

FiPRE: An Implementation Model to Enable Self-Reconfigurable Applications

Leandro Möller, Ney Calazans, Fernando Moraes, Eduardo Brião,
Ewerson Carvalho, Daniel Camozzato

Pontifícia Universidade Católica do Rio Grande do Sul (FACIN-PUCRS),
Av. Ipiranga, 6681 - P 30/Bl 4 - 90619-900 - Porto Alegre – RS– BRASIL
{moller, calazans, Moraes, briao, ecarvalho,
camozzato}@inf.pucrs.br

Abstract. ASIPs and reconfigurable processors are architectural choices to extend the capabilities of a given processor. ASIPs suffer from fixed hardware after design, while ASIPs and reconfigurable processors suffer from the lack of a pre-established instruction set, making them difficult to program. As intermediate choice, reconfigurable coprocessors systems (RCSs) contain dedicated hardware (coprocessors) coupled to a standard processor core to accelerate specific tasks, allowing inserting or substituting hardware functionalities at execution time. This paper proposes a generic model for RCSs, targeted to reconfigurable devices with self-reconfiguration capabilities. A proof-of-concept case study is presented as well.

1. Introduction

A single processor may meet the requirements of several embedded system scenarios if it is somehow parameterizable. Application-specific instruction set processors (ASIPs) and reconfigurable instruction set processors (RISPs) [1] are two opposite forms of implementing processors with regard to *design* versus *runtime* parameterization trade-offs. *ASIPs* provide flexibility and performance at the cost of extra silicon area for each new function directly supported in hardware. If the application requires a new specific functionality, the ASIP is redesigned. *RISPs* are processors where some or all instructions are implemented as dedicated hardware and loaded on demand, according to the software execution flow. Here, the highest degree of flexibility is achieved. However, the lack of a pre-established fixed instruction set makes it harder to generate the object code for new applications. This occurs because each new function must be supported at the same time by the dedicated hardware *and* the compiler.

An intermediate solution, named *reconfigurable coprocessors systems* (RCSs) is addressed in this paper. As ASIPs and RISPs, RCSs contain dedicated hardware (coprocessors) to accelerate specific tasks. However, these are not fixed at design time as in ASIPs. It is possible to insert or substitute hardware functionalities at execution time in RISPs and RCSs without having to redesign the processor. Contrary to what happens in RISPs, RCSs contain a standard processor core with a fixed instruction set, enabling the use of standard compilers.

The processor and the parameterizable parts are *loosely coupled* in RCSs and *tightly coupled* in RISP and ASIPs. Additionally, ASIPs and RISP are inherently sequential approaches, while RCSs may benefit from the parallel execution of the processor software and dedicated computations in each coprocessor. Communication between the processor and the coprocessors can be achieved in this case through the use of e.g. interrupts.

A potential performance bottleneck faced by embedded applications in RISP and RCSs is the latency to perform hardware reconfiguration, which can be orders of magnitude longer than the time to perform application atomic operations. To reduce or eliminate this problem, RISP and RCSs assume the existence of an infrastructure to control the storage and the dynamic loading of hardware configurations, usually called a *configuration controller* [2].

Consider the current trend to increase the number of embedded processors in SoCs, leading to the concept of “sea of processors” systems [3], and add to this the above discussion on implementation alternatives for parameterizable embedded processors. From these, it is possible to justify the objective of this paper, which comprises proposing a generic implementation model for RCSs called FiPRE, and introducing a case study used to evaluate the ideas behind the model.

2. The FiPRE Implementation Model and the R82R Case Study

The FiPRE (**F**ixed core **P**rocessor connected to **R**econfigurable Coprocessors) model is conceived to allow self-reconfigurable applications implemented as RCSs and can be understood from the case study example in Fig. 1. First, there is a Fixed Region, which has an embedded processor to execute applications and to trigger reconfiguration actions. This region also contains a configuration controller (CC), to manage the details of the reconfiguration process. The existence of external devices intended to provide input/output capabilities for the embedded system is also part of the model. Besides, a memory is needed to store coprocessor bitstreams, a block named Configuration Memory. Finally, the model assumes the existence of a Reconfigurable Region that contains a subset of configured coprocessors. This region presents data exchange and configuration interfaces to the rest of the system.

A fixed instruction set processor provides advantage in terms of code and hardware reuse, because neither the processor nor the compiler needs to be changed in the process of developing coprocessors to achieve performance and functionality goals.

The CC handles coprocessor selection and dismiss procedures produced by the processor. When selection is executed, the configuration memory is accessed and a coprocessor bitstream is sent to the configuration interface.

In order to evaluate the FiPRE model to implement RCSs embedded systems, an example case study, named R82R was designed and implemented. The system was implemented in a single VirtexII device, with the exception of the Configuration Memory. The case study employed a soft core processor customized for the FiPRE model. The changes made in the original processor were to add specific instructions to support reconfiguration (Table 1), and a specific external interface to the reconfigurable region and to the configuration controller.

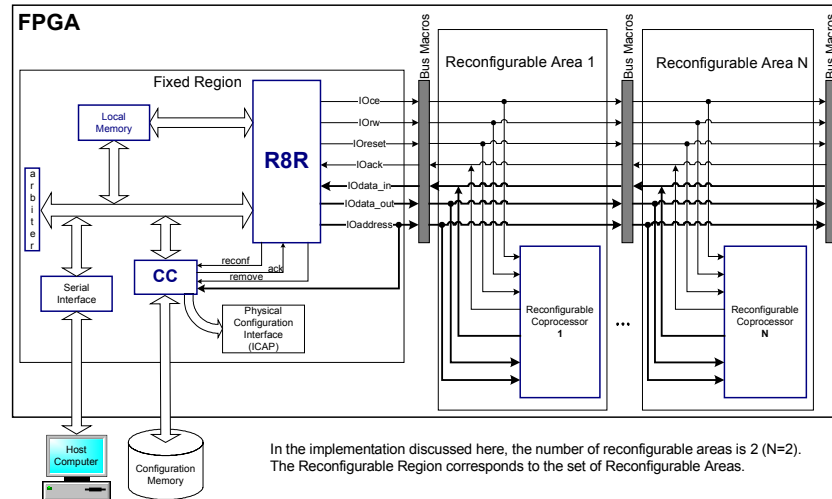


Fig. 1. General structure of the R8NR system.

Fig. 1 displays the organization of the R82R system. The system is composed by three main modules: a host computer, providing an interface with the system user; a configuration memory, containing all partial bitstreams used during system execution; the FPGA, containing fixed and reconfigurable regions of the R82R system. The fixed part in the FPGA comprises the R8R processor [4], its local memory, containing instructions and data, a system bus controlled by an arbiter, and peripherals (serial interface and the configuration controller). The serial interface peripheral provides capabilities for communication with the host computer (an RS232 serial interface). The CC is a specialized hardware, acting as a slave of the R8R and of the host computer, which fills the configuration memory before system execution starts.

The R8R processor was wrapped to provide communication with (i) the local memory; (ii) the system bus; (iii) the CC; (iv) the reconfigurable region. The interface to the reconfigurable areas comprises three identical sets of signals interconnected through special components furnished by the FPGA vendor, called *bus macros*.

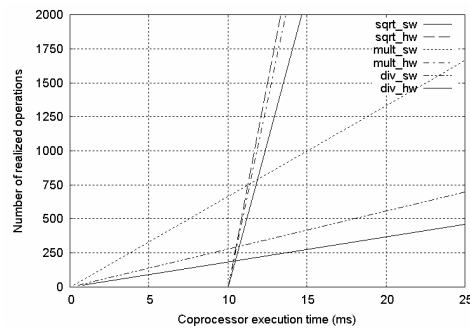
Table 1. Instructions added to the R8 processor in order to produce the R8R processor.

Reconfigurable instruction	Semantics description
SELR <i>address</i>	Selects the coprocessor identified by <i>address</i> for communication with the processor, using the <i>reconf</i> signal (see Fig. 1). If the coprocessor is not currently loaded into the FPGA, the CC automatically reconfigures some area of the FPGA with it.
DISR <i>address</i>	Informs the CC, using the <i>remove</i> signal (see Fig. 1), that the coprocessor specified by <i>address</i> is dismissed and can be removed if needed.
INTR <i>address</i>	Resets the coprocessor specified by <i>address</i> , using the <i>loreset</i> signal. The coprocessor must have been previously configured.
WRR <i>RS1 RS2</i>	Sends the data stored in <i>RS1</i> and <i>RS2</i> to the coprocessor selected by the last SELR instruction. <i>RS1</i> can be used for passing a command while <i>RS2</i> passes data.
RDR <i>RS RT</i>	Sends the data stored in <i>RS</i> to the coprocessor (typically a command or an address). Next, reads data from the coprocessor, storing it into the <i>RT</i> register.

3. Results

The R8NR system has been prototyped and is operational in two versions, with one and two reconfigurable areas, respectively (R81R and R82R). A V2MB1000 board from Insight-Memec was employed in the prototyping process.

Fig. 2 shows the comparison between the number of operations executed and the time to execute hardware and software versions of three 16/32-bit arithmetic modules: multiplication, division and square root. Note that for each module, the execution time grows linearly but with different slopes for software and hardware implementations. Also, the reconfiguration time adds a fixed latency (~10 ms) to the hardware implementations. The fixed latency is an approximation of the time measured to configure one FPGA area dedicated to hold one coprocessor.



The hardware reconfiguration latency, 10ms, is dependent on the size of the reconfigurable area partial bitstream and on the performance of the CC module. This graph was plotted for a CC working at 24MHz frequency and for reconfigurable bitstreams of 46Kbytes, corresponding to a reconfigurable coprocessor with an area of roughly one tenth of the employed million-gate device.

Fig. 2. Execution time versus the number of operations for three arithmetic modules, multiplication, division and square root, implemented in HW (hw suffix) and SW (sw suffix).

The break even point for each functionality determines when a given hardware implementation starts to be advantageous with regard to a plain software implementation, based on the number of times this hardware is employed before it is reconfigured. From the graph, it can be seen that the multiplier, division and square root coprocessors are advantageous starting from 750, 260 and 200 executions without an intervening reconfiguration step. Consider the application of a filter (e.g. edge or smooth) over an image with 800x600 pixels. If only one operation is applied per pixel 480000 operations are executed, easily justifying the use of a hardware coprocessor. This simple experiment highlights how in practice it is possible to take advantage of RCSs, achieving performance gains, flexibility and system area savings.

The R82R case study was synthesized using Leonardo Spectrum. The area report data for the fixed modules is presented in Table 2.

Table 2. Area report data for a XC2V1000 FPGA and for 0.35 μm ASIC CMOS technology.

Module	ASIC Gates	LUTs	FFs	%LUTs
R82R	6331	1020	555	9.96
Memory	3139	307	366	2.99
Serial Interface	5430	616	607	6.01
CC	2790	493	218	4.81
Arbiter	157	27	15	0.26
Total	17847	2443	1761	23.85

Configuration controllers (CC) found in the literature are mostly software implementations. The CC proposed here was implemented in hardware, having a small area footprint (around 3,000 gates) and is expected to present superior performance over software versions in terms of reconfiguration speed. Another important advantage to implement the CC in hardware is that the embedded processor is free to execute tasks in parallel during the reconfiguration process.

The Modular Design method [5] employed for generating partial bitstreams, limits the size of a reconfigurable area to a minimum of 4 CLB FPGA columns. One minimum size reconfigurable area contains 1280 LUTs (each column contains 320 LUTs). Nevertheless, the implemented coprocessors use in average 140 LUTs. Therefore, it is possible to implement much larger coprocessors in these areas. Examples are simple dedicated processors, FFT operators, and image filters.

4. Conclusions

The major contribution of the present work is the FiPre model for RCSs. An advantage of the model is the parallelism between processor and coprocessors, enabling the use of non-blocking operations. Also, the compiler does not need to be changed when a new coprocessor is added. On the other hand, an increased latency in communication may be observed, since the system parts are loosely coupled.

Also, since RCSs are reconfigurable systems, they potentially reduce the final system cost, as the user can employ smaller configurable devices, downloading partial bitstreams on demand. In addition, partial system reconfiguration makes it possible to benefit from a virtual hardware approach, in the same manner that present day computers benefit from the use of virtual memory.

Application areas for RCSs are another crucial point. Unless sound applications are developed to show real advantage of using RCSs over conventional solutions, such as RISPs and ASIPs, RCSs will remain no more than an academic exercise on an interesting subject area. Ongoing work includes performance measurement of benchmarks and improvements on the configuration controller to reduce the time wasted during partial reconfiguration.

5. References

- [1] F. Barat; R. Lauwereins. Reconfigurable instruction set processors: a survey. In: Rapid System Prototyping (RSP'00), pp.168-173, 2000.
- [2] D. Robinson; P. Lysaght. Modeling and Synthesis of Configuration Controllers for Dynamically Reconfigurable Logic Systems using the DCS CAD Framework. In: 9th Field-Programmable Logic and Applications (FPL'99), 1999.
- [3] J. Henkel. Closing the SoC Design Gap. IEEE Computer, vol 36(9), pp. 119-121, 2003.
- [4] F. Moraes and N. Calazans. R8 Processor Architecture and Organization Specification and Design Guidelines. 2003. http://www.inf.pucre.br/~gaph/Projects/R8/public/R8_arq_spec_eng.pdf
- [5] Xilinx, Inc. Two Flows for Partial Reconfiguration: Module Based or Difference Based. Application Note XAPP290, Version 1.1. 2003.