6200</td>
<td>XC6200</td>
<td>Vortex-II Pro</td>
<td>No</td>
<td>No</td>
<td>Vortex-II</td>
</tr>
<tr>
<td>Scheduler type</td>
<td>Static</td>
<td>Static</td>
<td>Dynamic</td>
<td>Dynamic</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Preemption Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Planned</td>
</tr>
<tr>
<td>Relocation Support</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Configuration storage</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Configuration decoder</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Controller Location</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>FPGA</td>
<td>FPGA</td>
<td>FPGA</td>
<td>FPGA</td>
</tr>
</tbody>
</table>

*NA means Not Applicable

Curd [CUR03] describes a configuration controller implementation for Virtex-II Pro devices. This configuration controller is implemented in software. A code executes in the Virtex-II Pro embedded PowerPC processor. It reads configuration data from internal 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. [BLO03] 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 those previously considered in this Section.

4. 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. 1, comprises 6 modules: (i) Configuration Memory, (ii) Self-Configuration Module, (iii) Configuration Interface, (iv) Reconfiguration Monitor, (v) Configuration Scheduler and (vi) Central Configuration Control.

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

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 provides results of the reconfiguration 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 for 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.
XIV Microelectronics Seminar

Configuration Memory

Self-Configuration

Central Configuration Control

Configuration Scheduler

Reconfiguration Monitor

Device configuration port

Configuration Interface

Access Controller

Fig. 1 - RSCM model and its usage in the implementation of DRSs.

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.

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).

In this work, the RSCM model is implemented in hardware, but since the model is generic, it could 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 resource to external controlling devices.

5. Case study: R8NR

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. As a proof of concept for the RSCM controller, this Section proposes a system, named R8NR, composed by a processor with N attached reconfigurable coprocessors.

The R8R processor is based on the R8 processor, a 16-bit load-store 40-instruction RISC-like processor [MOR03]. The original R8 processor was transformed into the R8R processor by the addition of 5 new instructions intended to give support to the use of partially reconfigurable coprocessors [MOL04]. The R8R processor 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 [XIL03a].

Fig. 2 displays the organization of the R8NR system implementation. The fixed part in the FPGA is a complete computational system, comprising a simplified version of the RSCM, the R8R processor and its local memory containing instructions and data. Additionally, there is a system bus controlled by an arbiter and peripherals to interface to a host computer, not shown in the picture but implemented.

Fig. 2 - R8R processor structure.

The RSCM acts as a slave of the R8R processor or the host computer. The host computer typically fills or alters the configuration memory before system execution starts. 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.

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).

6. Results

The system described in Section 4 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.

An experiment was conducted to compare the partial and total reconfiguration times, using the Self-Configuration module of the RSCM system against the reconfiguration from the download software (Impact). Fig.3(a) presents a plot comparing the reconfiguration times as a function of the bitstream size (i.e. number of configuration words). It is possible to notice the obviously better performance obtained using the Self-Configurer of the RSCM controller.

To compare execution times, software implementations of each coprocessor were used. The experiment consisted in comparing the execution time of hardware and software versions versus the number of times the operation is executed. The results are illustrated in Fig.3(b). 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. The determination of this break-even point is important to establish the advantage of using DRSs. This break-even point is indeed a worst-case situation, since the use of more than one reconfigurable area, coupled with powerful scheduling strategies implemented by the processor in software can potentially hide reconfiguration times behind the parallel of useful work by other running coprocessors and the processor in itself.

Tab.2 presents a quantitative analysis for the implemented RSCM controller. These data were obtained from logical synthesis using the Leonardo Spectrum tool. This table presents area consumption data for the RSCM controller for a XC2V1000 Virtex-II Xilinx device.

Tab.2 - Area consumption data for RSCM in million-gate FPGA. The configuration interface module employs a built-in FPGA module, justifying its null area consumption.

<table>
<thead>
<tr>
<th>Module</th>
<th>LUTs</th>
<th>DFFs</th>
<th>Area (%LUTs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Interface</td>
<td>0</td>
<td>0</td>
<td>0.00 %</td>
</tr>
<tr>
<td>Reconfiguration Monitor</td>
<td>16</td>
<td>11</td>
<td>0.16 %</td>
</tr>
<tr>
<td>Configuration Scheduler</td>
<td>83</td>
<td>34</td>
<td>0.81 %</td>
</tr>
<tr>
<td>Central Configuration Control</td>
<td>126</td>
<td>67</td>
<td>1.23 %</td>
</tr>
<tr>
<td>Self-Configuration</td>
<td>268</td>
<td>106</td>
<td>2.62 %</td>
</tr>
<tr>
<td>RSCM estimated</td>
<td>493</td>
<td>218</td>
<td>4.81 %</td>
</tr>
</tbody>
</table>
7. Conclusions

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.

8. References


