TEACHING COMPUTER ORGANIZATION AND ARCHITECTURE WITH HANDS-ON EXPERIENCE

Ney Laert Vilar Calazans[1], Fernando Gehm Moraes[2], César Augusto Missio Marcon[3]

Abstract *- This work describes part of a novel approach employed at the authors' institution in the last five years, which comprises the teaching of computer organization/ architecture through the effective implementation of processors and computers. The context of the courses is presented first, including a comparison of two hardware courses tracks in Computer Science and Computer Engineering curricula. Previous publications have described the structure of courses dealing with the minimal implementation of a working processor. Here, the emphasis is on subsequent courses, which take the minimal implementation and guide the students through the necessary steps to add performance, such as pipelining, and functionality, such as memory management and basic IO subsystems.*

Index Terms - Computer Organization and Architecture Teaching Methods, Undergraduate Curriculum, Hardware, Digital System Prototyping

INTRODUCTION

Computer organization is defined as the discipline that studies the computer while an electronic apparatus, whilst computer architecture is the discipline that studies the computer as the abstract machine defined by the organization. Then, computer organization is often seen as the electronics engineer view of a computer, while computer architecture is often thought as the assembly programmer view of the computer. In practice, separating computer organization from architecture is a hard and useless task.

A deep understanding of computer organization and architecture is mandatory for the more technical Information Technology (IT) degrees such as Computer Science and Computer Engineering. However, traditional curricula often rely upon a dichotomy between theory and practice of computer construction. Examples are the curricula where computer organization and architecture courses are based on excellent books like that of Patterson and Hennessy [1].

The authors of the present work have proposed and implemented a teaching approach that is based on the integration of theory and practice of computer construction [2][3]. This approach differs in several aspects from the traditional one. It dictates that students should learn how computers work not only by studying their inner details, but also by concomitantly building processors and computers or embedded systems. Students are exposed the earliest the possible to computer construction activities, typically starting at the third period of academic activities. Consequently, the computer organization and architecture courses must rely strongly upon lab courses and/or lab activities.

Two relatively recent technological advances allow the new approach to become a reality in the classroom. First, there is the availability of cheap, powerful hardware prototyping platforms based on reconfigurable hardware such as FPGAs and CPLDs. A good example of the profusion of available hardware aids is the list maintained by Guccione [4]. Next comes the existence of easy to use, powerful, free and/or commercial computer aided design tools for high-level design entry, validation and implementation. Examples of these tools are the current simulators and synthesizers based on Hardware Description Languages (HDLs).

The traditional approach is not devoid of practical aspects though. Many such courses employ assembly language tools, conveying the view of computer architecture intricacies through the use of assemblers, architecture simulators, compilers or even assembly programming of real life processors [1]. Others advocate the use of organization simulators to clarify concepts such as pipelining or cache control [5]. However, none of these conduct the students through the process of building an original processor from scratch. In the authors' view, this is the best way to teach computer organization and architecture so that the acquired knowledge persists longer and the interface between hardware and software becomes absolutely clear.

The novel approach has also its pitfalls. At least two restrictions can be stated against it. First, learning modern techniques and tools employed in building computers takes long. Next, what is constructed in the context of undergraduate courses is necessarily far from what is available as state of the art processors or computers. A discussion and an assessment of the students' opinion pointed out that the new approach is nonetheless a rather good one [3].

This work describes part of this novel approach to teach computer organization and architecture through the analysis, simulation, design and effective construction of processors. Previous work, described in [2] and [3] have focused on the first steps of the approach, where a minimal implementation of a processor is addressed. The emphasis of this paper is on subsequent steps, which enhance the efficiency of the

[1] Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Porto Alegre - BRAZIL - 90619-900 calazans@inf.pucrs.br
[2] moraes@inf.pucrs.br
[3] marcon@inf.pucrs.br

processor and add functionality to it to build a minimal computer implementation. The next Section describes the context and the structure of the implemented courses. Follows a Section that discusses a course where the students are expected to improve the processor performance, followed by a Section about a course on building a minimal computer implementation using the enhanced processor. The set of tools employed to enable the approach in the two courses is discussed in a subsequent Section, and the text ends by presenting a set of conclusions and future work.

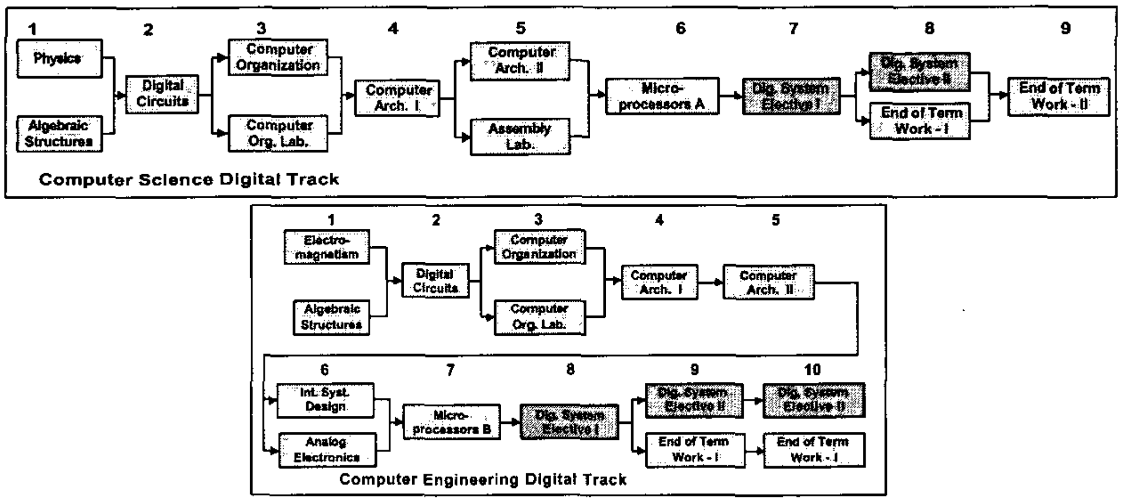

FIGURE 1
COMPUTER SCIENCE AND COMPUTER ENGINEERING DIGITAL HARDWARE TRACK OF COURSES. ARROWS INDICATE PREREQUISITES.

COURSES CONTEXT AND STRUCTURE

The approach proposed here has been adopted since 1997 in a Computer Science curriculum. Its success led the authors to apply it to a newly proposed Computer Engineering curriculum, with strong emphasis in automation, either industrial automation or other kinds such as home and office automation. The structure of the digital hardware track of courses for both curricula appears in Figure 1 and is the subject of this Section.

In both curricula there is a basic core destined to provide the main concepts of computer organization and architecture. This core is essentially the same and corresponds to the five first semesters of each curriculum. The minor differences of the courses in this period arise from the differing needs for the two kinds of professionals to be formed. Computer scientists need a stronger basis in programming, justifying a dedicated course to assembly language programming. On the other hand, the physics courses in the first semester are different although both cover the same subjects namely electricity and electromagnetic phenomena. Computer engineers need a deeper knowledge of such subjects, justifying a more demanding course.

The prerequisites for the main courses on semesters 3 to 5 are an introduction to Digital Circuits, a course on Algebraic Structures and another on Physics. These courses provide the student with traditional combinational and sequential logic design techniques, lattice and Boolean algebra theory, and a brief account of circuits, electronics, electromagnetic phenomena and instrumentation, respectively. Another required course in both curricula is Microprocessors, which again are different in each curriculum. In Computer Science, the Microprocessor A course is an overview of state of the art processors and their characteristics. Important aspects of this course are the distinction between the so-called generic processors and digital signal processors, and the comparative discussion of modern architectures such as Intel Pentium, Power-PC, MIPS and SPARC, among others. On the other hand, the Microprocessor B course in Computer Engineering reinforces the use of microprocessors as embedded processors, to account for the automation emphasis of the curriculum. This means that input-output issues, reactive and real-time processing are stressed, justifying also the use of the Analog Electronics course as a prerequisite.

In both curricula, it is possible to take elective courses on selected advanced topics on digital systems. Examples of such topics include programmable logic, electronic design automation, embedded systems, microelectronics and so on.

Thus, in both cases, the student is exposed during the whole curriculum to digital hardware issues regarding his/her future profession. This is one of the objectives of this teaching proposal. The next paragraphs of this Section present an account of the specific courses on computer organization and architecture.

The computer organization teaching was implemented as two required courses, a 4-hour a week course on computer organization and a 2hour a week laboratory course both taught in the 3rd semester and are more extensively discussed in [2] and [3]. The lecture course comprises a study of Central Processing Unit (CPU) classical models, comparing the stored program computer (or von Neumann model) to the Harvard model and an introduction to assembly language by means of a practical educational processor. Also included as part of the teaching approach is an introduction to a hardware description language (HDL), currently VHDL [6]. This last characteristic enables the end of course work to be realized, i.e. the construction of a simulateable description of a load-store processor from scratch in the learned HDL. The companion lab course puts into practice all contents of the lecture course, using both classical schematics based design and HDL design paradigms, allied to real hardware implementations through the use of FPGA-based fast prototyping platforms.

The Computer Architecture I and II courses are the main subject of this work and are discussed in detail next.

The Computer Architecture I course contents are distributed into five units:

- *Unit 1*: Computer architecture performance evaluation - where the basics for quantitatively comparing architectures are introduced, including the notion of speed-up, throughput and standard *de facto* benchmarks like SPEC.
- *Unit 2*: Pipelines, general structure, control and construction – here, the main performance enhancement technique for architectures is explored in detail, paving the way to the end of course final work.
- *Unit 3*: Advanced computational arithmetic – explores something more than the basic arithmetic structures approached in digital circuits course, including integer multiplication and division, arithmetic operations on rational numbers and the study of the IEEE754 floating point standard representation.
- *Unit 4*: Programming and program execution support systems – this Unit addresses the main issues related to how a written high-level language program is processed before it is ready to execution by the processor, including compiling, assembly, linking and loading procedures.
- *Unit 5*: The relationship between architecture and high-level language programming – where the hardware-software interface is finally explored, using the high-level language to assembly translation process to clarify the relationship stated in the title of the Unit.

On the other hand, the Computer Architecture II course is divided into three parts:

- *Unit 1*: Input and output subsystems - where the students familiarize with the more important aspects of making the processor communicating with the external world. This includes the discussion of classifications of the interactions between processor and the rest of the world (polling, interruption and DMA), and the study of several devices and device interfaces. These range from simple, low speed communication such as serial and parallel, to highly demanding interfaces such as video and fast secondary storage.
- *Unit 2*: Memory subsystems – in this Unit, a quantitative assessment for the need of using memory hierarchies in modern high-performance architectures is explored, introducing the associated concepts of software and hardware such as cache memory structure and levels, translation look-aside buffers (TLBs) and virtual memory.
- *Unit 3*: Architectures for parallel processing – an introduction to the vast world of parallel processing is the subject of this Unit, going from ancient classifications like the one of Flynn, to modern concepts such as that of cluster computing. During this overview, parallel processing paradigms, hardware for parallel programming and software issues and tools are presented.

It is easy to see that, unlike the lecture contents of the computer organization courses, the computer architecture lectures are rather conventional when compared to other modern approaches. The difference between the traditional approach and that proposed here is in the way the technology and case studies taught and used in computer organization are reused to provide the practice of the concepts in the subsequent courses.

It should also be clear that Computer Architecture I as Computer Organization courses before it are restricted to discuss the CPU inner workings, while Computer Architecture II deals with what should be added to a processor to build a complete computer. Accordingly, the next two Sections cover the practical aspects of Computer Architecture I and II courses, respectively.

COMPUTER ARCHITECTURE AND PROCESSOR IMPLEMENTATION

There is no doubt that the most pervasive single aspect that affects the structure of modern processor architectures is the pipelining of instructions. The instruction sets of these processors reflect the choice of pipelining strategies, and the control unit and datapath organizations are directly affected by these choices. Performance issues dictated a major change on the processor design paradigm in the 80's, creating the RISC (Reduced Instruction Set Computer) concept, to transcend the problems created by previous machines, from that time on named CISC machines, standing for Complex Instruction Set Computers. The acronym today is outdated, since modern RISC processors have not a reduced instruction set at all, but it is still widely used. The main characteristics of the first RISC machines still remain however, i.e. pipelining and a load-store organization. The load-store denomination means that every instruction that references a memory position can only either *read the contents of this position into an internal register* or storing a single datum in a memory position, precluding complex memory operations.

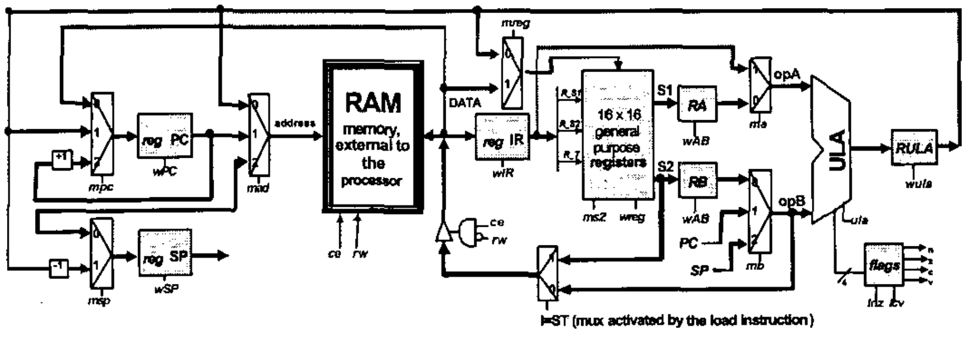

FIGURE. 2
R8 RT LEVEL DATAPATHES EXPLORED IN THE COURSE. THE REGISTERS ARE INSTRUCTION REGISTER (IR), PROGRAM COUNTER (PC), STACK POINTER (SP), AND A REGISTER FILE WITH 16 GENERAL-PURPOSE REGISTERS. THERE ARE 18 CONTROL SIGNALS THAT COMPOSE THE MICROINSTRUCTION WORD GENERATED BY THE CONTROL UNIT OF THE PROCESSOR (NOT SHOWN).

The previous paragraph provides a justification for the hands-on experiment proposed to students of the Computer Architecture I course. Given the previous experience acquired by students with an HDL and its use in implementing a simple load-store processor in Computer Organization courses, they are required to turn that processor *into a pipeline one. Every semester, a* brand new architecture is proposed in the organization course, and in the subsequent course this architecture is to be turn into a pipeline processor. The architecture is named *Rx*, where *x* is the number of the proposed architecture. The students have the choice to pick their own previous semester implementation as starting point, or to use a fully functional implementation provided by the instructors.

The basic organization of a typical processor as specified to the students appears in Figure 2. This architecture is always a load-store, von Neumann, multi-*cycle instruction architecture*. The work is accordingly divided into three steps. First, transform the von Neumann implementation into a Harvard machine, so that no structural hazards are present. Then, the students must "pipeline" the processor, equalizing the number of cycles of all instructions, introducing registers to store intermediate results between the pipeline stages and changing the control unit to make the whole a working processor. Finally, comes the step where the students increment the architecture to provide conflict detection and resolution capacity with hardware support. Data hazards must be detected and solved *if possible or a bubble must be* inserted in the pipeline. Control hazards must be detected and bubbles must be inserted to solve them. Optionally, strategies of branch prediction may be implemented and employed to increase performance in the resolution of control hazards. This ungrateful task may provide the students with an extra bonus if any speed-up is achieved with regard to the basic bubble insertion technique.

The students are given about 50 days to complete the practical work, and the specification is handled to them as soon as the lecture course enters Unit 2, about pipelining. The learning of the pipeline concepts is then simultaneously exercised in hardware design. The students are required to employ the learned HDL and they must also show speed-up computations for every step of the pipelined processor design with regard to the initial non-pipeline implementation. Example simulations showing the original and final performance of the architecture at *the timing level* for assembly language programs is shown in Figure 3.

Along the eight first editions of the Computer Architecture I course, the proposed processor and practical work have evolved from a simple pre-fetch architecture to a full-fledged 5-stage pipeline processor. The structure of the practical work is now mature, and it is no surprise that it resembles the MIPS I architecture used in consecrated books like [1]. Figure 2 presents an intermediate version processor, with an expected solution presenting a 4-stage pipeline. One interesting lesson learned by instructors and students during the work with the R8 is why it is useful to use 5-stage pipelines instead of 4-stage ones. The reason is not immediately clear without conducting an implementation work. The choice of a 4-stage pipeline implies the need to add another output port to the register file memory bank to allow execution of memory store instructions, a rather expensive hardware increase because of a single instruction. Also, the same experiment showed why a special register like a stack pointer could be quite expensive in hardware, pointing to the usefulness of emulating stack pointers in assembly language instead.

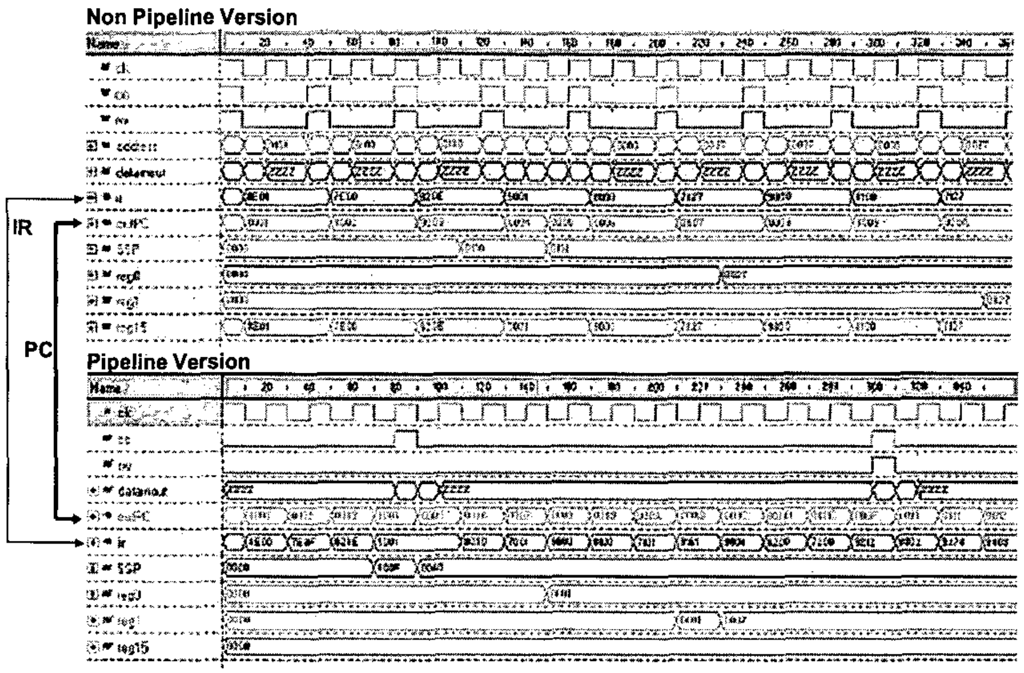

FIGURE. 3
PARTIAL FUNCTIONAL SIMULATIONS OF AN OBJECT CODE PROGRAM IN AN RX PROCESSOR. OBSERVE THE IR AND PC VALUES. IN THE NON-PIPELINE VERSION TWO CLOCK CYCLES ARE NECESSARY TO CHANGE THE PC VALUE, WHILE IN THE PIPELINED VERSION ONLY ONE CYCLE IS SUFFICIENT. THE PIPELINED VERSION FETCHES ONE INSTRUCTION PER CLOCK CYCLE.

An important limitation of the approach in the current version of this course is that students work with HDL simulators alone, without dealing directly with hardware implementations in prototyping platforms. This is not a limitation inherent of the laboratory setup, but rather a matter of content maturity. The choice of an early approach to processor design make students involve themselves very early with convoluted concepts of computer organization and architecture and there is no time left to explore the issues behind a processor implementation in hardware. Examples of problematic issues are the timing problems that need to be addressed during pipeline processor implementations, which can be overlooked in functional simulation, but not in hardware prototyping. A solution to this is currently being studied, and will involve furnishing a controlled implementation environment where the timing issues are hidden from students.

Nonetheless, the basic objective of the hands-on experience is achieved namely to allow students assess the real complexity of modern processors.

COMPUTER ARCHITECTURE AND COMPUTER IMPLEMENTATION

While the real hardware implementation of a pipeline processor in the scope of the Computer Architecture I course is not currently feasible, the contents of Computer Architecture II can easily capitalize on hardware implementations upon current prototyping platforms. This is true for several simple input-output interfaces, such as serial and or parallel communication, and even for simple audio and video interfaces. Currently, these are just planned for hands-on activities of hardware design and construction, due to the lack of trained instructors available to teach the course. However, the problem is already solved and the activities described here will be implemented for the first time in the second semester of 2002.

The solution found to provide the students with practical work involving hardware implementation is distinct from that in the previous courses, since the number of distinct subjects addressed here is much larger than there, and the subject complexity is much lower. Thus, students are expected to do several small practical works, with the objective of understanding the variety of input and output interfaces available and the justification for existing so many of them.

Practical work tasks start with the design of simple interfaces such as hardware mouse and keyboard drivers, evolving to simple audio and video drivers. This has already been tested in several editions of the elective courses in Computer Science, those depicted in Figure 1. The result is a bit surprising to students, they learn not to take for granted the complexity of such low-level tasks in hardware and are able to understand details never explained in purely theoretical courses. For example, students learn that mouse and keyboard are not just input devices, but that they may interact as needed with the CPU. Also, the capacity of video memories is finally understood more thoroughly, both in terms of storage consumption and in terms of performance gain and/or degradation.

In fact, it is the panoply of input and output hardware devices and interfaces working in harmony with the processor that makes a computer. The overall complexity of a computer system is not located in the processor alone, but is also a result of many pieces of complex hardware easing the input and output of data to and from the processor and memory subsystems. This is in essence what the practice with designing and building computer hardware elements conveys to the students.

The set of practical works on input and output modules hardware design has as goal to allow the integration with a given pipeline processor design (some version of the Rx architecture). Given a working processor it is up to the students to build a fully working computer, although still without an operating system or basic tools as compilers linkers and loaders.

One interesting and difficult part of the computer construction is the memory subsystem design, depending of course on the degree of complexity desired to approach. Even if complex subsystems such as the current three-level caches are impossible to consider, one-level caches should be enough to convey most of the issues in modern cache design. We are currently considering the acquisition or construction of reconfigurable hardware platforms that allow the hardware implementation of cache controllers to take place. These should contain at least two distinct memory types, such as DRAM and SRAM, to allow the emulation of cache and main memory systems to take place.

TOOLS

The current tools employed to make the approach proposed here viable include commercial electronic design automation (EDA) tools, commercial FPGA-based fast prototyping platforms and in-house educational software. The EDA tools employed are Xilinx Foundation for hardware design capture and synthesis (possible migration to Xilinx ISE is under consideration), Aldec´s Active-HDL simulator (migration to Menthor ModelSim is under consideration). The XESS XS40-010 plus the XST-1 input and output extension boards form the prototyping platform. Although this platform has been quite useful, its limitation to allow dealing with some computer architecture issues like cache implementation and its size limitation to allow a full-fledged computer to be implemented has led to considering other solutions. One good candidate to fulfill current and future needs of all courses is the proposition presented in [7], due to its flexibility.

CONCLUSIONS AND FURTHER WORK

Implementing processors and computers as part of the process of taking computer organization and architecture courses is an approach that has now been under application for five years, with clear advantages to the learning process. Several unclear intricacies of computers are revealed to and understood by students that design their own hardware. The use of prototyping platforms more powerful than those currently available is expected to lead to greater integration of the learning and implementation processes. The interface and integration with other subjects such as compiler construction and operating systems is a very desirable feature of the proposed method, and it may capitalize in the existence of functional nearly complete computers built along several courses.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the continued support of Xilinx, Inc. The authors are also grateful to the Brazilian support institutions CNPq and FAPERGS that through the years have financed their research work, making possible the developments reported here. The authors are also in great debt with Aline Mello, a former student of the digital hardware courses who implemented reconfigurable assembler and simulator programs for the processors used as case studies in several courses.

REFERENCES

[1] Patterson, D. A. and Hennessy, J. L. "Computer organization and design: the hardware/software interface". Morgan Kaufmann Publishers, Inc. San Mateo, CA,. 2nd Edition, 1998. 964pages.

[2] Calazans, N. L. V. and Moraes, F. G. VLSI Hardware Design by Computer Science Students: How early can they start? How far can they go? *In: 1999 FRONTIERS IN EDUCATION CONFERENCE*, San Juan. IEEE Computer Society Press, 1999. pp. 13c6-12-13c6-17.

[3] Calazans, N. L. V. and Moraes, F. G. Integrating the Teaching of Computer Organization and Architecture with Digital Hardware Design Early in Undergraduate Courses. *IEEE Transactions on Education*, Piscataway, v. 44, n. 2, pp. 109-119, 2001.

[4] Guccione, S. List of FPGA-based Computing Machines. Available at: http://www.io.com/~guccione/HW_list.html.

[5] Grünbacher, H. Teaching Computer Architecture/Organization using Simulators, In: 1998 *Frontiers in Education Conference*, Tempe, AR. Session S2C, pp. 1107-1112, November 1998. Available at: http://fairway.ecn.purdue.edu/~fie/.

[6] S. Mazor and P. Langstraat. "A guide to VHDL". Kluwer Academic Publishers. Norwell, MA, 1992.

[7] Eduardo Bezerra, Marianne Pouchet, Ney Calazans, Fernando Moraes, Michael Gough. An adaptable educational platform for engineering and IT laboratory based courses. Approved for publication at *2002 Frontiers in Education Conference*.