A General Purpose Partially Reconfigurable Processor Simulator (PReProS)

Alisson V. Brito[1,2], Matthias Kuehnle[2], Elmar U. K. Melcher[1], Juergen Becker[2]

[1] Federal University of Campina Grande
Department of Electrical Engineering
Campina Grande, 58.109-970 Brazil
{alisson,elmar}@dee.ufcg.edu.br

[2] Universitaet Karlsruhe (TH)
Inst. für Technik der Informationsverarbeitung
Karlsruhe, D-76128 Germany
{brito,kuehnle,becker}@itiv.uni-karlsruhe.de

Abstract

An innovative technique to model and simulate partial and dynamic reconfigurable processors is presented in this paper. The basis for development is a SystemC kernel modified for dynamic reconfiguration. The presented approach can either be used at transaction-level, which allows the modeling and simulation of higher-level hardware and embedded software, or at register transfer level (RTL), if the dynamic system behavior is desired to be observed at signal level. The reconfigurable processor can be easily set to model the desired architecture in a behavioral but reasonable way. An example is presented where a XPP processor is implemented and simulated, executing typical applications. The resulting statistics assist either in the choice of the best cost/benefit configuration area that should be available on chip, or in the choice of the target architecture itself.

1. Introduction

Nowadays the dynamic and partial run-time reconfiguration is a reality [1, 2]. It is offered by a great number of vendors such as at fine- (like FPGAs [5, 6, 7]) as well as at coarse-grained (like XPP [3] and KressArray [4]) architecture level.

By using this feature less configuration memory is necessary since not actually used modules in a system do not allocate configuration memory. Thus it is possible to reduce the necessary configuration area and develop lower-cost and more power efficient systems. However, the timing and power consumption necessary to perform consecutive configurations should be considered. Power dissipation by run-time reconfiguration has been investigated in [8].

1-4244-0910-1/07/$20.00 ©2007 IEEE

For each system design, in order to evaluate the positive aspects of the dynamic and partial run-time reconfiguration, and to decrease the negative impacts of the known trade-offs, it is necessary to know them as soon as possible in the design flow. Normally, the relation between the configuration time and the necessary chip area is not known before the system has been programmed on chip during the testing phase. This is at least the case for complex systems.

This paper uses a simulation technique (described before in [17]) to implement a general purpose simulator for processors, whereas this technique supports run-time reconfiguration. Such a technique uses high-level representations to model and simulate the reconfiguration, giving the opportunity to designers to foresee the dynamic behavior of your system before the hardware is going to be implemented for the target architecture, or even before the system specification in HDL, if desired.

An important aspect is the integration of such modeling and simulation techniques into the design flow. It should be as general as possible in order to avoid re-working and time spending on using different formalisms to implement, model and simulate the system. To provide this light-weight integration, our technique uses the SystemC [9] description language, to model as well as to simulate the partial and dynamic reconfigurations. These capabilities are presented by an easy to use, well defined API. Any system described with SystemC can have the dynamic and partial reconfiguration capabilities added to its behavior and can be simulated. On the other hand, the system can easily be parameterized to reach its previous behavior again.

There are other efforts to provide these or similar functionalities. The Adriatic project [10] presents a system-level modeling technique for dynamically reconfigurable systems. It concerns on selecting candidates for dynamic reconfiguration. It is used specifically for a hardware architecture formed by a

Dynamically Programmable Gate Arrays (DPGA [11]) and a co-processor responsible for reconfigurations management. The modeling and simulation processes are implemented using SystemC, but it is not able to simulate the dynamic behaviors of the modeled systems. The OSSS+R project [12] aims at using the Object-Oriented concepts as inheritance and polymorphism in order to simulate dynamic reconfiguration. It implements an extension to SystemC, adding commands, to be able to dynamically switch modules during simulation. It enables the switching operation among elements descendent from the same base class, using the concept of inheritance. Modules with identical interfaces can be dynamically switched, but not disabled or removed from the system without them being replaced by other equivalent modules. Our approach attacks the dynamic and partial behavior in a more general way. Any module can be removed, added or switched at simulation-time.

This paper is organized into different sections. Section 2 presents the technique that made modeling and simulation of run-time reconfigurable systems possible. It also shows a simple example about how these features can be used. Section 3 presents the concept of the Partially Reconfigurable Processor Simulator, named PReProS. A usage scenario is presented in section 4, where the simulator is configured and tested. The simulation results are presented in section 5. Section 6 presents the final considerations and gives an outline about further works.

2. Simulation of dynamic reconfiguration

In order to simulate dynamic and partially reconfigurable systems, the simulator must perform some specialized operations. A partially reconfiguration simulator should perform basic operations such insertion and removal of modules into and from running systems. These operations are the basis for all possible operations to perform dynamic and partial reconfiguration.

The idea is to provide these two basic operations for SystemC in the form of two different routines, named *dr_sc_turn_on* and *dr_sc_turn_off*. They were implemented by applying some modifications to the SystemC kernel.

Each SystemC module can have one or more processes, which executes the module's algorithms. During each simulation cycle, the execution table is checked by the simulator and all requested processes are executed. The routine *crunch* from *sc_simulation_context* class is responsible for executing every process.

A linked list named *configList* was implemented to store the name of the modules that should not be executed. At each simulation cycle, before executing the processes, the module name is searched on the linked list, if it is there, its execution is avoided and the simulation keeps normally working. From the entire system point of view, the avoided processes are like non-existing processes.

The routines *dr_sc_turn_on* and *dr_sc_turn_off* respectively add and remove modules names from *configList*. Once *dr_sc_turn_off* routine is called during simulation referencing a module name, processes from this module will not be executed until a call to routine *dr_sc_turn_on* is executed.

2.1. Proof of concept

A simple example was implemented in order to explain how to use the simulation routines dr_sc_turn_on and dr_sc_turn_off. It contains two modules (moduleA and moduleB), which generate two different waveforms (see Figure 1). The modules produce square waveforms. While moduleA produces a waveform with a period of 1ns, moduleB's output shows a period of 3ns. These two modules are connected to the same signal, which is traced. This signal presents a mixed behavior, where the two modules sending data at the same time. Using the routines dr_sc_turn_on and dr_sc_turn_off, the modules will work always at different moments, avoiding the overlapping of their signals. This way, two scenarios are formed, one with just moduleA configured to work, and the other with the contrary situation, where just moduleB is configured, hence in use. In the simulation's context, just one module is working at each moment.

Figure 2 presents the waveforms, generated from this simulation. It shows the signals of moduleA and moduleB, being executed separately without partial reconfiguration in order to show their behavior only. The output signal produced during partial reconfiguration is named reconf and it aggregates the behavior of moduleA until 30ns, and the behavior of moduleB during remaining simulation time. This waveform illustrates that the system really changed its behavior during simulation time.

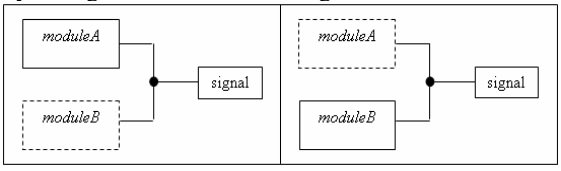

Figure 1. Two scenarios switched dynamically at simulation time. At each scenario just one module is configured.

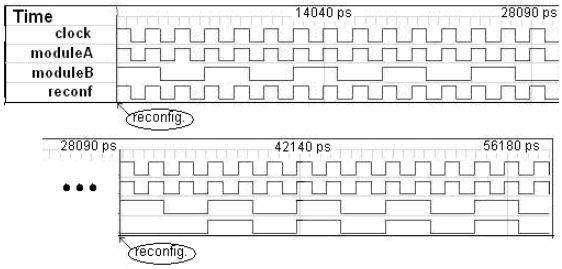

Figure 2. Waveforms from the example. The reconfiguration moments are pointed.

3. The Partially Reconfigurable Processor Simulator (PReProS)

Considering the simulation of dynamic and partially reconfigurable systems, a couple of steps should be done, like the target architecture specification, the definition of necessary hardware resources and the designing of the applications. The presented approach aims at writing a reusable parametrizable SystemC program able to model and simulate real target processor architectures. For example, coarse-grained, like XPP [15], which consists of configurable ALUs communicating via a packet oriented, automatically synchronized communication network. Also like PiCoGA, a reconfigurable architecture based on a very long instruction word RISC processor featuring an embedded programmable hardware unit that implements a pipelined, run-time configurable data path and KressArrays. Further, fine-grained architectures, like standalone FPGAs and embedded FPGAs (e.g. FlexEOS from M2000), which have the well known FPGA behavior, or any other, running any kind of application.

The goal is to parameterize the individual processor's characteristics in such a general way that all kind of processing element can be fully described using this set of parameters. The main features that have to be considered here are the clock frequency, properties of the data and configuration ports, and the number of chip area available on chip. In the same way, the applications' properties can be set by the frequency, needed ports, data width and number of configured area units.

When using this simulator, the designer should just have to set the parameters and implement its own blocks to configure the applications and exchange data with the PReProS, as shown on Figure 3.

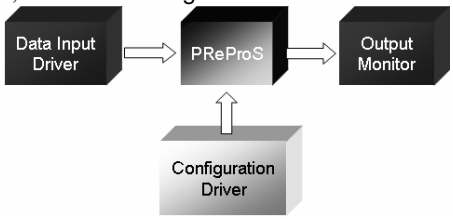

Figure 3. Usage example of the Dynamically Reconfigurable Processor.

After the simulation, statistic files are generated that illustrate e.g. chip performance and the area consumption. The chip area usage and the configuration delay impact are by default generated by the SystemC kernel, when using the above mentioned approach for dynamic reconfiguration simulation.

3.1. Simulator Specification and Implementation

To get a deeper understanding of the functionality of the system, a detailed description of the architecture is given in this section. This will not only show the programmed modules but it will also demonstrate the user-friendliness of the simulator.

Therefore, the approach is divided into different steps. One is the physical architecture of the processor that has to be set and simulated. The second step takes care of the programming model and the data exchange. Figure 4 shows the parametrizable, implemented architecture. The processor can be set through its property parameters, while the other elements should be implemented outside, as SystemC module, and connected to the simulator (see Figure 3).

Let us consider physical constraints only in the following paragraph. As mentioned above, the number of ports, their bit width, the processor's frequency and the area are parametrizable. The area units are arranged as an array, where column and line size can be set individually. The area units have to be chosen in a way to reflect the smallest programmable processing element of the complete processor. Taking a FPGA as example, the processing element could be a Slice. If greater resolution is desired, also the single LUTs can be defined to be the smallest area units. However, considering a coarse grained array, a complex ALU can be mirrored by a basic area unit. The processing power can then be adapted by setting the frequency respectively. This generic approach allows the emulation of any kind of reconfigurable processing element, any parallelism and every processing power.

Although the user is intended to deal with an architectural model as shown in Figure 4, which mirrors a high level view of the processor's physical architecture, the implementation is different. This was done intentionally to be able to better handle, first of all movement of application data, but also the reconfiguration mechanism. Using this approach, a geometry independent implementation is possible. Figure 5 illustrates that no processing elements are observed but rather slots, where the maximum number of slots equals the number of physical ports. By contrast, from the application point of view, more than one port can be used by one configuration.

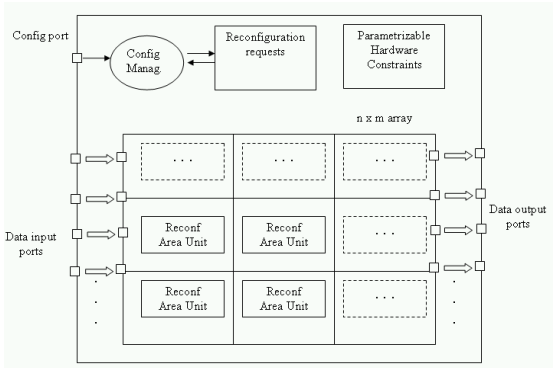

Figure 4. Parameterizable Dynamic Reconf. Processor Architecture

The data ports in Figure 5 are bidirectional, where the transfer protocol of the data driver can send and receive data from all ports in parallel. Finally, since we are not dealing with the physical location of each area unit, it remains unnecessary to take care of the geometrical relation of the connected port to the configured area units. The same holds for the area units, configured for one application, among each other. It is assumed that a well defined floorplan protects from unforeseen arising problems concerning the area constraints. Therefore, the configuration driver's task is merely to check the availability of the area and the desired port, whereas it isn't necessary to check, if the chosen area and the port are compatible in terms of their geometrical location on chip.

Referring to Figure 3, it can be seen that these tasks are executed by the drivers. The whole simulation is triggered by the configuration management. User defined configuration requests are processed in a serial order where the actual configuration bit stream is loaded right at the request time or just after demanded ports and area are available.

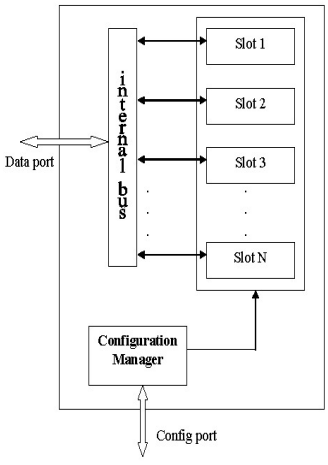

Figure 5. Processor's Application view

This request time has to be specified before the simulation. It is intended to emulate any kind of user interaction, interrupts or any unpredictable requested service routine. Dependent on the bit stream size and the bandwidth, calculated from bit width and the frequency of the configuration port, the configuration driver can exactly maintain real configuration timing. All important time stamps are logged and can later contribute to a statistical performance analysis.

Once parameterized, the user can implement its own algorithms for configuration and data exchange. The simulator can than be used not just to simulate different architectures, but mainly to model and simulate the dynamic and partial reconfiguration of applications, running in real systems, all of this in a fast and practical way. All data about chip area usage, reconfiguration delaying and data exchanging performance are generated automatically, making the analysis phases shorter as usual.

4. Use Case

For testing purposes, different processors have been examined and case studies have been implemented on the simulator. The selection was decided on top of available information for the examined reconfigurable processor. It was not only necessary to having collected information about the physical constraints of the processor, but also information about performance and timing of already mapped configurations and data processing were important. In the following, without loss of generality a coarse grained array, which is called the eXtreme Processing Platform (XPP) architecture [16] is chosen and examined in greater detail. This processor consists of configurable ALUs that communicate via a packet oriented, automatically synchronized communication network. Table 1 shows some parameters from the XPP II array that are implemented in the simulator model. All of the parameters are set in the processor model before system simulation startup.

To be able to validate the simulation results, existing tables of already mapped applications were taken. Table 2 shows the algorithms and their parameters. They have been implemented in the configuration driver in the order as shown by the application index.

frequency	200 MHz
number of ALUs (coarse grained elements)	8*8= 64
ports	4 IOs of 2*16 bit
total configuration bitstream	16kbyte
configuration bitstream for one area unit	0,25kbyte per ALU
configuration bitwidth	42

Table 1. Processor parameter of the XPP array

Right after simulation startup, the configuration driver is checking the user requests that are implemented in a configuration list. If the user request time is equal to the actual time stamp, the configuring procedure will start where the duration is dependent on the earlier defined parameters of the processor and the size of the bit stream.

app. index	application	configuration time	used area	free area
1	Fir filtering	4000 cycles	64	0
2	IIR filtering	4000 cycles	64	0
3	Multichannel viterbi	1340 cycles	22	42
4	Fourier transform	1250 cycles	20	44
5	Adapt beamforming	1625 cycles	26	38

app. index	number of configurations	Performance (ops per cycle)	used ports
1	1	128	4
2	1	128	2
3	2	43	1
4	3	40	1
5	2	52	1

Table 2. Parameter of the application/ algorithms for 8x8 XPP

Just after a single configuration is finished, the configuration driver triggers the data driver to notify the readiness of the configuration. The data driver is now able to start sending data to the proper ports. These ports are locked until being released again by the end of transfer signal of the data driver. After the notification of the configuration driver, the configured area will be released again. For any further configuration requests, feedback from the processor is demanded by the configuration driver. Feedback can be: occupied area or ports. This way, the consistency of the configured parts of the chip is assured. The simulation will be finished, when no more configuration requests are located in the request queue and no more data is being processed. Finally, with the help of the generated statistics, area occupation and timing analysis can be performed easily. Just to give an example: the multiplication of the frequency with the configuration time in clock cycles. This results in the real configuration time for each application as given in Table 2.

Finally, these values can be compared with the statistics that are generated by the simulator after the simulation.

5. Results

Some performance results were generated automatically by the PReProS, without the user writing any additional command. The results are presented below. Table 3 presents the performance referring to the data ports of the simulated system based on the XPP architecture.

application	#ports	sending data rate (MB/sec.)	receiving data rate (MB/sec.)
Fir filtering	4	3200	3200
IIR filtering	2	1600	1600
Multichannel viterbi	1	800	800
Fourier transform	1	800	800
Adapt beamforming	1	800	800

application	data sent (bytes)	data received (bytes)	total data rate (MB/sec.)
Fir filtering	6400	6400	6400
IIR filtering	1600	1600	3200
Multichannel viterbi	400	400	1600
Fourier transform	400	400	1600
Adapt beamforming	1600	1600	1600

Table 3. Data ports performance of the XPP processor

These results present a reasonable accuracy when compared to other related works, which made these measurements on chip [16].

The configuration performance can be seen on Table 4. This table shows the time stamp of request (cfg_request), configuration (config), start, and end of execution for each application. With this information the configuration and the response time for each application can be clarified. It depends directly on configuration bitstream size and on the chip usage of the specific moment. The FIR algorithm, for example, having 16Kbytes of configuration bitstream, took 1.9ms from its request until its configuration on chip. On the other hand, the Fourier transform algorithm, with a configuration bitstream of 5Kbits, needed only 0.6ms for the configuration.

application	t (cfg_request)	t (config)	t (start)
Fir filtering	10 ns	1915 ns	2020 ns
IIR filtering	1925 ns	3830 ns	3930 ns
Multichannel viterbi	3840 ns	4495 ns	4595 ns
Fourier transform	4505 ns	5105 ns	5205 ns
Adapt beamforming	5115 ns	5890 ns	5995 ns

application	t (end)	config bitstream (bits)	config rate (MB/sec.)
Fir filtering	4020 ns	16000	8400
IIR filtering	4930 ns	16000	8400
Multichannel viterbi	5095 ns	5500	8400
Fourier transform	5705 ns	5000	8300
Adapt beamforming	8000 ns	6500	**8400**

Table 4: Configuration timing and performance

More details about chip utilization can be found in the figures below. Figure 6 presents timing, when each application is configured on chip and how many ALUs each one is using at the moment. It is of interest to observe the parallelism of the application execution on chip. For example, the IIR works together with FIR and later with Viterbi, after that it is removed from chip, leaving space for the other applications.

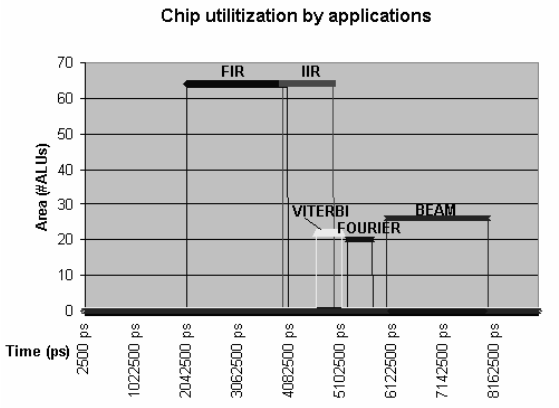

Figure 6: Chip area utilization by each application

On Figure 7 it is possible to see the amount of used resources of the chip. The XPP processor was simulated containing 144 ALUs, instead of the mentioned 64. In this way more parallel configurations can be simulated. The free area is marked by the darker area in the picture. By investigating these results, the best parallel performance and hence the best processing power and efficiency of the simulated processor can be achieved. It helps the designer to reevaluate his/her algorithms and implementation strategy, or if the selected architecture should be changed to better target his needs.

6. Final Considerations

This paper presents a general purpose partially reconfigurable processor simulator, named PReProS. Its intention is to support hardware as well as software designers. Hardware designers take benefit from the possibility of the simulators ability to quickly change processor parameters, whereas software designers can investigate the end of simulation statistics to reorganize their configuration scheduler, where the statistics help in the choice of the best cost/benefit configuration area that should be available on chip, or in the choice of the target architecture itself. This approach helps to easily model and simulate complex systems in a very short timeframe.

As further work, these capabilities are being considered to be executed by an embedded processor. It would be possible to analyze the system resources and dynamic behavior at run-time, enabling faster and more intelligent decisions. These features will be potentially used by different heterogeneous dynamically reconfigurable platforms supported by European research projects, like Morpheus (http://morpheus.arces.unibo.it), 4S (www.smart-chips.org) and Aether (www.aether-ist.org).

Acknowledgments

This research work is supported by "Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)", Brazil.

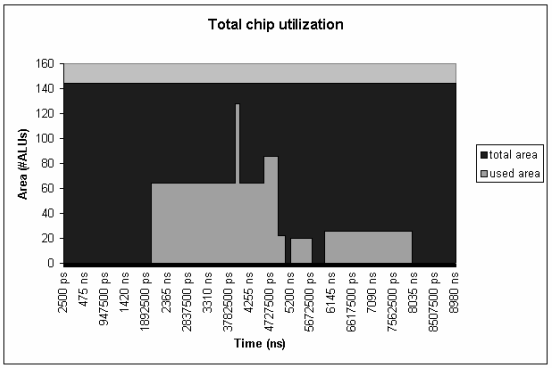

Figure 7: Total chip area utilization

References

[1] Reiner Hartenstein, "Coarse Grain Reconfigurable Architectures" Proceedings of the Conference on Asia South Pacific Design Automation, 2001.

[2] Becker, J., Hartenstein, R., Configware and morphware going mainstream. Journal of Systems Architecture. 49, 4-6, 127-142, September, 2003.

[3] Becker, J., Vorbach, M., Architecture, Memory and Interface Technology Integration of an Industrial/Academic Configurable System-on-Chip (CSoC), IEEE COMPUTER SOCIETY. ANNUAL Symposium on VLSI, Tampa, Florida, February 2002.

[4] Kress, R. "A Fast Reconfigurable ALU for Xputers". Ph.D. Thesis, University of Kaiserslautern, 1996.

[5] ATMEL. Application Note: Implementing Cache Logic with FPGAs. USA, Sep. 1999. Available at http://www.atmel.com (latest access august, 2006).

[6] Xilinx Inc. http://www.xilinx.com/products

[7] Ferrandi, F., Santambrogio, M., Sciuto, D., A Design Methodology for Dynamic Reconfiguration: The Caronte Architecture. Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'2005), 2005, Denver, CA, USA.

[8] Becker, J., Hübner, M., Ullmann, M.: "Real-Time Dynamically Run-Time Reconfiguration for Power-Costoptimized Virtex FPGA Realizations", VLSI03, Darmstadt, Sep. 2003.

[9] SystemC Community. http://www.systemc.org

[10] Qu, Y., Tiensyrja, K. Masselos, K., System-Level Modeling of Dynamically Reconfigurable Co-Processors, International Conference on Field Programmable Logic and Applications, Antwerp, Belgium, August-September, 2004.

[11] Dehon, A., DPGA Utilization and Application, in FPGA'2006, pp 115-121, February, 1996.

[12] Schallenberg, A., Oppenheimer, F., Nebel, W. Designing for Dynamic and Partially Reconfigurable FPGAs with SystemC and OSSS, Forum on Specification and Design Languages (FDL '04), Lille, France, Sept. 2004.

[13] Huebner, M., Becker, T., Becker, J., "Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration", 17th Brazilian Symposium on Integrated Circuit Design (SBCCI04), Brasil

[14] Blodget, B., McMillan, S., "A lightweight approach for embedded reconfiguration of FPGAs", DATE03, Munich, Germany

[15] Becker, J., Huebner, M., Ullmann, M., "Power Estimation and Power Measurement of Xilinx Virtex FPGAs: Trade-offs and Limitations", SBCCI03, Sao Paulo, Sep. 2003

[16] PACT, "The XPP White Paper – release 2.1". http://www.pactxpp.com

[17] Brito, A. V. , Melcher, E. U. K. ; Rosas, W. . An open-source tool for simulation of partially reconfigurable systems using SystemC. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006), 2006, Karlsruhe. IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006), 2006. v. 1. p. 434-435.