Teaching Embedded Systems with FPGAs Throughout a Computer Science Course

Vanderlei Bonato[1] Ricardo Menotti[1]
Eduardo Simões[1]
[1]Universidade de São Paulo
ICMC-Dep. Computação
São Carlos-SP-Brazil
emarques@icmc.usp.br

Marcio M. Fernandes[2] Eduardo Marques[1]

[2]Universidade Metodista de Piracicaba
FCMNTI-Ciência da Computação
Piracicaba-SP-Brazil
mmfernan@unimep.br

Abstract

Although embedded systems have been around for quite a long time, just in recent years they have attracted major industry and academic interest. There is a perception that a computing paradigm shift is taking place, and so the need to provide computer science students with the required expertise in the field. In this paper we describe our experience of using a reconfigurable computing platform throughout a number of courses. By doing so we allow students to get acquired to embedded systems concepts and practices under different contexts in the normal curriculum. The application of this strategy have allowed considerable gains for students taking embedded system courses, research projects in the field, and also professional activities.

1 Introduction

In recent years embedded computing has emerged as the new paradigm for the design and implementation of modern computer systems. They consist in the fastest growing market share for computing products, already accounting for the largest number of systems being deployed [15]. In terms of total revenue, they should also overtake desktop-based systems in just a few years. Typical examples of embedded systems include digital cameras, mobile phones, automotive control devices, and medical equipment, among others.

As it happens during any technology shift period, *skills shortage* can be a problem as the current curriculum may not address the whole set of issues involved. The range of skills required for embedded systems design encompass knowledge about hardware devices, computer architecture, microprocessor, and high-level language programming, among others [17]. Although these topics are adequately taught in Computer Science and Engineering courses, students tend to see them as isolated units, with little relation to embedded systems. We believe that reconfigurable computing [6] can be used as a platform for teaching all of those subjects, and also to expose students to some of the main concepts and practices in that field.

In this paper we describe our experience on how teaching computer architecture and related courses using reconfigurable computing allows students for a better understanding of key concepts involved in embedded systems design and implementation. The remaining sections discuss related technologies and concepts (2), the platform and tools employed by our courses (3), and how key concepts related to embedded systems are inferred from those disciplines (4). That section also describes how this strategy paves the way for research projects based on embedded systems. Finally, the last section (5) brings the conclusions of our experience and future directions.

2 Embedded Systems and Reconfigurable Computing

An *embedded system* can be described in general terms as an application specific system implemented using a programmable processor, usually integrated with other hardware devices such as sensors and actuators [4]. As opposed to desktop-based systems, embedded systems are designed to deliver the expected functionality and performance for one (or just a few) task, usually being part of a larger system. Key characteristics often required in such systems include real-time constraints, low-power consumption, and low-production cost. An important design decision is the so called hardware-software partitioning, defining the tasks that will be executed by an ASIC (Application Specific Integrated Circuit), or a software programmable microprocessor. The latter approach can be done using off-the-shelf devices, typically a microcontroller, or developing a SoC (System-on-Chip), with reusable IP (Intellectual Property) components. It should be noticed that the semiconductor industry is shifitting towards SoCs, as pointed out by the latest version of the International Technology Roadmap for Semiconductors [9].

An alternative to those approaches is to use *reconfigurable computing* [6], a technology based on reprogrammable integrated circuits, nowadays commonly

known as FPGAs (Field Programmable Gate Arrays). FPGA based systems allow performance levels comparable to those based on ASICs, but with advantages of on-site reprogrammabilty, and a shorter development cycle. Those are key attributes to deal with changing requirements, and time-to-market pressures, respectively.

In the addition, there are some evidences that the transistor count for FPGAs may be experiencing a higher growth rate than microprocessors, being comparable to the one observed for memory logic. As a result, at some point it may be possible to build FPGA based systems that are more complex than microprocessors. An evidence of this trend has been recently produced by Xilinx, with the announcement that in the near future it will make available devices topping the one billion transistor mark [19]. It should also be noticed, however, that FPGAs are still considerably slower and consume more power than ASICs, which still prevent their use in some scenarios.

3 Teaching Platform

.

As already said, key concepts and practices for embedded systems design are often taught in a number of courses with little relation, at least from the student's perspective. For this reason, we have adopted an *FPGA based platform* as a core component for those disciplines, in a attempt to encourage a "think embedded" attitude among students. The use of an FPGA platform to teach computer architecture and related disciplines has been adopted by many courses, sometimes employing supporting tools specially designed for that (e.g. [16]). However, to the best of our knowledge, it has not been adopted with the specific goal of enabling students to practice embedded systems concepts *within* other courses.

The choice for a reconfigurable computing platform, as opposed to other alternatives, was mainly due to the following reasons:

- It does not constrain the student to a particular microprocessor architecture or simulation tool;
- It accommodates a wide range of complexity levels, from simple logic blocks to a complete SoC implementation;
- It allows students to get acquired with EDA (Electronic Design Automation) and high level programming language tools.

The adopted teaching platform is based on Altera FPGA development boards, in particular the **UP2** and **Nios Stratix Edition**. The UP2 [3] is a low cost board composed of reconfigurable hardware, with a capacity of 70K gates. It also includes I/O devices and output monitoring capabilities.

The Nios Stratix Edition [1] is more sophisticated, consisting of a FPGA with 1M gates, RAM and Flash

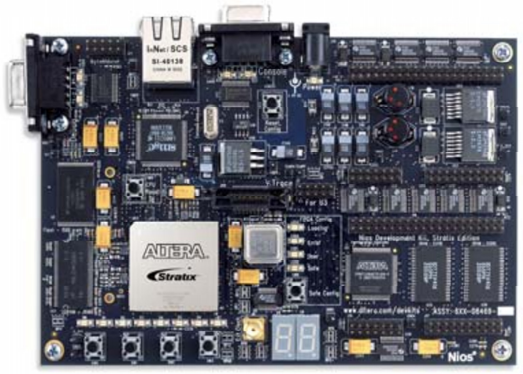


Figure 1: Kit Nios Stratix Edition

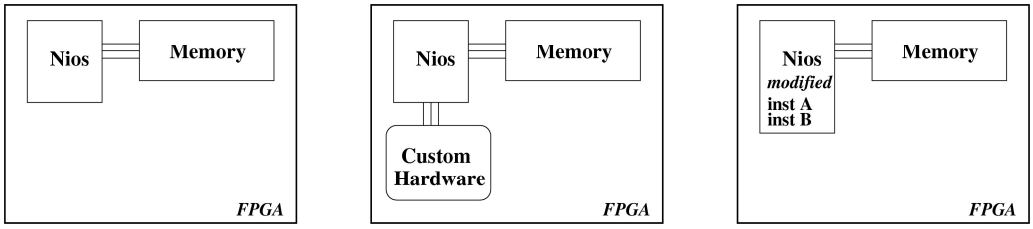
memory modules, and support for Ethernet and RS232 communication (Figure 1). The board also comes with Nios, a 32-bit RISC *softcore* processor, which can be easily synthesized into the FPGA. This processor has has a five-stage pipeline, with independent buses for data and instructions, respectively. It also allows for the implementation of *custom instructions*, a desirable feature to improve the performance of time critical sections of code. It is also possible to design logic to gain access to external resources such as memory and I/O devices. All these features have proven to be valuable in the teaching process.

The **Quartus-II** EDA tool is used during all phases of an FPGA based project, either with the UP2 or the Stratix board. In these phases are included design, compilation, timing analysis, simulation, and chip configuration. Projects are organized as modules, which facilitates reusing them. Modules can be defined by a schematic design, or using hardware definition languages. The languages currently supported are Verilog, VHDL, and AHDL. Programming the Nios RISC processor is carried out with an integrated tools chain, which includes the GnuPro C compiler, an assembler, and a debugger.

For the students, working on the design of systems including *both* hardware and software implementations have shown to be a key element to tackle embedded system concepts. The teaching projects they are exposed to are previously designed, implemented and tested, which allows for the generation of template files to be used by students. By doing so, they are prevented from spending time on repetitive tasks, common to most practices, and can concentrate on the important aspects of the work.

4 Main Courses and Activities

In this section we describe the main courses of the computer science curriculum that have adopted the reconfigurable computing platform presented in Section 3. As seen in Figure 2, during those courses students can work with embedded systems under three basic architecture scenarios: a) the simplest one, consisting of a CPU and memory modules, b) using custom



a) CPU+ Memory b) CPU + Custom HW + Memory c) CPU w/ custom instructions + Memory

Figure 2: Embedded system architecture scenarios for laboratory practices.

hardware blocks to improve performance of key sections of the application, and c) modifying the softcore CPU to include custom instructions.

Considering that the basic contents of those courses are homogeneous and well know among academic staff, we concentrate only on those details that are relevant to teach embedded systems concepts and practices. Please also notice that, for the sake of generality, the actual name of those courses may differ from the ones listed in the next section. A particular feature of some of those courses is the availability of *on-line* exams, using the EDA tool they became acquired to. Students are given a set of requirements and asked to design a circuit to meet the specification. As an example, they are asked to design a circuit to generate the signals corresponding to a given vector signal. The solutions are then sent electronically to tutors, in order to be marked. A poll among approximately one hundred students have shown that over 80% of them thinks the methodology is better than traditional ones.

4.1 Digital Systems

This topic is taught in two courses, introducing students to basic elements of digital system abstractions, such as gates, flip-flops, building blocks, binary arithmetic, multiplexing circuits, and so on. The Quartus-II EDA tool presented in Section 3 is used to implement simple lab practices to gain insight on the actual structure, behaviour, and interaction of those components. More complex assignments include the design and implementation of an ULA (Logic-Arithmetic Unit), and memory units. All projects are simulated, debugged and written to an FPGA, making possible for the student to understand timing and synchronization issues.

As an example, we have an assignement consisting in the design of *bus-based* communication interface to be implemented as an FPGA SoC. In this project students are asked to create a finite state machine for the implementation of a communication protocol. This has an appropriate complextiy level for a *second course* on digital logic, allowing students to work on structures such as fifo queues, memory, registers, etc., in order to build the design shown in Figure 3. Issues related to communication networks such as bus contention, priorities, and transmission delay are also introduced, paving the way for more elaborated projects in computer network courses (Section 4.6).

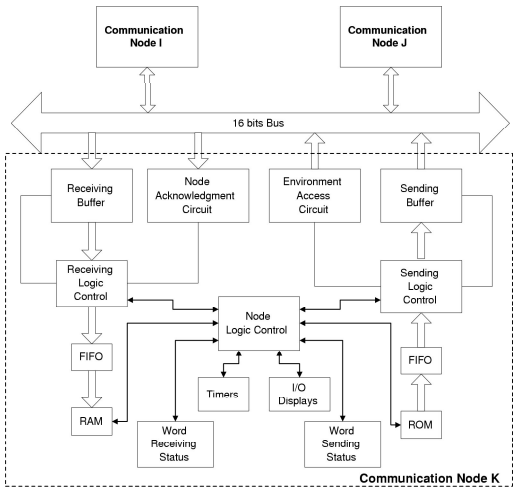


Figure 3: A bus-based communication interface.

4.2 Computer Organization

Computer Organization refers to the main units that compose a Von-Neumann machine, i.e., ALU and control unit, memory, register file, and buses [14]. The students can build a basic working system using Verilog or VHDL, starting from a simple instruction set architecture, and then expanding the ISA with other instructions. This is a particularly useful experience as the use of custom blocks (or instructions, when possible) is an effective way to execute performance sensitive tasks. Student designs are implemented and tested into the FPGA platform, which in practice gives them a feeling of a running embedded system. By doing so we allow them to practice some key concepts of embedded system design without the overhead of a new introductory course on the subject.

4.3 Computer Architecture

The theory part of this course concentrates on the usual topics such as microprocessor and pipeline organization, memory hierarchy, interconnection to I/O devices, etc. During the lab activities, students are asked to expand the CPU design of the Computer Organization course, using pipelining techniques to enhance performance. This helps to give them a better understanding on the differences that can be found between CPUs, even when they implement the same instruction set. This knowledge can be used to better eval-

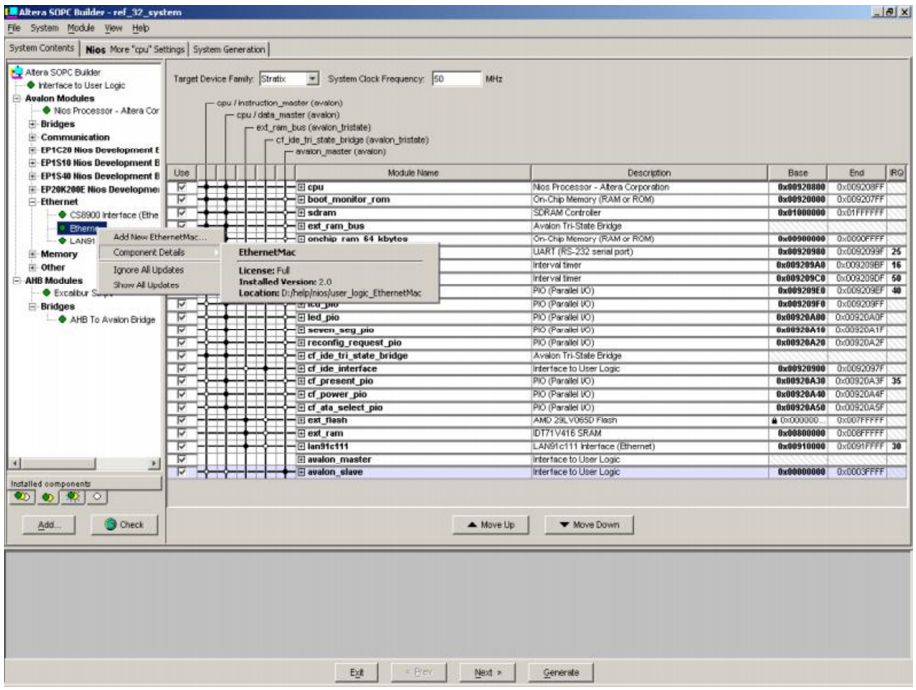


Figure 4: SoPC Builder: A tool for SoC design and implementation.

uate functionality, performance, and power consumption when choosing a CPU for a given embedded application [7].

In this course students have their first experience with the Nios softcore processor. Simple applications are implemented in C language, to be executed in the FPGA CPU implementation (Figure 2a). Then, key sections of code are implemented with custom logic, along with the additional code to switch processing and communicate data between the Nios CPU and the FPGA hardware blocks (Figure 2b). Comparing the performance results from both implementations is a good way for students to learn why hardware-software partitioning is such an important step of embedded systems design.

4.4 Compilers

The typical one-semester compiler course tend to spend more time on front-end concepts, as little time is left for back-end design and implementation. However, on a second course (usually at the graduate level) code generation and optimization techniques are the main focus, bringing compilers closer to computer architecture. We use the rich set of microprocessor architectures for domain-specific embedded applications to illustrate how code generation techniques can take advantage of that to improve performance dramatically [11]. We also use the FPGA platform for some practices aiming to create custom instructions for the Nios processor (Figure 2c), and then modify the GnuPro compiler to target those new instructions. The experience also helps students to have real evidence of a real-world aspect: that useful architecture feature may be difficult to unlock for the application code. Evaluating the quality of the *tools chain* can be as important as doing so for the architecture itself.

4.5 Operating Systems

Introductory operating systems courses tend to be general, not biased towards domain specific concepts. However, we do try to relate aspects such as interrupts, concurrency, scheduling, I/O and the device drivers to the lab platform students are getting used to. Some simple practices are designed to show the effects of not having an embedded operating systems running on the background, which is usually taken for granted on the desktop environment they are more familiar with. This experience may help students to understand what to look in the multitude of embedded operating systems available [10]. Those practices also help to introduce a new concept to them: code size, which can grow very quickly with the addition of their custom "operating system". We are currently working on a Nios port for *eCos* (embedded Configurable operating system), an open-source, configurable O.S. for embedded systems [8]. Once it is done we will be able to work with more elaborated practices, and also research projects.

4.6 Computer Networks

Computer network courses comes in all shapes and flavours depending on the course orientation. Some of them concentrates on high level abstractions, being the Internet an ubiquitous example. Others concentrate on fundamental aspects, such as the OSI model or wireless protocols. Laboratory practices varies according to the chosen emphasis, and whenever possible we offer a choice of practices on an embedded system sce-

nario. One example is a project aiming to implement a network connection between two FPGA based SoCs, letting for the student the use and customization of the required protocols. That can be made on top of existing implementations, such as an Ethernet core described in VHDL. A related research project under development refers to the integration of ethMac [13], an Ethernet MAC (Media Access Control) core designed for implementation of CSMA/CD LAN in accordance with the IEEE 802.3 standards. The ethMac Verilog code can be integrated with a Nios CPU and other softcore devices, to create a complete SoC with the required functionality.

4.7 Embedded Systems Design

We offer specefic courses on embedded systems design at both, undergraduate and graduate level. Students enrolling in the introductory undergraduate course clearly benefit from the previous experience of developing several small projects on a embedded platform. By doing so we can concentrate on the real issues of embedded systems design such as CPU architecture, and coding more sophisticated applications using high level languages [4]. It should be noticed that some analysts estimate that the *software* of embedded systems account for 80% of the *total cost* of development [9]. We give special attention to the later as there is a clear shift in embedded system design from low-level assembly implementations to the use of C or C++ language. That is not only due to productivity reasons, but also due to the emergence of complex architectures, such as VLIW [18], that can only be fully exploited by using optimizing compilers specifically targeted to them [11].

The introductory course uses the FPGA platform described in Section 3, and also DSP microcontrollers, such as the Motorola DSP56800 family. The graduate courses concentrate on SoC design, using the Altera SoPC Builder (Figure 4), a tool specially aimed at the design and implementation of SoCs on programmable chips [2]. Projects are defined according to the application areas of our research programs, which includes robotics, computer architecture, control and automation, among others.

5 Interaction with Research

As already stated, the increasing capabilities of the hardware and software currently employed for embedded systems design also results in a growing interest from academic research initiatives. That can be the case in either basic research, or applications. In our department we have projects in both areas. As an example of research on basic aspects of embedded systems, we have a project called Architect-R. Its aim is to build a tool for automatic generation of hardware and software components to implement systems for robotics [12]. Research on specific applications include implementations of multimodal interfaces, such as voice and gestures recognition systems. These can be used in a number of domains such as robotics, virtual reality, etc. As an example, the system shown in Figure 5 consists of a CMOS camera connected to an FPGA, which implements a RAM-based neural network to recognize hand gestures [5]. This embedded system shown to be robust, is able to meet real-time constraints (processing rate of 30 frames per second), and has a high efficiency in the recognition process. In addition, it also has on-chip training capabilities, a desirable functionality enabled by the reconfigurable hardware.

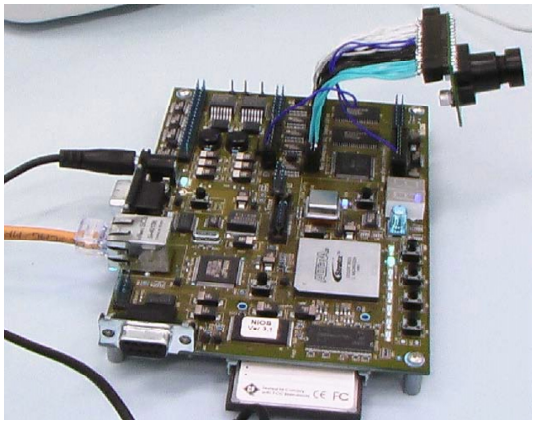


Figure 5: An embedded gesture recognition system

We have been following a number embedded systems research projects (Master's level) in our institution. Some analysis allow us to conclude that students that have been through those courses described in Section 4 are considerably more comfortable to work in this area than those that have not (typically coming from other institutions). Obviously a successful research project depend on other factors as well, but the learning curve to tackle basic concepts and practices of embedded systems is clearly reduced by using the approach described in this paper. We believe that graduates going to industry are also better equipped to start solving problems in the field. That has been confirmed to us when receiving some informal feedback from former students.

6 Conclusions

We have described our experience of teaching embedded systems to undergraduate and graduate students using a reconfigurable computing platform (FPGAs). Our goal was to devise a strategy to shift (or at least balance) the emphasis from desktop based to embedded computing, but without overloading the current curriculum. After some years of developing and applying the methodology, we understand that satisfactory results have been achieved. That is based on the increased interest from students to follow research projects related to embedded systems, the improved

performance of graduates in this domain, and also some feedback from former students working in the industry. We are still working to improve practice assignments, and also the coordination with the theoretical contents of those courses, trying to accomodate teaching priorities.

References

[1] Altera Corp. Nios Development Kit, Stratix Edition, May 2004. http://www.altera.com/products/devkits/altera/kit-nios_1S10.html.

[2] Altera Corp. SOPC Builder, May 2004. http://www.altera.com/products/software/system-/products/sopc/.

[3] Altera Corp. UP2 Design Laboratory Kit, May 2004. http://www.altera.com/education/univ/kits/unv-kits.html.

[4] A. Berger. *Embedded Systems Design: An Introduction to Process, Tools, and Techniques*. CPM Books, USA, 2002.

[5] V. Bonato, E. Simes, M. M. Fernandes, and E. Marques. A gesture recognition system for mobile robots. In *Proceedings of ICINCO-1st International Conference on Informatics Automation, Control, and Robotics*, Lisbon, Portugal, 2004 (to appear).

[6] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.

[7] S. Cotofana, S. Wong, and S. Vassiliadis. Embedded processors: Characteristics and trends. In *Proceedings of the 2001 ASCI Conference*, Netherlands, 2001.

[8] eCos. eCos Home Page, May 2004. http://eCos.sourceware.org/.

[9] D. Edenfeld, A. B. Kahng, M. Rodgers, and Y. Zorian. 2003 technology roadmap for semiconductors. *Computer*, 37(1):47–56, Jan. 2004.

[10] L. F. Friedrich, J. Stankovic, M. Humphrey, M. Marley, and J. H. Jr. A survey of configurable, component-based operating systems for embedded applications. *IEEE Micro*, 21(3):54–68, May/June 2001.

[11] J. Glossner, J. Moreno, M. Moudgill, J. Derby, E. Hokenek, D. Meltzer, U. Shvadron, and M. Ware. Trends in compilable dsp architecture. In *Proceedings of The 2000 IEEE Workshop on Signal Processing Systems*, USA, 2000.

[12] R. Gonçalves, J. Cardoso, M. Fernandes, E. Marques, et al. Architect-R: A system for reconfigurable robots design. In *Proceedings of SAC-2003 -The 18th Annual ACM Symposium on Applied Computing*, USA, 2003.

[13] Opencores.org. Ethernet mac 10/100 mbps:overview, May 2004. http://www.opencores.org/projects.cgi/web/-ethmac/overview/.

[14] D. Patterson and J. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers, Inc., USA, 1997.

[15] B. R. Rau and M. Schlansker. Embedded computing: New directions in architecture and automation. Technical Report HPL-2000-115, Hewlett Packard Laboratories, Sept. 29 2000.

[16] C. Teuscher, J. O. Haenni, F. J. Gomez, H. F. Restrepo, and E. Sanchez. A tool for teaching and research on computer architecture and reconfigurable systems. In *Proceedings of the 25th Euromicro Conference*, volume 1, pages 343–350, Milan, Italy, September 8–10 1999. IEEE Computer Society, Los Alamitos, CA.

[17] W. Wolf and J. Madsen. Embedded systems education for the future. *Proceedings of the IEEE*, 88(1):23–30, Jan. 2000.

[18] K. Wong and N. Topham. OneDSP: A unifying DSP architecture for systems-on-a-chip. In *Proceedings of ICASSP-2002 - International Conference on Acoustics Speech and Signal Processing*, USA, 2002.

[19] Xilinx, Inc. Press release no. 03131, May 2004. http://www.xilinx.com/prs_rls/silicon_vir/-03131_nextgen.htm.