# Integrating the Teaching of Computer Organization and Architecture with Digital Hardware Design Early in Undergraduate Courses

Ney Laert Vilar Calazans, *Member, IEEE,* and Fernando Gehm Moraes, *Member, IEEE*

***Abstract*—This paper describes a new way to teach computer organization and architecture concepts with extensive hands-on hardware design experience very early in computer science curricula. While describing the approach, it addresses relevant questions about teaching computer organization, computer architecture and hardware design to students in computer science and related fields. The justification to concomitantly teach two often separately addressed subjects is twofold. First, to provide a better insight into the practical aspects of computer organization and architecture. Second, to allow addressing only highly abstract design levels yet achieving reasonably performing implementations, to make the integrated teaching approach feasible. The approach exposes students to many of the essential issues incurred in the analysis, simulation, design and effective implementation of processors. Although the former separation of such connected disciplines has certainly brought academic benefits in the past, some modern technologies allow capitalizing on their integration. Indeed, the new approach is enabled by the availability of two new technologies, fast hardware prototyping platforms built with reconfigurable, hardware and powerful computer-aided design tools for design entry, validation and implementation. The practical implementation of the teaching approach comprises lecture as well as laboratory courses, starting in the third semester of an undergraduate computer science curriculum. In four editions of the first two courses, most students have obtained successful processor implementations. In some cases, considerably complex applications, such as bubble sort and quick sort procedures were programed in assembly and or machine code and run at the hardware description language simulation level in the designed processors.**

***Index Terms*—Computer organization teaching methods, digital system prototyping, hardware description languages, hardware design, undergraduate curriculum, VHDL.**

## I. Introduction

THE ADVENT of the electronic computer and its popularization in the last decades as a widespread commodity brought about several changes in well-established academic sectors like Electrical Engineering, Physics, and Mathematics departments. The creation of computer science courses emerging from single or joint efforts of such departments was a first change. Presently, besides the stable status acquired by independent computer science departments, there is a significant trend to offer degrees in subjects that are intermediate between computer science and many other disciplines. These range from computer engineering, computational mathematics, and computational physics to multidisciplinary efforts unlikely to make sense just a few years ago, such as degrees mixing computers and Arts or computers and Law. Even in the least technical degrees, it is necessary to provide the student with a basic set of concepts about the hardware, the software, and the interface and relationship between them. Thus, the contents of disciplines as computer organization and computer architecture are a mandatory part of any such degree. Of course, the amount and depth of material adequate to each degree will vary widely, but the basic concepts are always present.

This work addresses the teaching of computer organization and architecture in more technical degrees, where the amount of material on hardware and on the hardware-software interaction is significantly large.

Computer architecture and organization are well-established disciplines in computer science and computer engineering curricula. Quite often, these contents are covered early in undergraduate courses, and the approach is absolutely sound. This happens because of the crucial need to provide the student soon with a strong understanding of the relationship between the physical reality of hardware and the software abstractions it implements.

On the other hand, complex digital systems design is a subject usually covered, if at all on elective courses, at the end of the undergraduate curriculum. Also, these digital systems design courses frequently restrict themselves to low-level aspects of the design process, such as the physical and logical abstraction levels.

However, computers *are* digital systems, formed either by a composition of several subsystems or even centered on a single very large scale integration (VLSI) chip. Thus, an excellent way to teach and to understand their capabilities and limitations is by effectively designing them. Two recent technological advances allow such design to become a reality in the classroom. The first is the availability of cheap, powerful VLSI hardware prototyping platforms based on reconfigurable hardware such as field-programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs) [1]. A good example of the profusion of hardware aids is the list maintained by Guccione [2]. The second advance is the availability of easy to use, powerful, free, and/or commercial computer-aided design (CAD) 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).

Manuscript received May 22, 2000; revised December 28, 2000. This work was supported by CNPq Research Grants 522939/96-1 and 520091/96-5, and FAPERGS Research Grants 94/01340-3 and 96/50369-5.

The authors are with Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, RS, Brazil 90619-900 (e-mail: calazans@inf.pucrs.br; moraes@inf.pucrs.br).

Publisher Item Identifier S 0018-9359(01)03867-5.

This work describes a novel approach to teach computer organization and architecture through the analysis, simulation, design and effective construction of processors using the aforementioned advanced facilities.

The rest of this paper is organized as follows. Section II describes the state of the art in teaching digital systems design in general and computer organization and architecture in particular. Section III justifies the approach proposed in view of the state of the art. Section IV is about the context of the implemented courses. Section V explores the courses structure and the employed teaching methods, considering the merging of digital systems design techniques into computer organization and architecture. Section VI assesses issues arising from the choices of tools and problem solving strategies. Section VII introduces preliminary results on assessing the point of view of students about the courses. Finally, Sections VIII and IX present the current state and the future of the reported work, as well as a few conclusions.

## II. State of the Art

Computer organization and architecture involves many abstract concepts for the undergraduate beginner. Today, most students start undergraduate courses having already had contact with computers as users, which was hardly true some two decades ago. In this way, they see the computer in a preconceived way, as a tool useful for tasks such as Internet browsing or text edition. This view bares few obvious relationships to Boolean algebra, logic gates, registers, or arithmetic logic units. There is then an enormous gap to fill in order to teach computer organization and architecture adequately. The gap is caused by the increasing number of abstraction layers interposed between the real hardware and end-user oriented applications. To close this gap, several teaching methods are currently being proposed. Most of them are based on extensive use of practical work to support lectures.

Traditional approaches to introduce the study of computer organization and architecture, like the excellent one proposed by Patterson and Hennessy in [3], often limit practical work to using assembly language programming and associated assembler and simulation tools. This Section reviews modern approaches that aggregate effective hardware design and sometimes hardware implementation to help understanding computer architecture abstractions and to better motivate students to further work on the field.

A frequently debated issue among those suggesting modern approaches is the use of commercial versus noncommercial (also called special-purpose or educational) tools. Several authors advocate the use of educational tools as the only way to improve teaching. The idea behind this attitude is that using commercial tools has some serious disadvantages. In [4], the authors state that commercial tools are not adequate in teaching, because they are oriented toward a professional market, because they provide many features not needed for educational purposes, and because their leaning curve is too steep, slowing down the learning of fundamental design concepts. The authors then present a specially produced simulation environment called VISCP, based on schematic capture to help students in grasping digital systems concepts more easily. Maurer [5], [6] on the other hand, proposes another environment, called winFHDL with similar characteristics but working at a more abstract design level. He suggests that simulation is more important for the designer of hardware than practice in hardware laboratories, and also proposes the extensive use of educational tools. In another work, Grünbacher [7] presents four simulators, each one oriented to help in teaching specific concepts in modern computer architectures, like pipelining, cache memory functionality, and superscalar organizations.

An opposite view to enhance computer organization and architecture courses is to directly insert the use of commercial hardware and software tools as basis for practical work. Nixon [8] proposes employing medium complexity programmable logic devices (PLDs) associated tools and proprietary languages for introducing digital systems design concepts. The author reviews and compares the most prominent tools and languages products in the area of programmable array logic (PAL) design, including PALASM, PLPL, PLACE, ABEL and MAX+PLUS II. Nixon does not recommend the use of languages as VHDL for introductory levels, because of its complexity. Hamblen and others present two required courses taken by computer engineering seniors at Georgia Institute of Technology in [9]. The prerequisite knowledge for these courses includes digital design, FPGAs, VHDL modeling and synthesis, assembly language, C programing, operating systems, and computer architecture. The practical work involves the design, implementation, and validation of a 32-bit processor and associated basic software (assembly and C compiling tools). The design tools include commercial software for Xilinx FPGAs and retargetable compilers (Lcc) and assemblers (in-house). The implementation employs a rapid prototyping platform, ZyCAD Paradigm RP, comprising 16 10 000-gate FPGAs. Validation takes place through the use of simulation and emulation based on commercial logic analyzers. Finally, the authors conclude that feedback from students has increasingly encouraged the use of commercial tools, languages, devices, and test and measurement equipment. This involves CAD tools, the VHDL language, FPGAs, and logic analyzers. A third approach using mostly commercial tools is proposed by Kleinfelder and others in [10]. The authors implemented a set of five courses starting at the fourth quarter of a Computer Engineering curriculum at the West Point Military Academy. The courses employ increasingly more complex tools as the courses progress, starting with schematics using a student edition version of a commercial tool (Workview Office), followed by an introduction to PLDs and their hardware design using a simple HDL called CUPL. The introduction of VHDL occurs at the second course of the set, Computer Organization, where it is used for the specification and implementation of a 16-bit microcontroller. The last courses of the series expect the students to design a real system to solve a real problem, including both digital and analog components.

Other works have recently introduced the need to provide integrated laboratories for enabling hands-on experience for engineering and computer science students. The main goal of these efforts is providing capacity to solve real problems that most often require the students to deal with electrical, mechanical and

other engineering problems. Examples of such laboratories are those at the Colorado University at Boulder [11] and at Auburn University [12].

Teaching hardware using extensive hands-on experience relies on the availability of hardware platforms to support these design activities. Several authors have proposed the construction of such platforms, e.g., the prototyping platform described in [13] and the FPGA-based hardware tester depicted in [14].

Most of the results obtained in the present work points to the commercial tools use approach as the most adequate. This is corroborated by the opinion of the majority of the students, as discussed in Section VII.

## III. Justification of the Approach

There are two quite different ways of approaching the design of complex digital systems at the undergraduate level—the one employed in electrical and/or electronic engineering, and that of computer science and/or computer engineering.

Electrical/electronic engineering students start with a strong emphasis in linear circuits, which are used later as the foundation to investigate electronic phenomena and devices. The study of digital circuits and systems principles and techniques comes very often late in the curriculum as a special case behavior. In this way, electrical engineers are usually strongly aware of physical consequences of designing digital circuits, like clock distribution problems and timing constraints. However, they often fail to grasp architectural and software issues of a complex digital systems design.

On the other hand, computer science/computer engineering students get acquainted with digital systems much earlier than electrical engineering students get. They employ abstract models from Boolean algebra as a basis, instead of circuit theory models. Thus, it is hard for them to deal with timing and power constraints affecting the performance or even the functionality of the whole system. However, they are usually well equipped to deal with higher levels of abstraction such as architectural and software issues of digital systems.

As mentioned in the beginning of this paper, two technological advances, powerful CAD tools and fast hardware prototyping facilities, enable teaching of computer organization through the effective implementation of processors. They do so by allowing designers to overlook most physical synthesis issues, shifting the design effort to more abstract levels. This approach is particularly well suited to computer science students, since they may then use physical synthesis as a *push-button* activity, concentrating the design effort at higher level, organizational and architectural problems.

Teaching computer organization and architecture in this way helps Computer Science students to demystify hardware design activities. Also, students are able to understand more thoroughly problems appearing in future courses, like compiler performance optimization, memory management, and input–output bottlenecks at the system and operating system level.

However, the same set of tools can most often be used with lower level occupations in mind, since they include aids for performing tasks such as IC floor planning, delay and area constraints, performance-driven placement, routing and clock distribution. This would of course be a more appropriate approach to Electrical Engineering students.

From the above discussion, it is reasonable to conclude that the ideal team to design digital systems, and particularly computers, must be a composition of both engineers and computer scientists. One problem with multidisciplinary teams is that they often fail to communicate properly, because of the different views each professional has of the same basic concepts. Another beneficial side effect of using the above teaching approaches is to bridge this communication gap, by providing common ground knowledge in these concepts to both classes of professionals. This issue is better understood by presenting a few examples of practical subjects where communication problems arise. Examples are external/internal memories and pipelining. The authors consider in the analysis only what computer science students acquire to bridge the mentioned communication gap. Several other example subjects and the electrical engineering student side of bridging the gap could be devised as well.

A computer science student often learns to view memories simply, as tables coupled with means to access their data. The approach in this work proposes a simultaneous digital systems view of memory technology. Initially, a study of memories in an abstract way is conducted, as tables. An explanation of the internal structure of each memory bit follows addressing decoding in hardware, how this decoding implements access to individual memory positions, and how the needed control signals are generated. This view of memory technology is helpful in understanding the need to address the memory bandwidth problem not only in hardware design, but also in software and, above all, operating system design. In this way, computer scientists are able to better interact with hardware designers in issues like the SRAM/DRAM tradeoff.

Pipelining is another point where communication can be a problem between hardware and software personnel. Software people usually deal with an abstract view of pipelines, as provided by the assembly language level, which resembles very little the multistage organization of modern hardware pipelines. The exercise of implementing pipelined hardware enables the computer science student to better understand the hardware issues as well as the implementation of the assembly language abstraction. This enhances his/her capacity to communicate with hardware specialists.

There is also the recent trend to provide curricula that are midway between electrical engineering and computer science such as computer engineering. The students of these latter courses should benefit from the approaches proposed above, applied respectively in early and late moments of their undergraduate academic experience. The first basic idea here, as well as in computer science curricula, is to modify the traditional early courses on computer organization and architecture to include training with the digital systems implementation of the structures needed to support the abstract view of computers in these disciplines. The second idea is to employ elective courses at the end of the curricula to deepen the training of the interested student in advanced topics of digital systems and computer abstract design. This approach is currently being

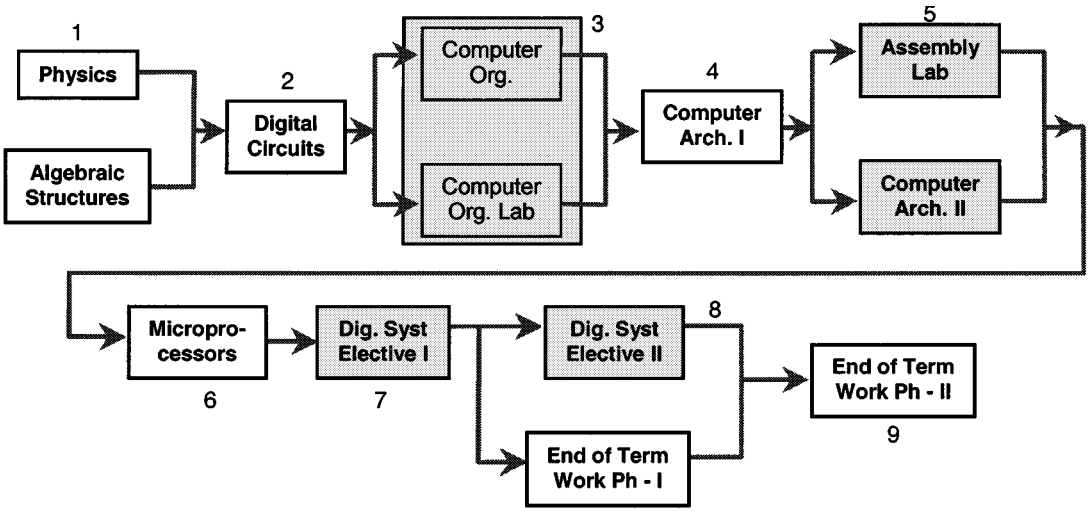



Fig. 1. Courses context. The numbers indicate the semester within the curricula where the courses take place. The arrows indicate prerequisites among courses.

applied in the computer science curriculum at the PUCRS by the authors.

As a conclusion of this Section, it is possible to state that the coupling of modern CAD tools, fast prototyping platforms, and carefully designed teaching methods is the key to justifying the mix of computer organization/architecture and (VLSI) digital systems design very early in computer science/computer engineering curricula.

## IV. Courses Context

The proposal for computer organization teaching is part of a revised computer science undergraduate nine-semester curriculum, which went into effect in the first semester of 1997 (starting in March) at the authors' institution. It was implemented as two required courses—a four-hour-a-week course on computer organization and a two-hour a week laboratory course, both taught in the third semester. The sequence of courses more closely related to hardware in the curriculum is depicted in Fig. 1

The prerequisites for both courses on computer organization are an introduction to digital circuits, a course on algebraic structures, and another on physics for computer science. These courses provide the student with traditional combinational and sequential logic design techniques, lattice and Boolean algebra theory, and a brief account of circuit and electronics theory and instrumentation, respectively. In the third semester, relying upon this background, the students learn the principles of computer organization in theory and in practice. They start working with a simple von Neumann architecture at the logic and register transfer level (RTL) levels of abstraction. In the second part of the course they return to the same subject employing higher abstraction levels of design, based on the use of hardware description languages (HDLs ). The detailed contents of these two courses are explored in Section V.

Four other required courses, two on computer architecture, one in assembly language programming, and one on microprocessors follows those on computer organization. Computer Architecture I explores computer architecture performance analysis, pipelining, basic software tools like assemblers, linkers and loaders, and the role of compilers in translating high-level language programs to assembly language. Computer Architecture II includes input–output subsystems, memory hierarchy and memory management, and principles of parallel computing. At the same time, in the fifth semester, the students are required to take a lab course where they implement a practical application (like cryptography algorithms) using assembly programming. Next, in the microprocessors course they undertake comparative studies of commercial microprocessors and microcontrollers.

This curriculum requires each student to take at least 16 credits on elective courses, corresponding roughly to four courses. These courses are taken on eight knowledge domains in computer science, namely digital systems, databases, software engineering, applied computing, artificial intelligence, computer graphics, distributed systems, and computer theory. The students can choose to take one or two of these elective courses on digital systems advanced topics. The current emphasis of the elective courses are on complex digital systems design using HDLs, designing, implementing, and validating some application on hardware prototyping platforms based on FPGAs.

The curriculum also requires students to produce a one-year End-of-Term Work. This work is conducted either individually or by groups of two students, under the advising of one academic. Students can, of course, choose some subject related to digital systems. In this way, every student is exposed at each period in the whole curriculum to hardware issues regarding his future profession. This exposure was one of the objectives of the 1997 revision.

## V. Courses Structure and Teaching Method

### A. Courses Structure

The Computer Organization (INF46183) course contents are distributed into four units. Unit 1 is entitled *The Digital Systems Design Process*. In this Unit, models for the design process like the $Y$-diagram [15] and others [16] are introduced, together with a discussion of digital systems' classifications and the basic digital circuit design flow with the use of CAD tools. Unit 2, *The CPU Classical Model*, is the place where the datapath and

control unit partitioning is explored through the presentation of both von Neumann and Harvard organization models. During this part of the course, a step by step case study design is developed at the RTL abstraction level. Unit 3, *Hardware Description Languages* (HDLs), presents another, more abstract way of designing digital systems and, in particular, processors. This includes the redesign of the CPU case study of Unit 2. Unit 4, *Advanced Topics on Computer Organization*, introduces advanced hardware structures for datapath and control unit performance enhancement, such as pipelining and floating point arithmetic hardware. At the end of the lecture course students are required to design a load-store 16-bit processor using the target HDL. The specification of this final work is delivered to students during the first HDL introductory lectures. In this way, they have 45–50 days to complete the design and present a functional HDL simulation.

The theoretical course comprises approximately 30 two-hour lectures. One-quarter of the lectures is consumed in revising digital circuits' basics and in presenting design process models and taxonomy. One-third of the lectures is used to present the Cleopatra case study, developing and discussing its schematic-based design. Another quarter of the lectures is used to introduce the basics of a specific HDL and to redesign the case study with it. The rest of the time is taken to analyze the advanced concepts in computer organization.

The Computer Organization Laboratory (INF46184) course is divided into three units. In Unit 1, entitled *Classical Digital Circuit Design*, students review basic concepts acquired in the previous digital circuits course, working with schematic capture and simulation tools. They implement basic combinational and sequential blocks such as adders, ALUs, counters, and finite state machines. Next, in Unit 2, *Classical CPU Design*, they employ the same set of tools to implement the case study of the lecture course step by step, using RTL schematic modules. Finally, Unit 3, *HDL Hardware Design* is the plan where an HDL language is introduced and used to describe hardware at the behavioral and structural levels, contrasting the complexity of the circuits they can handle with that of the previous approach. The target which is HDL chosen is VHDL [17].

The lab course comprises about 15 2-hour practical sessions. Table I shows a subset of currently available classes prepared for the students.

A subset of nine to ten of the practical sessions is picked at each semester for teaching as lab course contents. Of these, 60% are dedicated to get the students acquainted with CAD tools and to explore the schematic based approach to processor implementation. Of the remaining sessions, 30% investigate the HDL-based approach, and 10% introduce physical synthesis concepts and train the use of the available prototyping platform [18]. This platform is constructed with RAM-based FPGAs.

### *B. Teaching Method*

Both the lecture and lab courses are strongly based on the stepwise analysis of a single processor case study named Cleopatra [19]. This is in accordance with well-established practice of teaching computer organization like the exploration of the MIPS [20] architecture in [3]. The datapath for the Cleopatra processor is depicted in Fig. 2(a). Since the emphasis is on computer organization models, the case study is neither complex nor modern. It consists in a simple eight-bit, accumulator-based von Neumann core, with 13 operation codes and four addressing modes, amounting to 37 distinct instructions.

It is worth discussing the validity of choosing architectures like Cleopatra to teach the basics of modern computer organization. One reason is that this is the first contact of the students with the abstract concept of Instruction Set Architectures (ISAs), and it is necessary to build on the knowledge they acquired in the Digital Circuits course only. Furthermore, this approach gives the students a historical perspective of processor evolution. The first processor and microprocessor generations were accumulator-based machines, because of the scarcity of integration available and the consequent high cost of memory bits inside the CPU. Starting with this primitive organization, one can easily explore with the students the bottlenecks that led to the CISC processor crisis. Among these, it is possible to mention the high number of clock cycles per instruction (CPI) and the frequent access to a unified memory (the von Neumann or *stored program model*). Understanding these problems paves the way to accepting innovations like pipelining (to reduce the CPI), the Harvard model of organization, and increasing the number of internal registers (reducing the so-called von Neumann bottleneck). At the end of the course, students are faced with the need to implement a load-store architecture, and in the next course (Computer Architecture I) they are expected to implement pipelined processor organizations.

The Cleopatra core assists in introducing and discussing several new concepts in the lecture course. The specification allows exploring instruction sets, addressing modes, and principles of assembly language programming. These architectural concepts are apprehended in practice using an in-house basic software development and validation system, described in Section VI. After familiarization with the architectural concepts and assembly language programming, the proposition and the study of one datapath organization for implementing the Cleopatra processor follows. Among the concepts explored, it is worth highlighting the most important ones. First, the role and use of control and data registers is explored. Next, the interaction between the memory and the processor is studied, differentiating von Neumann and Harvard organizations. The choices incurred in the ALU implementation and the busing strategies follow, finishing with the control unit structure, comparing hardwired and microprogramed implementations.

Meanwhile, the lab course examines the digital systems design view of the computer organization in detail, using CAD tools, among other tasks, to construct increasingly more complex modules of Cleopatra.

Both lecture and lab courses start with a traditional, schematic-based approach of processor core design. Later, the whole design work is redone with modern, HDL-based tools. There are three main reasons for redoing this work. First, it shows that it is possible to design at very high abstraction levels and still obtain competitive implementations, because of the current quality of CAD tools. Second, students are provided with a comprehensive insight into the panoply of

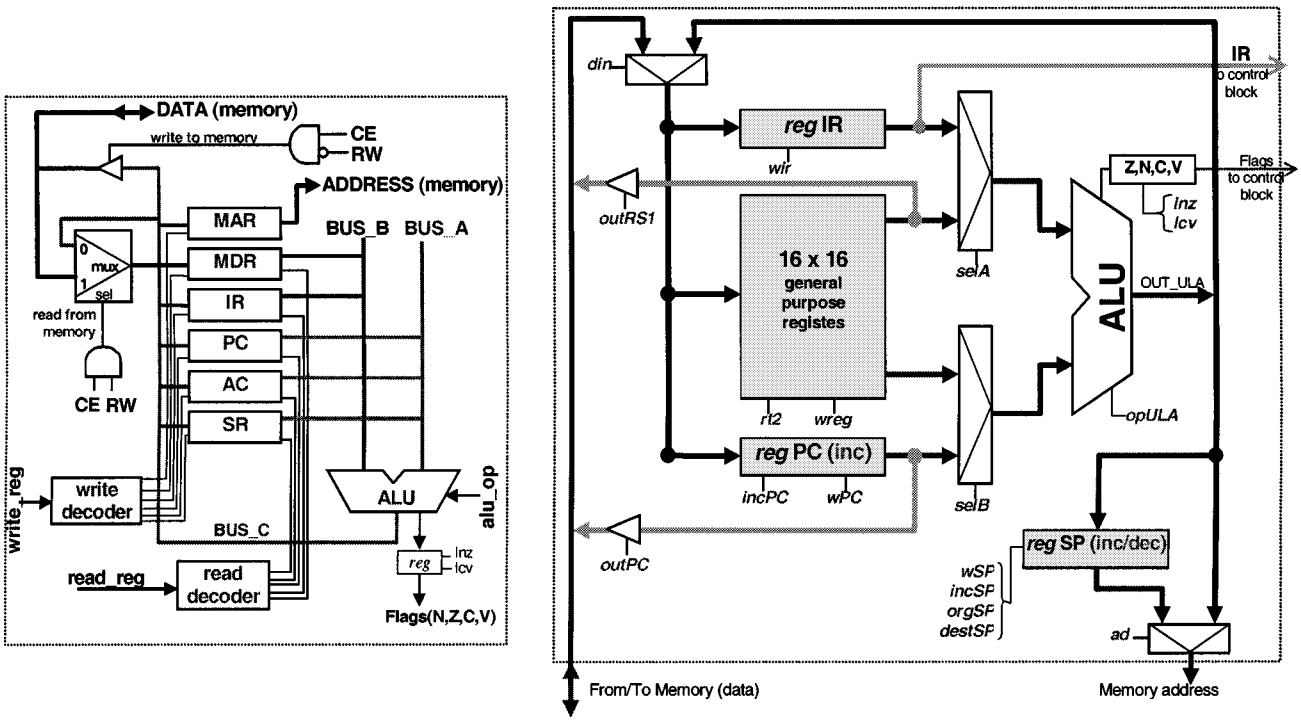
TABLE I
EXAMPLE SET OF AVAILABLE CLASSES FOR COMPUTER ORGANIZATION LABORATORY. AMONG THE 17 CLASSES, DEPENDING ON THE AVAILABLE TIME, 9 TO 10 ARE CHOSEN EACH SEMESTER (30 HOURS—15 CLASSES), DIVIDING CONTENTS EVENLY BETWEEN SCHEMATIC CAPTURE AND VHDL LAB CLASSES

| Class | Title | Goal |
|---|---|---|
| 1 | Design of an 8-bit adder (2 hours) | Schematic capture. Schematic capture editor training, use of design hierarchy and principles of functional simulation. |
| 2 | Design of a 4-bit ALU (2 hours) | Schematic capture. Implementation and functional simulation of a combinational circuit which is widely used in microprocessors. |
| 3 | Implementation of an 8-bit accumulator in an FPGA prototyping platform (4 hours) | Schematic capture. Introduces the students to the use of physical synthesis tools and to the use of a prototyping platform (XS40/Xstend). |
| 4 | Design of an up/down counter (2 hours) | Schematic capture. Practice with state machines. |
| 5 | Design of a *timer* (4 hours) | Schematic capture. Students must implement a *timer*, with minutes and seconds. They must specify the counter structure, implement the timer and test it in the prototype platform. |
| 6 | Simulation of a stack calculator (2 hours) | Schematic capture. Introduce the students to the concepts of data path and control circuits, and also the simulation of a programable circuit. |
| 7 | Cleopatra assembly language (2 hours) | Use of a *simulator* for the example architecture (Cleopatra), aiming at the practice of assembly language programing and particularly the understanding of the architecture addressing modes. |
| 8 | Cleopatra data-path (2 hours) | Schematic capture. The students should finish the design of the given data-path block, using micro-commands to excite it, by means of functional logic simulation. |
| 9 | Cleopatra control block (2 hours) | Schematic capture. The students implement the block responsible for translating assembly instructions into micro-commands addresses for a micro-ROM. |
| 10 | Use of the VHDL simulator (2 hours) | VHDL. Design of a simple VHDL example, simulating it with a test bench. This class introduces the use of the VHDL simulator. |
| 11 | Design and validation of an 8-bit multiplier (2 hours) | VHDL. Practice with several sub-circuits (VHDL entity-architecture training) in the same design. |
| 12 | Implementation of the 8-bit multiplier (2 hours) | VHDL. The students must take into account the input/output constraints imposed by the prototype board to define the multiplier external interface. The goal of this class is to implement and run the multiplier in the prototyping platform. |
| 13 | Behavioral bubble-sort in VHDL (2 hours) | VHDL. The goal of this class is to use VHDL as a programing language, implementing a sorting algorithm. |
| 14 | Structural VHDL (2 hours) | VHDL. Implementation of a second version of the bubble sort algorithm, comparing the physical results between the behavioral and structural versions. |
| 15 | Simulation of the VHDL Cleopatra version (4 hours) | VHDL. Introduce to the students to the design of complex test-benches. |
| 16 | Drink Machine (2 hours) | VHDL. Implements the classical example of the drink machine (VHDL), exploring the facilities to implement state-machines in VHDL. |
| 17 | State machine to control RAM memory access (4 hours) | VHDL. Shows how an FPGA can control an external memory. Uses state-machines. |

hardware design methods and tools, by comparing these two significantly distinct approaches. Third, it makes it clear to students that HDLs are not programing languages, but instead is the mapping of schematic symbols and structures to the syntactic and semantic structures of the chosen HDL. This last reason makes instructors insist that students must "think hardware" when using HDLs, even if the language they use allows them to view hardware as software.

Immediately after the study of the HDL starts in the lecture course, the end of course work specification is given to the students. This specification includes the instruction set architecture of a load-store processor, and the block diagram of its datapath organization. One example datapath that is provided in the last portion of the course appears in Fig. 2(b). The most important design constraint imposed by the specification is a fixed number of clock cycles per instruction (CPI $= 2$).

The last part of the lecture course (Unit 4) is dedicated to showing how concepts presented in the first three Units can be extended to reach the real structure of modern processors. In this sense, this Unit is much more a preview of what is going to be explored in the next courses than a deep or complete image of the subjects. The chosen topics are pipelining and hardware implementations of floating point arithmetic functional units. Pipelining is showed as a key concept for achieving greater performance, improving the instruction throughput, without changing the instruction latency. The IEEE-754 standard [21] serves as the basis to explore the implementation of floating-point hardware. After presenting the standard, an




(a) (b)

Fig. 2. (a) Cleopatra and (b) Ramses RT level datapaths explored in the INF46183 course.

algorithmic view of hardware multiplication and addition is explored. No real hardware implementations of these are investigated here, because of the lack of time.

At the end of the semester, students have obtained an understanding of the basic processor design flow at high levels of abstraction. They write assembly language programs that run on the machine they have implemented. Thus, students are able to develop software and follow the data and control information flow internal to the hardware.

## VI. Tools

As mentioned in Section II, the authors strongly advocate the use of commercial tools whenever possible. For didactic reasons, the processors used to present the basic computer organization and architecture concepts are not commercial ones. It would not be feasible for such an early course to do otherwise. Thus, it was necessary to develop a set of in-house tools to support the introduction of the concepts. The next Section explores these tools, while Section VI-B discusses the employed commercial tools.

### A. In-House Tools

There are two sets of in-house built development tools, one for the Cleopatra case study and another for the course final work. Both sets comprise an integrated environment with a text editor, an assembler, and a simulator for the corresponding processor. Fig. 3 shows example windows for both environments.

These environments are an invaluable support for the teaching of the ISA basic concepts, including the notions of control and data registers, addressing modes, assembly language and its relationship to object code, not to mention a deeper understanding of memory organization, variables, variable addresses (pointers), and variable contents.

Also employed is the SIS public domain academic synthesis system from Berkeley [22], for illustrating control unit logic synthesis strategies.

### B. Commercial Tools

There are four sets of commercial tools employed in the Computer Organization lecture and lab courses. All of the tools run over a Windows NT PC host computer. The most important of these is the CAD system Foundation, version 3.1i from Xilinx [23]. From this CAD system the most employed tools are schematic capture and module generators, logic simulation, and logic and physical synthesis tools. It also used the Active-HDL, version 3.5 from Aldec [24], an HDL simulation environment, which can be fully integrated with the Foundation software.

The next set of tools is a hardware prototyping platform composed by two boards, the XS40 and XStend, produced by the enterprise Xess, Inc. [18]. This is an educational platform useful for small hardware implementations (supporting up to 5000 equivalent logic gates). Attaching the boards to a PC parallel port allows the communication with the host computer to take place. The last set of tools is the support software for design download and communication between the XS40/XStend platform and the host computer.

As stated before, the first units of the Computer Organization courses rely upon the use of schematic based capture and validation tools, including the real implementation of hardware over the prototyping platform. For the last units, the authors have chosen to use VHDL as the hardware description language

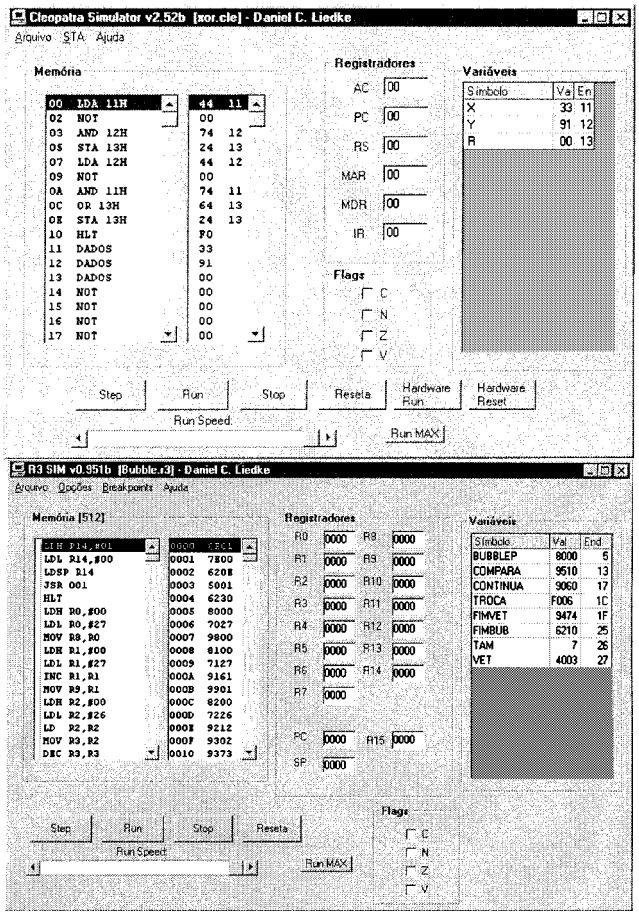



Fig. 3. Cleopatra and Ramses assembly language simulators developed for the INF46183/INF46184 courses. The tools include a built-in text editor, an assembler, and a simulator whose graphical user interface (GUI) is shown. The GUI allows controlling the execution speed, inspect and change values of registers, flags and memory. The right window exhibits the symbol table generated by the assembler, and can also be edited during the simulation.

in both courses. Again, this is not the easiest way to approach higher level abstract design entry, but they consider it worthwhile, since VHDL is not only one of the most complete, but also the most used of HDLs today. The same tools mentioned above also support other HDLs, such as Verilog and Abel.

The Active-HDL tool is used to guarantee functional validation of the VHDL code. One example of validation using this tool appears in Fig. 4, which displays a partial simulation of a program executing in the Cleopatra processor. Fig. 4 stresses the functional nature of the simulation, where no relative delay among signals is considered.

As a conclusion of this section, it is important to note that students are exposed in the courses to a complex and integrated framework for digital design. Students get used to several abstraction levels, starting with functional specifications (either in schematic and/or VHDL) and going down until the physical synthesis, implementation, and testing of real hardware prototypes.

## VII. Results

The authors have elaborated a survey to measure the students' view of several points concerning the Computer Organization courses INF46183 and INF46184. This section presents the structure of the survey, the raw data obtained after applying the survey to more than a hundred students, and a critical analysis of some aspects of the data.

### A. Population and Survey Description

The survey contains 16 questions, divided into five subjects. A first set of questions (Q1–Q3) concerns the profile data of the population attending the courses. Information collected includes the semester when they took the course(s) for the first time (Q1), the number of times the students took the course until succeeding (Q2), and taking the lecture and lab courses together or not (Q3). The first question is the main key to the classification of the data, chosen to reflect the evolution of the approach versus time. Table II shows the number of students per semester that answered the survey.

A second set of questions is dedicated to evaluate the overall quality of the courses both in themselves and compared to other courses in the Computer Science curriculum. The specific questions concern:

- Q4. adequacy of the courses to the serialization within the curriculum;
- Q5. overall importance of the courses for the student formation;
- Q6. overall quality of the contents presented in the courses;
- Q7. classification among all courses in the curriculum;
- Q8. degree of motivation to pursue hardware studies/work.

The third set of questions asks the students to evaluate specific contents of the lecture course and concern:

- Q9. the use of the hypothetical architecture Cleopatra to approach the teaching of computer architecture concepts;
- Q10. the use of VHDL to describe/validate/implement hardware;
- Q11. the validity of studying pipelining and floating-point arithmetic.

The fourth set of questions asks the students to evaluate the lab course. The specific questions concern:

- Q12. the available resources for performing practical works;
- Q13. the amount of help the lab course represents in apprehending the lecture course concepts;
- Q14. the complexity of the employed CAD tools (most of them commercial ones).

The last two questions are related to the complexity (Q15) and validity (Q16) of the lecture courses final work.

### B. Survey Data Presentation and Discussion

The evaluation of the survey led the authors to choose a 0–5 range for the answers of the students. Except for questions Q1 to Q3, a score of 5 represents the best evaluation for the course aspect and 0 represents the worst possible evaluation. The global score 2.5 for any question is the average value. Each particular aspect can be above or below what the authors (and the students) expect from the courses based on this average value.

Table III presents the average scores for each question and for each semester, together with the average score for questions Q4–Q16 for each semester. There is also a global overall score.

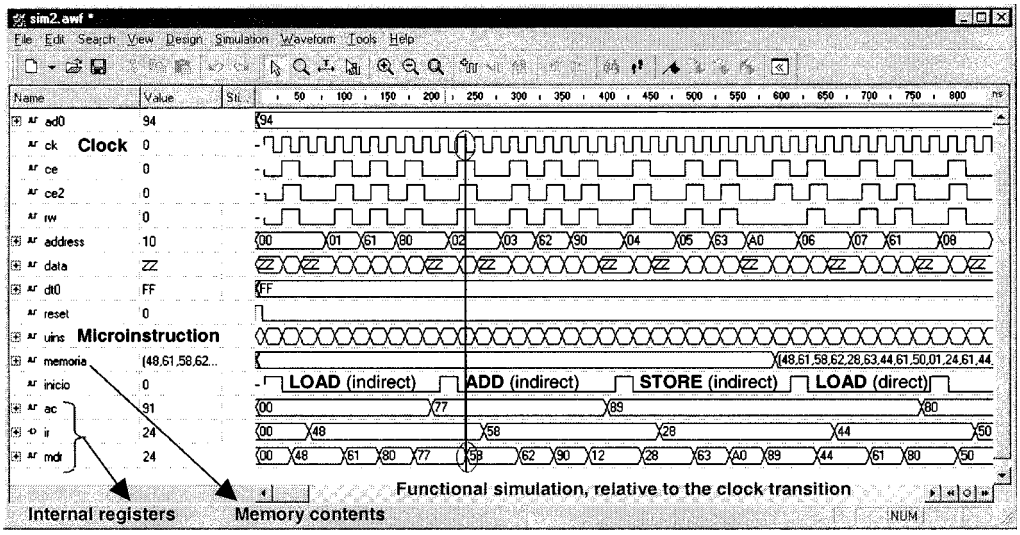



Fig. 4. Partial functional simulation of an object code program in the Cleopatra processor. The simulation shows the clock by clock execution of a four-instruction sequence, LDA, ADD, STA, LDA. Signal `inicio` marks the first clock cycle of each instruction. The relevant register contents are shown in the last three signals of the simulation window (AC, IR, and MDR).

TABLE II
NUMBER OF STUDENTS THAT ANSWERED THE INF46183/INF46184 SURVEY

| Semester | 98/I | 98/II | 99/I | 99/II | Total |
|---|---|---|---|---|---|
| No. of students | 26 | 29 | 37 | 20 | 112 |

TABLE III
AVERAGE SCORES FOR THE INF46183/INF46184 SURVEY QUESTIONS, CLASSIFIED BY THE FIRST QUESTION, i.e., THE SEMESTER WHERE THE STUDENTS TOOK THE COURSES FOR THE FIRST TIME, AND EXCLUDING CONSIDERATION OF Q1 TO Q3 SCORES

| | Question | 98/1 | 98/2 | 99/1 | 99/2 | Average |
|---|---|---|---|---|---|---|
| **Student profile** | Q2 | 1.35 | 1.45 | 1.36 | 1.50 | 1.41 |
| | Q3 | 4.42 | 4.83 | 4.32 | 4.75 | 4.58 |
| **Overall quality** | Q4 | 3.77 | 3.82 | 3.72 | 3.75 | 3.77 |
| | Q5 | 2.85 | 3.28 | 3.27 | 3.75 | 3.29 |
| | Q6 | 3.46 | 3.21 | 3.38 | 4.21 | 3.56 |
| | Q7 | 2.54 | 2.32 | 2.50 | 3.65 | 2.75 |
| | Q8 | 1.88 | 1.59 | 2.08 | 3.20 | 2.19 |
| **Contents** | Q9 | 3.73 | 3.62 | 3.70 | 4.75 | 3.95 |
| | Q10 | 3.85 | 2.86 | 3.68 | 4.53 | 3.73 |
| | Q11 | 3.62 | 3.52 | 3.92 | 4.00 | 3.76 |
| **Lab course** | Q12 | 2.35 | 2.97 | 3.58 | 3.30 | 3.05 |
| | Q13 | 3.24 | 2.93 | 3.92 | 4.00 | 3.52 |
| | Q14 | 3.15 | 3.00 | 2.97 | 3.70 | 3.21 |
| **Final work** | Q15 | 2.27 | 2.38 | 2.03 | 1.80 | 2.12 |
| | Q16 | 3.08 | 2.93 | 2.92 | 4.05 | 3.25 |
| **Average from questions Q4 to Q16** | | 3.06 | 2.96 | 3.21 | 3.75 | 3.24 |

Next, follows a set of conclusions regarding the teaching approach proposed here, which can be extracted from the raw data. Some of the conclusions are rather obvious. Others corroborate the views of the authors while suggesting the approach. Still others indicate major difficulties faced by the students, pointing to directions to review and adapt the current method.

One of the most important conclusions comes from Q4, where the students agree, with a high degree of assurance (Global Average 3.77), that the courses are located where they should in the curriculum, occurring neither too early nor too late. This is indeed one of the least obvious conclusions, and reinforces the position of the authors to expose students much earlier to hardware design than traditional methods. Also, most students consider the organization courses of high overall importance (Q5, Global Average 3.29).

Regarding the contents of the courses, the students' evaluations were quite positive (Q9–Q11, all with Global Average above 3.70), especially concerning the use of a hypothetical ar-

chitecture to teach the basic computer organization concepts. Also it is a consensus among students that using a standard language such as VHDL is a good choice to teach hardware implementation at high levels of abstraction. This result indicates that there is no need to recur to educational languages such as CUPL, as suggested in [10].

The lab course was also seen as being of fundamental importance to complement the apprehension of the lecture course concept (Q13, Global Average 3.52). Students of the first two editions of the lab course did not evaluate the lab resources very well. This situation is easily explained by the lack of available prototyping boards in these editions. The average increased significantly as the prototyping platforms started to be used (Q12). Another point of the approach proposed here which is certainly sensitive to criticism is the use of commercial CAD tools. Question Q14 reassures that the choice has been seen as a good one by the students (Global Average 3.21). There is no need to underestimate the students' capacity to deal with such complex tools.

The final work is seen as extremely demanding (Q15, Global Average 2.12), and this situation points to a need to revise its basic requirements. More significant, though, is the students' evaluation concerning the importance of this final work, which is very good (Q16, Global Average 3,25). It is important to notice that the three first averages were around 3.00, and the last one was significantly higher (4.05). The explanation is in the use of the final work as the starting point for one practical work in the following course, Computer Architecture I; practice started only in the last period. This certainly has motivated students to evaluate very well what they had learned in the previous courses.

## VIII. Current State and Evolution

In the first two editions of the Computer Organization courses (98/1 and 98/2) only design entry and functional simulation tools have been employed, because of the lack of prototyping platforms, obtained only in 1999/1. The next step was adapting the course contents to the new reality, inserting lab experiments to address physical synthesis, approaching performance results of this step, and downloading and testing of designs on the prototyping platform. This implied a reformulation of lecture and lab contents. As a result, a better balance between abstract design issues and effective digital systems implementation was achieved in the available time. Table I reflects part of the structure of the lab course.

Another consequence of the availability of effective hardware design resources is the extension of the approach proposed here to the Computer Architecture (I and II) and microprocessor courses in the same curriculum. The main objective is to provide students with a continued exposure to hardware design issues during their academic life. Advanced topics that are to be covered in such extensions are, for example, pipelining and cache hardware implementation issues. Currently, the Computer Architecture I course has already been adapted. There, the students design a simple pipelining enhancement for the version of the Ramses architecture they worked in Computer Organization.

Connecting the subjects covered here to the elective courses on digital systems at the end of the curriculum is another ongoing work. In the future, the authors intend to explore other advanced aspects of digital systems. One example is the development of CAD algorithms. With the hardware and software support tools available, there is one subject which is recent and important that can be examined in elective courses, the hardware software partitioning and codesign [25]. Last but not least, the authors are considering the application of digital systems design technology to other, apparently unrelated disciplines in Computer Science, such as compilers, computer networks, and computer graphics.

## IX. Conclusion

The main goals set while developing the work proposed in this paper have been attained. It is the view of the authors that abstract digital systems design and computer organization subjects have been successfully integrated in a single teaching approach. The first results of this work were reported in [26].

After teaching the courses four times, the authors have achieved a considerable level of satisfaction on the part of the students as discussed in detail in Section VII. Several successful functional processor implementations have been obtained and some of these present outstanding quality. In some cases considerably complex applications, such as bubble sort and quick sort procedures, were programmed and run in the designed processors by the students alone.

Some questions can be posed and answered in the context of the approach proposed here. First, it is convenient to ask, "*How early can computer science students start designing digital systems design?*" The authors' experience demonstrates that students may start at the third semester of an undergraduate curriculum, and yet achieve a high degree of success. Second, "*How far can computer science students go in doing digital systems hardware design?*" So far, it has not been possible to answer this question fully, since the first students are now taking the first of the elective courses in digital systems. The authors are very optimistic about this question, based on the experiments conducted with undergraduate and graduate students at the research level.

A problem that only now starts to be treated by the authors is the quantitative assessment of student performance and effectiveness of the approach, and this should receive increasing attention in the next editions of the courses.

The positive aspects of the approach proposed here (as obtained from the survey data in Section VII-B) are: 1) teaching of computer organization together with digital systems design at early stages of undergraduate courses; 2) use of standard HDLs like VHDL; 3) use of hardware prototyping platforms in lab courses; and 4) final work including the design and validation of a computer organization/architecture. On the other hand, the negative aspects are seen as: 1) high failure rate in the lecture course; 2) overall complexity of the final work; and 3) and difficulty in motivating students to proceed studying hardware subjects.

Presently, several measures address the solution to the negative aspects identified above, although it is still too early to evaluate the efficacy of these measures. First, there was a migration of subjects from the unit on advanced topics of the

Computer Organization course to the Computer Architecture I course, reducing the overall contents of the first. Second, the final work implementation of the Computer Organization course is presently used in the Computer Architecture I course, enhancing the integration and continuity of contents in the two courses. The amount of practical work in the classroom has been increased, to enhance students' motivation. Finally, the final work complexity has been reduced by the extraction of instructions and structures that are seldom used in the Ramses ISA. The 2000/I and 2000/II results have already shown a small decrease in the failure rate.

In the near future other measures are taking place. The most important of these is in the Digital Circuits course (see Fig. 1). This course is presently under the responsibility of the Electrical Engineering Department, which teaches the course in a traditional way, just like it does for engineering students. This course will pass to the responsibility of the Computer Science Department, where the ideas presented in Section III of this work can be applied fully. This lack of application is perceived today as the main source of students' lack of motivation for the subsequent hardware courses.

Most of the material employed in the implemented courses, like case study specifications and design, lecture presentations, course notes, and free software are available at the following URLs (for the moment, most of this material is in Portuguese): http://www.inf.pucrs.br/~moraes/Dorglab.html, http://www.inf.pucrs.br/~calazans/org_comp.html. Material that is not freely available to students can be obtained by contacting the authors directly via e-mail.

## ACKNOWLEDGMENT

The authors would like to acknowledge the continued support of Xilinx, Inc., through its University Program. Aldec, Inc. has provided the authors with two years of free licenses of its VHDL simulator. They are in great debt to D. C. Liedke, a former student who implemented the software tools. The authors are also grateful to all students who answered the survey.

## REFERENCES

[1] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*. Boston, MA: Kluwer, 1992, p. 206.

[2] S. Guccione. List of FPGA-based computing machines. [Online]. Available: http://www.io.com/~guccione/HW_list.html

[3] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 2nd ed. San Mateo, CA: Morgan Kaufmann, 1998, p. 964.

[4] L. R. Pardo, M. Moure, M. Valdés, and E. Mandado, "VISCP: A virtual instrumentation and CAD tool for electronic engineering learning," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/, pp. 1095–1099.

[5] P. M. Maurer, "Electrical design automation: An essential part of computer engineer's education," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/.

[6] ——, "Enhancing the hardware design experience for computer engineers," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/, pp. 60–63.

[7] H. Grünbacher, "Teaching computer architecture/organization using simulators," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/, pp. 1107–1112.

[8] M. S. Nixon, "On a programmable approach to introducing digital design," *IEEE Trans. Educ.*, vol. 40, pp. 195–206, Aug. 1997.

[9] J. O. Hamblen, H. L. Owen, S. Yalamanchili, and B. Dao, "An undergraduate computer engineering rapid systems prototyping design laboratory," *IEEE Trans. Educ.*, vol. 42, pp. 8–14, Feb. 1999.

[10] W. Kleinfelder, D. Gray, and G. Dudevoir, "A hierarchical approach to digital design using computer-aided design and hardware description languages," in *Proc. 1999 Frontiers Educ. Conf.*, San Jose, PR, Nov. 1999, http://fairway.ecn.purdue.edu/~fie/, pp. 13c6-18–13c6-22.

[11] J. P. Avery, J. L. Chang, M. J. Piket-May, J. F. Sullivan, L. E. Carlson, and S. C. Davis, "The integrated teaching and learning lab," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/.

[12] J. Y. Hung and S. M. Wentworth, "An integrated approach for electrical engineering laboratories," in *Proc. 1998 Frontiers Educ. Conf.*, Tempe, AZ, Nov. 1998, http://fairway.ecn.purdue.edu/~fie/.

[13] A. Zemva, A. Trost, and B. Zajc, "A rapid prototyping environment for teaching digital logic design," *IEEE Trans. Educ.*, vol. 41, p. 342, Nov. 1998.

[14] N. R. McKenzie, C. Ebeling, L. McMurchie, and G. Borriello, "Experiences with the MacTester in computer science and engineering education," *IEEE Trans. Educ.*, vol. 40, pp. 12–21, Feb. 1997.

[15] D. Gajski and R. Kuhn, "New VLSI tools," *IEEE Computer*, vol. 16, pp. 11–14, Dec. 1983.

[16] N. L. V. Calazans, "Automated logic design of sequential digital systems, Chapter 1: Introduction" (in Portuguese), in *Imprinta Gráfica e Editora Ltda*, Rio de Janeiro, 1998, http://ftp.inf.pucrs.br/pub/calazans/pubs/prjlog/v1.0/Cap1.ps., pp. 1–42.

[17] S. Mazor and P. Langstraat, *A Guide to VHDL*. Norwell, MA: Kluwer, 1992.

[18] Xess Corporation. FPGA products. [Online]. Available: http://www.xess.com/FPGA/

[19] F. G. Moraes, N. L. V. Calazans, F. Silva, and M. Barrios, "Cleo-LIRMM: An experiment of dedicated processor implementation on embedded systems prototyping platforms" (in Portuguese), in *Proc. V IBERCHIP Workshop*, Lima, Peru, 1999, pp. 81–90.

[20] G. Kane and J. Heinrich, *MIPS Risc Architecture*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[21] *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Std 754-1985, Aug. 1985.

[22] E. Sentovich *et.al*. (1992, May) SIS: A system for sequential circuit synthesis. Univ. California, Berkeley, CA. [Online] Tech. Rep. UCB/ERL M92/41. Available: http://www.cad.eecs.berkeley.edu/Respep/Research/vis/~usrDoc.html.

[23] Xilinx Inc., "Homepage,", http://www.xilinx.com/.

[24] Aldec Inc., "Homepage,", http://www.aldec.com/.

[25] G. De Micheli and M. Sami, *Hardware Software Co-Design*. Boston, MA: Kluwer, Jan. 1996, p. 467.

[26] N. L. V. Calazans and F. G. Moraes, "VLSI hardware design by computer science students: How early can they start? How far can they go?," in *Proc. 1999 Frontiers Educ. Conf.*, San Jose, PR, Nov. 1999, http://fairway.ecn.purdue.edu/~fie/, pp. 13c6-12–13c6-17.

**Ney Laert Vilar Calazans** (S'90–M'92) was born in Maceió, Brazil, on October, 15, 1959. He received the bachelor's degree from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in electrical engineering in 1985, received the M.Sc. degree in computer science in 1988, also from UFRGS, and received the Ph.D. degree in microelectronics in 1993, from the Universite Catholique de Louvain, Belgium.

He is currently a Professor at the Catholic University of Rio Grande do Sul (PUCRS). His research interests include fast hardware prototyping techniques, hardware-software codesign, digital systems teaching, and computer-aided design tools and techniques for digital systems.

Dr. Calazans is a member of the Brazilian Computer Society, SBC.

**Fernando Gehm Moraes** (M'97) was born in Passo Fundo, Brazil, on August 19, 1965. He received the Bachelor's degree from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in electrical engineering in 1987, received the M.Sc. degree in computer science in 1990, also from UFRGS, and received the Ph.D. degree in microelectronics in 1994, from the Université de Montpellier, France.

He is currently an Associate Professor at the Catholic University of Rio Grande do Sul (PUCRS). His research interests include fast hardware prototyping techniques, digital systems teaching, and computer-aided design tools and techniques for integrated circuits layout.

Dr. Moraes is member of the Brazilian Computer Society, SBC.