New Challenges in Computer Science Education

João M. P. Cardoso[1,2]
[1] Faculty of Sciences and Technology, University of Algarve,
Campus de Gambelas, 8000 – 117 Faro, Portugal, [2] INESC-ID, Lisbon, Portugal
jmpc@acm.org

ABSTRACT
It is predicted that by the year 2010, 90% of the overall program code developed will be for embedded computing systems. This fact requires urgent changes in the organization of the current computer science curriculums, as advocated by a number of academics. The changes will help students deal with the idiosyncrasies of embedded systems, which requires knowledge about the computation engine, its energy consumption model, performance, interfaced artifacts, reconfigurable hardware programming, etc. This paper discusses some important issues to be included in modern computer science programs, in order to prepare students to be able to program future embedded computers. In particular, we present an approach we are attempting to implement at our institution. We also illustrate infrastructures that permit students to implement complex examples and gain deep knowledge about the topics being taught. Finally, with this paper we hope to foment a fruitful discussion on those issues.

Categories and Subject Descriptors
K.3 [**Computers and Education**]: Computer & Information Science Education — *Curriculum*

General Terms
Management, Documentation, Standardization, Design.

Keywords
Embedded Systems, Reconfigurable Hardware, Computing, Computer Science Education, Curriculum

1. INTRODUCTION
Most future computing systems will be embedded systems. Such systems will integrate hardware and software components and require developers with skills in both subjects [30]. Based on current trends we believe that by the year 2010 90% of the software developed will be for embedded systems [1,11]. Therefore it is necessary to adapt current computer science courses to deal with the issues of developing those systems. Recent books, such as [25, 28, 20, 16], introduce some of the needed introductory concepts in a mixed fashion. However, most curriculums limit teaching of embedded system concepts to one or two semester courses, optional or obligatory.

ITiCSE'05, June 27–29, 2005, Monte de Caparica, Portugal.

The successful use of reconfigurable hardware to implement more efficiently parts of an application drives the desire for using it in embedded computing systems [11, 31]. But, how many of our computer science students are able to program a Field-Programmable Gate Array (FPGA)?

Reiner Hartenstein, a prestigious IEEE Fellow, claims that we are teaching computer science students to employ half of their mind! [10] His point is related to the fact that we continue to teach computer science students to develop software - mainly using the procedural approach and with the von Neumann computational model, where flow of instructions performs the needed sequence of operations - without also focusing on how to program reconfigurable hardware (called the gap between procedural versus structural mentality) [11]. When data stream processing is required, suitable processing engines become increasingly important. Reconfigurable hardware tends to be an important component of such engines and therefore developers are needed to program both software and hardware components. System-on-a-chip (SOC) solutions will be the target platforms, including one or multiple microprocessor systems and reconfigurable hardware. Since programming hardware requires a different way of thinking than developing software, it is desirable that computer science degrees add more emphasis on teaching how to program systems with both components.

The emergence of embedded systems as the dominant approach has not produced corresponding changes in most computer science programs. Some programs treat embedded computing as traditional microprocessor systems. Wayne Wolf, from Princeton University, claims correctly that "An embedded computing course is not (at least in my view) a traditional microprocessor-based systems design course." [29]. Wolf et al [30] have already advocated the need to focus more on analysis and design of systems (with strong focuses on concurrent systems) than on discussing components. One of the key points is that embedded systems designers must have knowledge of the complete design process in order to make efficient design choices [30].

Pervasive computing [21], ubiquitous systems, wearable computing, and ambient intelligence are all new topics that will play an important role in future computing systems. These topics are all somewhat related to embedded systems. Smaller computing devices, such as Personal Digital Assistants (PDAs), tend to have the potentialities of recent PCs but with severe restrictions in memory size and energy consumption. How many of our computer science students know that the way a particular application was programmed has impact on the time we can use a PDA running the application without recharging the battery? How many know that a *scrollbar* in a PDA's GUI increases energy consumption [27]?

Embedded systems pose several needs related to networking, device interfacing, real-time response, reliability, time-to-market, fast deployment, fault-tolerance, multiprocessing, heterogeneity, mixed-signal interface, memory size constraints, etc. Despite the importance of the previous issues, hardware design using hardware

description languages (HDLs) and FPGAs is one of the most difficult challenges for typical computer science programs. Due to its importance, reduction of energy consumption [13] is also a major topic to be considered.

Recent editions of some conferences have included special panels on the skills needed for developing embedded systems. One of the recent panels, entitled "Embedded systems education: How to teach the required skills?", took place during the 2004 edition of one of the most prestigious hardware/software codesign conferences, sponsored by IEEE, ACM, and IFIP [2]. Beside other important aspects, the panelists unanimously stated that Embedded System Design is not well represented in academic programs.

The Computing Curricula 2001 of the ACM, IEEE joint task force does not provide much detail about embedded systems [3]. The Computer Science volume, from 2001, briefly introduces embedded system topics in *computer architecture* and *operating systems* courses. An embedded systems course is included in the "Body of Knowledge" of the Computer Engineering volume [23] draft from 2004.

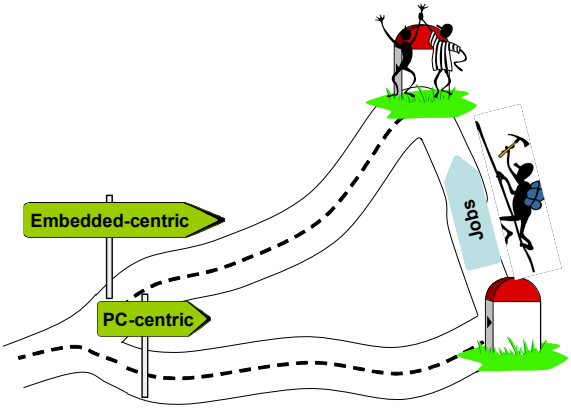

Figure 1. Computer Science education roads: one taking the PC-centric approach, which is the currently used in most computer science programs; the other taking the embedded computing centric approach, which intends to provide the skills required to deal with embedded computer systems.

Figure 1 illustrates the possible roads taken by two distinct computer science programs. The PC-centric (desktop computing) approach might not lead easily to the skills needed for future embedded computing jobs. The embedded computer centric road would guide the student to the needed skills. Graduates of the PC-centric approach will have to take by themselves another road that certainly may require a long time to acquire the appropriate skills. The opposite path, i.e., from the embedded computer approach to skills required for a typical computer science job, requires less time and effort.

This paper is organized as follows. The next two sections explain the importance of software and reconfigurable hardware, respectively. Section 4 argues about the importance of developing systems with both components. Section 5 discusses how a cheap board can be used through a number of courses to implement typical embedded computing systems. Section 6 presents some of the changes needed in traditional computer science curriculums and finally, the last section concludes the paper.

2. SOFTWARE

Software is present in most embedded systems. There are many reasons for this. Software is easily implemented by microprocessors, which are less costly solutions than the use of Application Specific Circuits (ASICs). The re-programmability of microprocessors is one of several aspects that justifies their use.

Clifton [8] has proposed a course focusing on software development for embedded systems. The approach bears in mind the underlining hardware and computer science students are also invited to make some component connections. Although the approach strongly enhances the student's perspective on a number of embedded systems idiosyncrasies, it does not, in my opinion, provide sufficient expertise for the development of future embedded systems.

Computer science curriculums use imperative and procedural programming languages, but mostly focusing software programming rather than system programming and specification. Specification languages used in some contexts, such as Esterel, SDL, and SystemC, may also be introduced. They may be important for developing and programming some embedded systems. These languages allow students to deal with specification problems not normally met when using a typical software programming language, such as Java, C, or C++. Aspects related with the problems that may be faced when selecting a software programming language are also important for dealing with future computing systems. Deep knowledge about the pros and cons of using C, Java, or C++ are also important to acquire [26].

Although the performance of a given algorithm using the target microprocessor is often taught to students, aspects related to the energy consumption of software are usually neglected. Reduction of energy consumption is an important embedded system goal and it is now being considered in ways never thought of before [27]. It is important that students understand some of the code transformations that can be used to improve performance and/or to reduce energy consumption. Even if they are not included in the introductory compiler course, concepts such as loop tilling, loop distribution, unroll-and-jam, and loop fusion should be taught somewhere. Even if such code transformations are not taught since they can be automatically generated by compiler, students should still know how to apply them.

Knowledge about how energy consumption can be modeled at instruction levels and how a particular software program can be transformed to reduce the overall energy consumption are very important. Web-based frameworks similar to JouleTrack [22] and the one presented in [15] can be used for lab experiments. JouleTrack provides estimations of energy consumption for the StrongARM SA-1100, an advanced low-power RISC processor, given an input C-program and the chosen clock frequency of the microprocessor. The tool outputs energy statistics and the estimated latency to execute the program as well.

Knowledge about the effect of the operating mode state of some microprocessors on energy consumption (see, as an example, the StrongARM SA-1100) and how those states can be used by an operating system are also important topics to teach.

It is obvious that the introduction of software development with platforms such as PDAs and cell phones, for instance in human-machine interface courses, is of great importance (see, for instance,

[6] for an approach). Courses on *networks* and *distributed systems* can also focus on some aspects of embedded systems.

3. RECONFIGURABLE HARDWARE

Hardware is becoming an important player due to its softening [24]. Reconfigurable hardware has blurred the gap between hardware and software programming. However, knowledge about digital systems and hardware description languages (HDLs) is needed to efficiently program reconfigurable hardware. For an unknowledgeable computer science student an HDL, VHDL or Verilog program appears to be written in an exotic programming language!

Digital systems design using an HDL and targeting FPGAs is being introduced in some computer science programs. Such concepts do not require deep skills in electronics. Thus, we do not need to add more courses on electronics than the usual ones present in most computer science programs (usually on circuits and systems) for students to acquire the needed skills to program FPGAs. Previous experience shows that a one semester course can lead to good results and provide the needed knowledge [4, 5, 7].

University programs provided by companies such as Xilinx and Altera permit the use of state-of-the-art commercial tools to program FPGAs. Simulators are also freely available for students (e.g., ModelSim from Mentor Graphics), although they suffer from some restrictions. Even with restrictions on code size, the use of free simulators permits students to study designs of suitable complexity. Students can also download versions of the tools and use them at home to do homework assignments and to acquire more expertise.

With respect to energy consumption, students should understand some of the techniques used to reduce power dissipation such as voltage scaling, clock frequency decrease, etc. Those concepts can be introduced in traditional *computer architecture* courses. Simple models of area, power, and delay can be taught in a week or two in introductory courses as is advocated by [9].

4. HARDWARE-SOFTWARE

The ability to program systems with reconfigurable hardware and software components is increasingly important. Codesign disciplines attempt to integrate a concurrent design of such systems. The development may have to partition the input program code through the system components in order to satisfy system constraints (e.g., energy consumption, performance, etc.). Knowledge of techniques to assign tasks of an application to the heterogeneous components of the target system, which, apart from reconfigurable hardware components, can be different microprocessors or DSPs, is also very important.

Since we are not close to automating the tasks related to partitioning, a programmer of these systems must be knowledgeable about programming software, reconfigurable hardware, and the interface and communication schemes between both.

The integration of both components is neglected or superficially taught in most curriculums. A one-semester course on embedded systems can accommodate topics related to reconfigurable hardware/software codesign, and experiments in labs to increase student's sensibility and expertise.

5. USE OF FPGA BOARDS

The use of FPGAs is now a suitable approach for introducing many concepts in labs and allowing students to implement real systems. This is due to their programmability, the availability of Intellectual Property (IP) cores for easy interface to external devices, and the availability of softcores that program part of the FPGA resources as a microprocessor or DSP. A typical board (see example in Figure 2), costing about 100 dollars may be used through a number of courses. It can be used to prototype simple digital systems in introductory *digital design* courses, to implement a microprocessor in the *computer architecture* course and in other more advanced courses such as *hardware-software codesign*, *embedded systems*, etc.

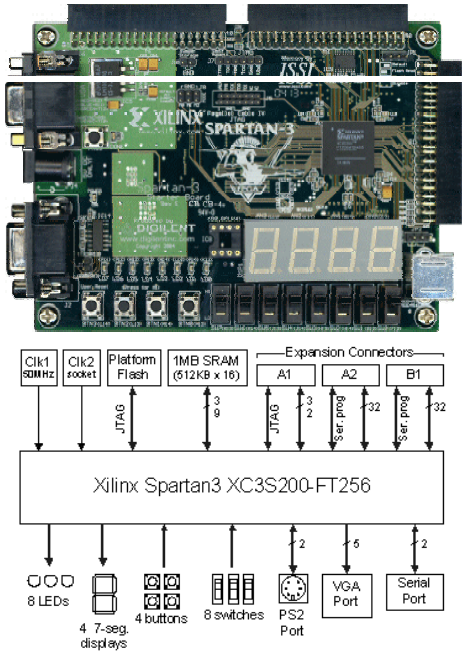

Figure 2. Digilent Spartan-3 board. Picture of the board on the top and its block diagram on the bottom.

As an example, the board in Figure 2 includes a Xilinx Spartan-3 FPGA, with a capacity of about 200,000 system gates, on-chip block and distributed memories, and embedded multipliers. The board also includes buttons, switches, LEDs, 7-segment displays, VGA, RS-232, and PS2 ports, SDRAM, and expansion connectors. Those artifacts can be used in both simple and complex designs. The expansion connectors can be used to interface to other artifacts presented in typical embedded systems (e.g., sensors, actuators, displays, etc.).

IP cores to implement specific hardware modules (e.g., PS2, USB, VGA interfaces) are available for free (see [12], as an example) and permit instructors to focus on reuse and interesting, realistic implementations that help motivate students.

FPGA companies provide RISC *softcores* that can be used in some FPGA devices. Examples of such *softcores* are the Xilinx's MicroBlaze™ [17] and the Altera's Nios™ [19]. Software kits can be donated by these companies for teaching purposes and thus furnish an economical environment to implement complex exam-

ples in advanced courses. This approach also permits exploitation of multiprocessor systems by instantiating more than one *softcore* in a single FPGA.

In advanced courses, the same board can be used to implement more complex systems. It can be interfaced to external input/output devices (e.g., to a keyboard and to a VGA monitor). Some examples of applications are image filters, control systems, MP3 players, hand-gesture recognition interfaces, etc. They permit students to start with software-centric solutions and to evaluate the use of specific architectures to accelerate some of the tasks of those applications. Examples of this complexity are being implemented in our institution as a semester project in the final year of the undergraduate degree.

6. DISCUSSION

Apart from minor changes related to the skills needed by recent technologies and concepts, no substantial changes have been performed in a decade or more in most computer science programs. With the widespread growth of embedded systems, changes to computer science curriculums should be discussed.

Most of the suggested changes in this paper can be achieved through two new one-semester courses (*embedded systems* and *advanced digital design*) plus some minor changes to four existent courses - *operating systems*, *computer architecture*, *programming languages*, and *system modeling and analysis*. Figure 3 illustrates some of the topics required and the dependences between courses. These changes may be facilitated by reducing some of the traditional topics. For instance, the assembly programming taught in some *computer architecture* courses can be greatly reduced. As another example, a reduction on CPU development details taught in *computer architecture* courses, to accommodate system concepts, is advocated in [9].

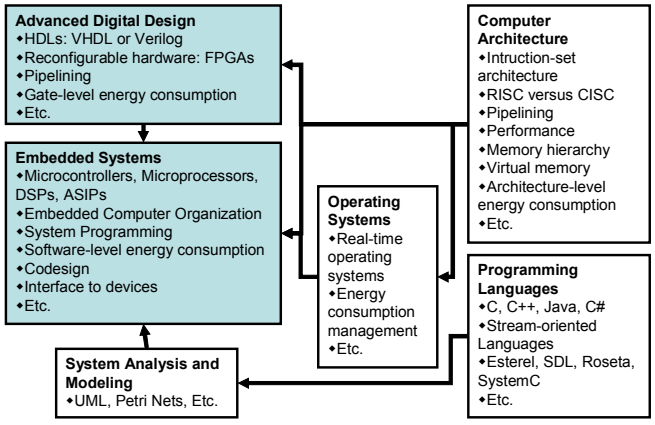

Figure 3. Main courses to support the issues presented in this paper. Note, however, that the relation of these courses to other computer science courses is omitted.

Topics such as energy consumption and performance can be included in a number of courses in different ways: in a *computer architecture* course show the techniques used by state-of-the-art microprocessors to reduce energy consumption; in a *digital design* course show the reductions of energy consumption at lower levels; in an *operating system* course show the ways to use the operating system to reduce energy consumption; and in an *embedded systems* course show how reduction can be performed at the algorithmic level.

An *embedded systems* course can also deal with hardware-software integration and complement the necessary knowledge not taught in *computer architecture*, *operating systems*, and *digital design* courses.

It is now commonly accepted that digital design using a hardware description language and targeting FPGAs does not require any courses on electronics beyond those currently available in most computer science programs. Therefore, the inclusion of these topics does not morph a computer science degree into an electrical or computer engineering degree.

Although requiring significant changes to curriculums, some of the ideas presented in this paper have been implemented in a branch of a recent computer science program. We anticipate that complete implementation is not easy, mostly because it is difficult to get faculty to agree to modify or remove courses that have been in use for the last two decades, even though the courses are no longer relevant for the current situation.

As a special note we think that the introduction of the proposed topics can also contribute to the bottom-up strategy advocated by Knuth [14].

7. CONCLUDING REMARKS

This paper intends to alert modern computer science programs to some issues requiring more attention. It explains how important it is to adapt computer science courses to address skills that will be required to design and build future embedded computing systems.

The broad knowledge needed in embedded computing systems creates difficult challenges and requires discussion and debate. The challenges are not being deeply discussed by the computer science community as confirmed by the fact that recent editions of the SIGCSE technical symposium on computer science education and ITiCSE have included issues about teaching robotics but lacked discussions about the skills needed to program future embedded computer systems.

In our opinion, it is time to think about all the enumerated issues in order to provide the skills required by current and future computer science students: 2010 is just around the corner! It is time to think about an embedded computer-centric approach as was recently accomplished by ACM and IEEE with the net-centric approach.

8. ACKNOWLEDGMENTS

We acknowledge the support from Xilinx, by donating software tools and boards, and the support of Faculty of Sciences and Technology of the University of Algarve. Their support has been important to enhance, experiment, and implement some ideas presented in this paper. The author is also in debt to Eduardo Marques for all the interesting discussions and suggestions about teaching hardware using FPGAs, and to Daniel Joyce for helping to improve the English of this manuscript.

9. REFERENCES

[1] ____, Department of Trade and Industry (DTI), London, UK, 2001.

[2] 2nd IEEE/ACM/IFIP Int'l Conference on Hardware/Software Codesign and System Synthesis, Stockholm, Sweden, Sept. 8-10, 2004. http://www.codesign-symposium.org/

[3] ACM/IEEE-CS Joint Curriculum Task Force, *Computing Curricula 2001*, 2001. http://www.computer.org/education/cc2001/; http://www.sigcse.org/cc2001/

[4] Amaral, J., Berube, P., and Mehta, P. Teaching Digital Design to Computing Science Students in a Single Academic Term, in *IEEE Transactions on Education*, vol. 48, no. 1, February 2005, pp. 127-132.

[5] Bonato, V., et al., Teaching Embedded Systems with FPGAs Throughout a Computer Science Course, in *Workshop on Computer Architecture Education (WCAE 2004)*, June 19, 2004, at *31st Int'l Symposium on Computer Architecture*, Munich, Germany, June 19-23, 2004, pp. 8-14.

[6] Burge, M. Pervasive computing in the undergraduate curriculum, panel in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia, USA, March 3-7, 2004, ACM Press, pp. 181-182.

[7] Calazans, N., and Moraes, F. Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses, in *IEEE Transactions on Education*, vol. 44, no. 2, May 2001, pp. 109–119.

[8] Clifton, J. A CS/SE Approach to a Real-Time Embedded Systems Software Development Course, In *Proc. of the 32nd SIGCSE Technical Symposium on Computer Science Education*, February 21-25, 2001, pp. 278-281.

[9] Dally, W. The Case for Broader Computer Architecture Education, keynote in *Workshop on Computer Architecture Education (WCAE'04)*, June 19, 2004, at *31st Int'l Symposium on Computer Architecture (ISCA)*, Munich, Germany, June 19-23, 2004.

[10] Hartenstein, R. The Changing Role of Computer Architecture Education within CS Curricula, invited talk in *Workshop on Computer Architecture Education (WCAE'04)*, June 19, 2004, at *31st Int'l Symposium on Computer Architecture*, Munich, Germany, June 19-23, 2004. Presentation slides: http://xputers.informatik.uni-kl.de/staff/hartenstein/lot/HartensteinWCAE04.

[11] Hartenstein, R. The Digital Divide of Computing, in *Proceedings of the ACM Int'l Conference on Computing Frontiers (CF'04)*, April 14–16, 2004, Ischia, Italy, ACM Press, pp. 357-362.

[12] http://www.opencores.org

[13] Jacome, M., and Ramachandran, A. Power Aware Embedded Computing, in *The Industrial Information Technology Handbook*, Richard Zurawski, Ed., CRC Press, 2005, pp. 1-23.

[14] Knuth, D. Bottom-up education, keynote in *Proceedings of 8th annual conference on Innovation and technology in computer science education (ITiCSE'03)*, Thessaloniki, Greece, 2003, ACM Press, pp. 2-2.

[15] Lee, I., et al. A web-based energy exploration tool for embedded systems, in *IEEE Design and Test of Computers*, (2004: to appear).

[16] Marwedel, P. *Embedded System Design*, Kluwer Academic Publisher, 2003.

[17] *MicroBlaze Hardware Reference Guide*, Xilinx Corporation, http://www.xilinx.com, 2004.

[18] Newman, K., Hamblen, J., and Hall, T. An introductory digital design course using a low-cost autonomous robot, in *IEEE Transactions on Education*, vol. 45, no. 3, August 2002, pp. 289–296.

[19] *Nios Embedded Processor User's Guide*, Altera Corporation, http://www.altera.com, 2004.

[20] Patt, Y., and Patel, S. *Introduction to Computing Systems: from bits & gates to C & beyond*, McGrawHill Press, 2001.

[21] Saha, D., and Mukherjee, A. Pervasive computing: a paradigm for the 21st century, in *IEEE Computer*, Volume 36, Issue 3, March 2003, pp. 25-31.

[22] Sinha, A., and Chandrakasan, A. Jouletrack - a web based tool for software energy profiling, in *Proc. of ACM/IEEE Design Automation Conference (DAC'01)*, June 2001, pp. 220–225. http://carlsberg.mit.edu/jouletrack/JouleTrack/index.html

[23] Soldan, D., et al., Computing curriculum - computer engineering, *IEEE - ACM*, 2004. http://www.eng.auburn.edu/ece/CCCE/CCCE-IronDraft-2004June8.pdf

[24] Vahid, F. The softening of hardware, in *IEEE Computer*, Volume 36, Issue 4, April 2003, pp. 27-34.

[25] Vahid, F., and Givargis, T. *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley and Sons, October 2001.

[26] Vallerio, K., and Jha, N. Language selection for mobile systems: Java, C, or Both?, in *Proc. Int'l Conf. Embedded Systems and Applications (ESA'04)*, June 2004, pp. 185-191.

[27] Vallerio, K., Zhong, L., and Jha, N. Energy-efficient graphical user interface design, in *Proceedings of the Int'l Conference on Pervasive Computing and Communications, (PCC'04),* Las Vegas, Nevada, USA June 21-24, 2004,. CSREA Press 2004, pp. 959-962.

[28] Wolf, W. *Computers as Components: Principles of Embedded Computing Systems Design*, Morgan Kaufmann Publishers, 2000.

[29] Wolf, W. What and why about architecture for embedded systems, keynote address in *Workshop on Computer Architecture Education (WCAE'00), at 27th Int'l Symposium on Computer Architecture*, Vancouver, Canada, June 10, 2000.

[30] Wolf, W., and Madsen, J. Embedded systems education for the future, in *Proceedings of the IEEE*, Vol. 88, No. 1, January 2000, pp. 23-30.

[31] Wong, S., Vassiliadis, S., and Cotofana, S. *Embedded Processors: Characteristics and Trends*, Technical Report CE-TR-2004-03, Delft, The Netherlands, May 2004.