From Boolean Algebra to Processor Architecture and Assembly Programming in One Semester

José Silva Matos, José Carlos Alves, Hélio Sousa Mendonça, António José Araújo
Department of Electrical and Computer Engineering
Faculty of Engineering of the University of Porto
Porto, Portugal
{jsm, jca, hsm, aja} @ fe.up.pt

Abstract* — The paper presents the approach followed at the Faculty of Engineering of the University of Porto, to introduce design automation tools and structured design techniques in the first course on digital system design of our Integrated Master in Electrical and Computer Engineering. Digital Systems Laboratory is an introductory course on digital design, with the classical task of teaching Boolean algebra and combinational and sequential circuit design, using gates, flip-flops and medium complexity components/function blocks like counters and shift-registers. The need to cope with new curriculum requirements and modern digital design demands, motivated an extensive reformulation of the course contents and organization, leading to the introduction of the use of hardware description languages and synthesis tools, in order to implement small systems, of increasingly complex nature, on an FPGA platform. At the same time its coverage was extended to include low-level processor architecture issues, and to teach assembly programming for the MIPS processor. The paper describes how this reformulation was carried out. It presents the course contents and timeline, and discusses the main choices that were made. The paper also describes the laboratory experiments that were developed and discusses some of the challenges and results obtained so far.

Keywords — Digital design; processor architecture; assembly language; HDL; FPGA.

I. Introduction

One of the main challenges of teaching and learning integrated circuit and system design is the wide range of abstraction levels to be covered and mastered. They range from the very low level of transistor circuit design, or mask layout design, up to the higher abstraction levels of architectural and algorithmic design. The engineering of complex systems, taking advantage of today's advanced processes and extremely high integration densities, requires moving up in the abstraction hierarchy. It requires the ability to make use of behavioral/structural design description languages and synthesis tools, and to pay increasing attention to the co-design of software and hardware. The know-how and skills expected from modern digital designers demand specialized training which may benefit from an early introduction to higher-level, more abstract views of the design process.

Digital Systems Laboratory is a first year, first semester course of the Integrated Master in Electrical and Computer Engineering, a 5-year program offered by FEUP, the Faculty of Engineering of the University of Porto. It is classical program with a common core of courses on physics, matehematics and on fundamental subjects in Electrical Engineering, and three main branches of specialization: Telecommunications, Electronics and Computers; Automation; Energy.

The last revision of the program, which took place in 2009, established a demanding set of new objectives for the course:

- to increase its laboratorial component;
- to introduce and promote the early use of design automation tools and of design-entering techniques like schematics capture and hardware description languages;
- to offer an introduction to processor architecture and to assembly language programming.

The goal is to prepare the students for courses to follow on computer architecture, on microprocessor based systems, and on integrated circuit and system design, giving them a solid understanding of both fundamental concepts and of the underlying basis on which hardware and software systems are built. These new objectives are to be achieved in addition to the ones found in a classical first course on digital design.

The approach that was followed adopted a move towards a more "RTL-like" view of the design process, and the use of Electronic Design Automation tools with a reconfigurable platform based on a Xilinx FPGA [1]. A sequence of laboratory assignments was developed to conduct the students through the design of a simple processor with a very basic instruction set, which was used to write simple, truly low-level programs. In order to highlight fundamental aspects of the evolution from this very simple architecture to that of a real processor, MIPS was chosen for its wide use in the teaching of computer architecture [2]. Learning assembly programming of the MIPS processor benefitted considerably from the good understanding of the register transfer mechanics that was obtained earlier, and resorted to MARS, a widely available open access MIPS assembly language simulator [3].

This section establishes the framework on which the course was developed and its general requirements. Section II presents the objectives, contents and learning outcomes of the course as well as its timeline, highlighting the time and effort dedicated to the different subjects. Laboratory experiments are described in more detail in Section III, together with the FPGA based hardware platform and the EDA tools that were selected. Section IV discusses some of the difficulties encountered and results obtained. Section V closes the paper and draws some preliminary conclusions.

II. Course contents and organization

Leading students from the fundamentals of Boolean algebra, through combinational and sequential design, to the basic architecture of a microprocessor and to the assembly-language programming of a real processor is a very challenging goal. It becomes even more demanding if we realize that students take this course in the first year of their university education. Thus, the organization of the course, its contents, and their articulation with a strong laboratory component required careful attention.

The three main objectives of the course are: i) to explain the theoretical foundations and practical aspects of the analysis and synthesis of digital systems (combinational and sequential); ii) to offer an introduction to modern digital system design using hardware description languages and tools for specification, simulation and synthesis, and iii) to introduce the fundamental concepts associated with the internal organization and operation of microprocessors and their programming in assembly language.

In addition to acquiring the basic skills expected from a classical introductory course on digital design, students are introduced to the use of a hardware description language (Verilog), and to applying concepts of modularity and hierarchy in the description of digital systems. Additionally, a basic degree of familiarity with the use of schematics capture, simulation and synthesis tools is developed along the sequence of laboratory assignments to be described next. These assignments exercise the typical design flow of digital system implementation in programmable logic devices (FPGA).

Understanding the organization and the operation of the datapath of a simple 8-bit microprocessor (ALU, registers, multiplexers and buses) and of its control unit (instruction decoding and sequencing) comes naturally from the sequence of laboratory assignments. The students observe the incremental improvement of an initial design, from a simple ALU up to a stored-program processor. They deal with issues of sequencing and instruction set design and modification, and end up writing bit-level sequences of instructions, which are in fact machine-language programs for the prcessor they developed.

The last part of the course is fully dedicated to the assembly programming of an industrial *de facto* standard. MIPS was chosen for being well-known in the academic community, and for its extensive range of industrial application areas [4]. The students learn its programming model, the instruction set and the assembly language syntax and rules. They are asked to analyze, develop and demonstrate small programs for vector searching and sorting, with particular attention being given to modular programming, with subroutines. We observe that picking up assembly programming skills, in such a short period of time is greatly facilitated by their earlier exposure to relatively low-level aspects of a similar architecture.

The course meets 5 hours per week: one 1-hour class and two 2-hour classes. The 1-hour classes are used to motivate and present new concepts, to introduce new laboratory assignments and to highlight results and discuss problems encountered in the lab. In the first five weeks of the semester both 2-hour classes are used to lecture fundamental material: Boolean algebra, basic binary arithmetic, logic minimization, combinational design with gates, decoders and multiplexers. Examples and exercises are also given in classes that are essentially of a pencil-and-paper nature. In the remaining nine weeks of the semester, a stream of 2-hour classes is used for the laboratory assignments, running in parallel, and interacting closely, with the other more lecture-oriented classes. The organization and sequencing of the laboratory classes are described in the following section.

III. Laboratory experiments

A. Organization

In the first laboratory assignment students build a basic digital circuit with gate-level logic devices (e.g. the 74xxx series). The objective is to get them acquainted with the physical and electrical issues of assembling simple electronic systems in a breadboard prototyping environment.

The second block of laboratory experiments leads the students from the gate-level design of an adder/subtractor to the construction of an elementary microprocessor. It includes a sequence of four lab assignments, leading to the construction of a simple, albeit fully functional, 8-bit RISC-style microprocessor. The students do not build by themselves the whole system from scratch, as some parts of the design are pre-prepared for them. However, the incremental approach adopted successfully helps them obtaining a good understanding of the internals of a microprocessor, exercising in parallel the application of digital design principles and the implementation flow for an FPGA platform.

The third and final block of labs deals with the MIPS assembly programming, moving up in the abstraction ladder from the introductory concepts of low-level code and execution flow introduced with the microprocessor built in the previous group of labs.

B. From logic gates to a microprocessor

The first lab in the second block addresses gate-level design principles, from truth-tables and Boolean functions to two-level logic minimization. In this lab the students start by building a full-adder cell and then assemble, simulate, implement and test a fully combinational 4-bit adder/subtractor using solely schematic capture for design entry (Fig. 1). Concepts of hierarchical design and detail abstraction are introduced and put in practice here.

The second lab tackles the utilization of standard logic functions mapped to register-transfer level blocks, exploits RTL synthesis for basic combinational functions and identifies the set ALU, accumulator and flags, as a core element of microprocessor architecture. The students are given the description of a logic system, in Verilog HDL, containing an ALU and accumulator, and are asked to design a custom block to generate 4 flags from the ALU result and to integrate it in the project (Fig. 2).

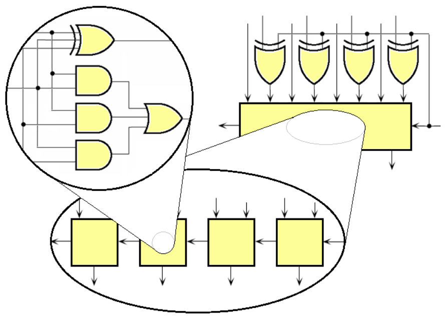

Fig. 1. Hierarchical view of the gate-level design of a 4-bit adder/subtractor

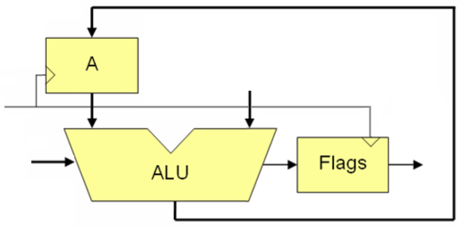

Fig. 2. ALU, accumulator and flag generation block

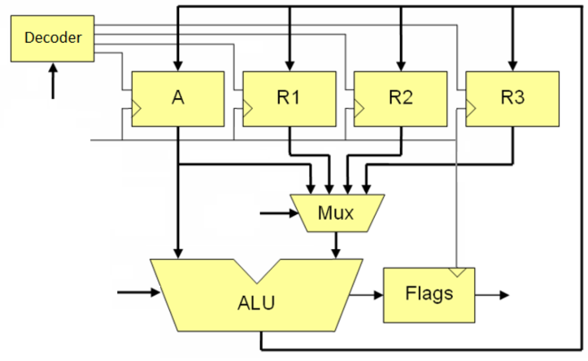

Fig. 3. 8-bit data-path with ALU and 4 registers

In the third lab, the previously designed ALU-accumulator datapath is enhanced with 3 general-purpose registers, thus creating the basis for enabling the execution of sequences of elementary operations and introducing the concept of assembly-level and machine-level instruction and program (Fig. 3). From the digital design point of view, the students have to assemble the datapath using existing blocks and developing missing parts. As an early introduction to low level programming, the students practice with the partition of simple computations into elementary register-to-register operations capable of being handled by the datapath, and sequentially execute them by manually setting the logic control signals and activating the clock signal. Although the students encode manually their programs to machine code, the concept of *assembler*, as the process for automatic translation of the assembly language to machine code, is also introduced here.

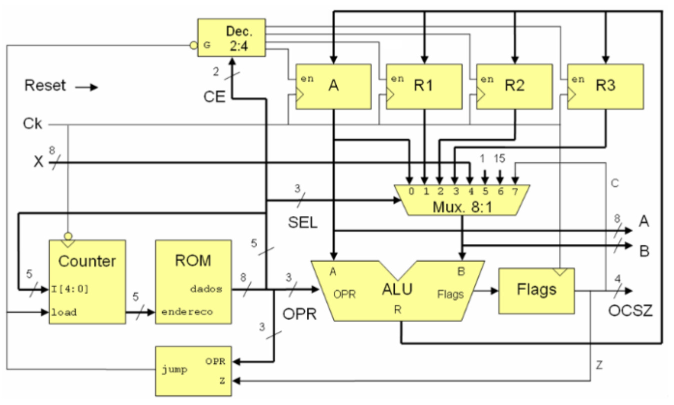

Fig. 4. The complete processor

The last lab exercise completes the series with the addition of the stored program paradigm and control flow mechanisms based on conditional and unconditional jump instructions (Fig. 4) that are added to the instruction set. The students design a ROM-like block, pre-loaded with their binary hand-encoded programs and experiment with small examples to illustrate the control and data flow within the processor. One example of an exercise proposed is the search of the maximum in a list of numbers. The program receives a list of positive numbers introduced sequentially through the processor's input port (8 slide switches), ended with a zero. Figure 5 illustrates the type of very elementary programs that the students are asked to develop, showing side by side the assembly code and the ROM contents for the corresponding machine level program.

The students have to write and verify assembly level programs, translate them to binary code by joining the bit fields defining the opcode, source operand, destination register and destination address (for the jump instruction), integrate the code in the Verilog model of a ROM memory, and implement and verify its operation in the physical FPGA platform.

```
                                   operand  destination register
         R1 <- A - A      0 0 1 0 0 0 0 1
loop:    A  <- X          0 0 0 1 0 0 0 0
         jnz   contin     1 0 1 0 0 1 0 1
         A  <- 15         0 0 0 1 1 0 0 0
stop:    jnz   stop       1 0 1 0 0 1 0 0
contin:  R2 <- A - R1     0 0 1 0 0 1 1 0
         R2 <- carry      0 0 0 1 1 1 1 0
         jnz   newmax     1 0 1 0 1 0 1 0
         R2 <- 1          0 0 0 1 0 1 1 0
         jnz   loop       1 0 1 0 0 0 0 1
newmax:  R1 <- A          0 0 0 0 0 0 0 1
         jnz   loop       1 0 1 0 0 0 0 1
                          opcode  destination address
```

Fig. 5. The assembly and machine code for a typical exercise

C. *MIPS assembly programming*

By the end of this sequence of exercises and laboratory experiments the students are aware of the constraints resulting from the limited number of registers, the accumulator-based nature of the datapath, the lack of a data memory and the narrow 8-bit width for encoding instructions. Upon this, the MIPS single-cycle architecture appears logically as an expanded version of the processor developed, facilitating the understanding of the internal mechanisms involved in the execution of instructions and shortening significantly the learning curve of low-level assembly programming.

This is the point where students are introduced to MARS [3], a well-known simulator/development system for the MIPS architecture and assembly language. The lab assignments explore the main features of MARS to support the analysis and debugging of programs, including step-by-step instruction execution, the use of breakpoints, and the observation of register and data memory contents.

As in the more hardware-oriented part of the course, a great deal of importance is given to the fundamental concepts of modularity and hierarchy, and thus lab exercises pay particular attention to features that support subroutine calling and return address handling. The students are given a program, using structured and properly commented code, and are asked to complement it by adding routines for specific tasks to be performed on vectors of integer numbers (e.g. computing the average, finding a maximum, or minimum value, swapping vector elements, sorting).

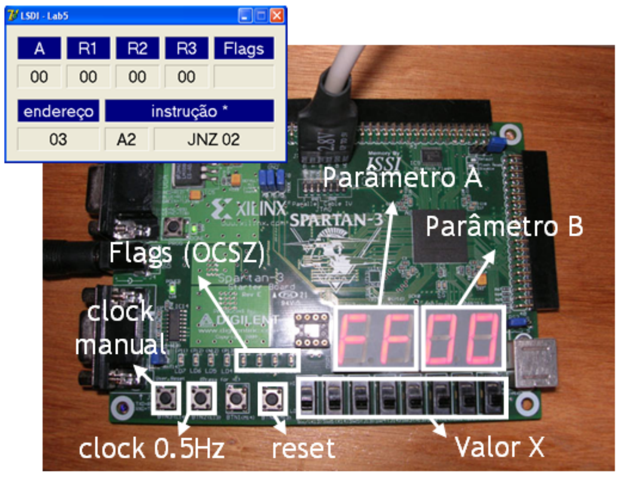

Fig. 6. FPGA board and PC application for enhanced interaction.

D. *FPGA board and development tools*

The sequence of laboratory assignments leading to the development of the microprocessor, makes use of a low cost development board with reconfigurable logic: the Spartan-3 Starter Kit [5]. After designing the circuit with the ISE WebPACK Design Software from Xilinx, the students program the FPGA and validate the expected results. This interaction with the developed system is performed through the input/output elements in the board as can be seen in Fig. 6. For this purpose, a complete design is made available, with all the logic needed to interface with the switches, pushbuttons, LEDs and other displays on the board.

The limitations of the board and the increasing complexity of the system led to the development of an application, running on a PC, whose output may be seen overlapping on Fig. 6. Besides monitoring the values present in the processor registers, it allows the disassembly of the instruction being executed as well as its memory location. This way the students are guided through the step-by-step execution of a program, and have therefore a better understanding of the microprocessor operation.

IV. Discussion of Main Challenges and Results

Several challenges may be identified in the task of reformulating a classical introductory course on switching theory and digital design, in order to cover a considerable amount of new material, at a higher level of abstraction, with a stronger laboratorial content. One of them is certainly the fact that the duration of the course remained at one semester; another is the consideration that the course is located in the first year of the program and the students do not have yet the autonomy and maturity that would be desired for a more in-depth treatment of the same subjects.

Covering more material brought about the need to consider several trade-offs quite early in the design of the course, accepting less depth in the treatment of certain topics. Examples are a reduction in the attention given to the manual design of finite-state machines, with different types of flip-flops, and to the minimzation of their state/output logic. Also the time for the design of small systems with counters and shift-registers, based on the 74 family of MSI devices, in particular their laboratorial set-ups, had to be shortened.

Another change that was decided after the first edition of the course was the emphasis on the behavioral side of Verilog, which had to be reduced. At this point, the students still have an insufficient background on high-level programming to support the exploitation of behavioral descriptions. The solution we adopted included to provide the students with examples of such descriptions, which were fed as input to the synthesis tool, without requiring from them the understanding of their details.

The use of design automation tools and HDL-based design-entry techniques, exercising the typical design flow for a reconfigurable implementation platform, brought about an enormous advantage in the compexity and sofistication of the digital design problems that may be tackled and of the systems that can be designed. We believe this advantage clearly offsets many possible drawbacks. Furthermore, it is in tune with an unavoidable need to promote an early adoption of high-level, abstract thinking, and detail encapsulation techniques, in the way students learn modern digital design.

Despite their relatively young age and lack of previous experience and exposure to these materials, the students enjoy the course. It is our opinion that they recognize and value the quality of the training that is made available to them, and the effort that was put in its preparation. A very positive reaction is particularly visible in the best students, who seize the possibility of doing extra work, to extend and improve the original design. The overal results (pass/fail ratio) are in line with those for other courses at the same level in our School.

The goals are, recognizedly, ambitious. Of course, having more time (e.g. spreading the course over two semesters), and more contact hours, would make things less demanding for teachers and easier for students to master the concepts and the techniques they are taught. Still, it is our experience that the organization that was presented, with the limitations that have to be recognized, is effective at reaching the stated goals.

V. Conclusions

Admitedly, the treatment that is provided on the different subjects of the course is of an introductory nature. We don't claim the students end up with a strong domain of logic design, processor architecture, and assembly programming. We believe though that the students are offered a good opportunity to learn structured digital design in a well organized teaching environment. In our view this gives them a good understanding of the issues and enables them to handle more confidently the design challenges that they find in more advanced courses in the program, and later in their professional life.

References

[1] http://www.xilinx.com [accessed on May 2014]

[2] A. Patterson and J. L. Hennessy, Computer Organization and Design - The Hardware/Software Interface, 4th ed., Morgan Kaufmann, 2009

[3] Vollmar, Kenneth, and Pete Sanderson. "MARS: an education-oriented MIPS assembly language simulator." ACM SIGCSE Bulletin. Vol. 38. No. 1. ACM, 2006

[4] MIPS32 Architecture, http://www.imgtec.com/mips/architectures /mips32.asp [accessed on May 2014]

[5] Spartan-3 Starter Kit Board User Guide, ug130 V1.1 – XILINX, 2005