Introductory Digital Design & Computer Architecture Curriculum

David Money Harris
Department of Engineering
Harvey Mudd College
Claremont, CA
David_Harris@hmc.edu

Sarah Harris
Department of Engineering
Harvey Mudd College
Claremont, CA
Sarah_Harris@hmc.edu

Keywords—digital design, computer architecture, project-based learning

I. Introduction

According to the ABET criteria for Electrical and Computer Engineering Programs, curriculum must include "engineering topics necessary to analyze and design complex electrical and electronic devices, software, and systems containing hardware and software components." A key element of this curriculum is a course or sequence of courses covering digital design and computer architecture. The introductory portion generally is offered in the second or third year and spans one or two semesters or two quarters.

We have taught such a course at Harvey Mudd College over the last fourteen years [1] and have recently published the second edition of a textbook providing an integrated introduction to digital design and computer architecture [2]. This paper describes a set of principles that guides our approach and that we believe are applicable to the curriculum elsewhere.

II. Principles

- **Capture the Magic**

  Most budding engineers have grown up as savvy computer users, and many have rich programming experience, but few understand what is happening inside a microprocessor. For many, demystifying this black box is an almost magical experience; we aim to capture this excitement to motivate not only this subject, but other subjects in nearby abstraction layers. Fig. 1 shows one of the metaphorical cartoon chapter openers used to simultaneously convey concepts and excitement.

- **Project-Based Learning**

  Extensive research confirms the value of project-based learning. Working our way up to the design of a simple microprocessor ties together most of the concepts in digital design, making the integrated course more than the sum of the parts. Students quickly appreciate the power of digital systems to build interesting systems. Many obtain a first internship or job based on their digital knowledge because a modest amount of study prepares them to tackle real-world problems. We find that commercial tools, languages, and architectures are preferable for many reasons over simplified versions concocted for educational purposes.

- **Abstraction**

  Abstraction is at the heart of computer engineering, and its subtleties are best illustrated by example. We repeatedly explore abstractions in different contexts to help it become part of the student's thought process.

- **Emphasize the Fundamentals**

  Although the field is broad, we focus on a core set of fundamental ideas necessary to lead students to the point that they are capable designers.

- **Integrate Hardware and Software**

  This subject is an ideal place to bring together the worlds of hardware and software. Engineers in all fields today need to apply software skills as well as domain-specific knowledge to solve complex problems, and this demand for programming ability will continue to increase.

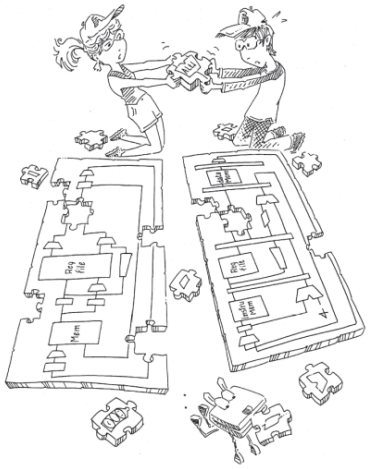

Fig. 2. Chapter 6 opener cartoon

III. CURRICULUM

We take both bottom-up and top-down approaches to the subject. We briefly outline the relevant levels of abstraction, shown in Fig. 2. We begin with the digital abstraction, starting from 0s and 1s and logic gates that combine them. We briefly dip down to consider the physical representation of these digital values and the implementation of gates using CMOS transistors viewed as switches.

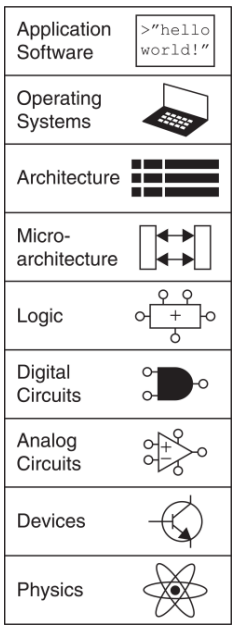

Fig. 2. Levels of abstraction

We then begin an upward march through combinational and sequential logic design to reach finite state machines and digital blocks including arithmetic circuits and memories. We emphasize Boolean algebra and sum-of-products as intuitive design tools. We cover Karnaugh maps because of their prevalence as interview questions, but omit more advanced synthesis algorithms that are best executed by computer. Some texts treat digital logic as a branch of discrete mathematics, but we prefer for practical and pedagogical reasons to treat it as a branch of design with emphasis on creativity and problem solving. Applying abstraction again, we combine logic gates to form SR latches, D latches, and D flip-flops; we then focus on synchronous sequential circuits using D-flip-flops because that is the discipline used in nearly all commercial products. We omit asynchronous design in the introductory course, but emphasize the appropriate use of synchronizers at clock domain crossings. Finite state machines and arithmetic blocks tie together most of the topics developed so far.

We initially emphasize visualizing circuits using schematics because that focuses on the tangible nature of hardware and the inherent parallelism. After students become comfortable with schematics, we introduce a Hardware Description Language (HDL). The most common source of problems among beginners is to view HDL as a programming language rather than a shorthand for efficiently specifying hardware. We address this with an idiomatic approach, starting with the hardware we want to describe and generating the appropriate HDL code that implies such hardware. We offer side-by-side coverage of SystemVerilog and VHDL 2008. Both languages smooth out some of the clumsy constructs that troubled newcomers in the original Verilog and VHDL syntax. Every hardware idiom comes with examples in both languages, providing a "Rosetta stone" for those who learn one HDL and need to quickly switch to the other.

We now jump to a high level of abstraction and work back down to meet in the middle. We believe that all engineers graduating today need competence in a high-productivity language such as Python and a domain-specific language such as Matlab; from this experience, they will be able to learn other languages on the job. Electrical and computer engineers also need familiarity with C, both because of its immense practical value in embedded systems and because it provides a conceptual bridge between programming concepts and the underlying memory model and hardware. We expect students will learn scripting and domain-specific languages elsewhere in their studies, but we quickly cover C to the point that one can read and write simple programs.

We next work our way down to instruction set architecture. Following the lead of Patterson and Hennessy [3], we examine the MIPS architecture because it is easy to learn, elegant, and commercially significant. We develop through example the relationship of C programs to assembly and machine language.

Finally, we descend to microarchitecture to complete the link from 0's and 1's to microprocessor and programs. We examine three different MIPS microarchitectures: multicycle, single-cycle, and pipelined, to show how the same architecture can be implemented with multiple cost and performance tradeoffs. We briefly examine the key capabilities of a superscalar out-of-order processor to provide an appreciation of modern CPU architecture.

The final chapter of the book covers memory systems and I/O. Our E85 class now deemphasizes memory systems to make room for C programming in a single semester, but a longer sequence would be able to offer better coverage. The I/O chapter contains many practical examples of embedded I/O on a MIPS-based PIC32 microcontroller, including GPIO, RS-232 and SPI, ADC and DAC, timers, motor drivers, and interfacing with LCDs. Each example has code and schematics to demystify the hardware-software interface. The chapter also has a SystemVerilog example of a character generator for a VGA monitor. We return to these examples in our second-level course on embedded systems [4].

IV. LABORATORY

Digital design can only be fully appreciated by doing it, so a set of laboratory exercises accompany the book. Table I lists the lab projects. The first lab builds a full adder with discrete TTL chips to show logic gates as physical objects, but subsequent labs are done on field-programmable gate arrays (FPGAs) because productivity is higher and FPGAs are the commercially relevant way to prototype today. The next four labs involve combinational and sequential circuit design, with two weeks of schematics and two of SystemVerilog. Labs 6-8 switch to a MIPS-based PIC32 board to cover C and assembly language programming and interfacing a microcontroller with external hardware. Lab 9 guides students through the design of a single-cycle MIPS processor running a test program, reusing the ALU from Lab 5. The capstone pair of labs gives students the chance to design and debug their own multicycle MIPS processor in SystemVerilog, tying together all of the previous work.

We have used both Xilinx and Altera tools and chips, but presently organize the labs around an Altera DE2 board. Both companies have been very generous with donation of software and development boards. The Xilinx tools used to work well for education, but as they have become optimized to handle very large designs, they have become slow and overly complex

at processing tiny designs, and students encounter error messages that can be difficult to decipher. Moreover, Xilinx has been slow about providing SystemVerilog support. We continue to use Xilinx tools for some advanced projects, but presently prefer Altera for introductory classes.

TABLE I. LABORATORY TOPICS

Week	Laboratory	Objectives
1	Full Adder	Breadboarding & TTL chips
2	7-Segment Decoder	Combinational logic, schematics
3	Adventure Game	FSM, schematics
4	Tail Light Controller	FSM, Verilog
5	ALU with Testbench	Verilog, testing
6	C Programming	Intro to C on a PIC32
7	Temperature Control	Microcontroller interfacings
8	MIPS Assembly	Assembly language on PIC32
9	Single Cycle Processor	Guided processor design
10-11	Multicycle Processor	Independent processor design

V. CONCLUSION

This paper has described the philosophy and content of a textbook on *Digital Design and Computer Architecture*. The text fills a niche in programs that appreciate the elegance of Patterson & Hennessy's *Computer Organization and Design* but wish to integrate digital logic with architecture in a single course or cohesive sequence.

REFERENCES

[1] Digital Design and Computer Architecture (E85) web page: www3.hmc.edu/~harris/class/e85

[2] D. Harris & S. Harris, *Digital Design and Computer Architecture*, 2nd Ed., Waltham, MA: Morgan Kaufmann, 2013.

[3] D. Patterson & J. Hennessy, *Computer Organization and Design*, 4th Ed., Waltham, MA: Morgan Kaumann, 2008.

[4] Microprocessor-Based Systems (E155) web page: www3.hmc.edu/~harris/class/e155