ARM-based Digital Design and Computer Architecture Curriculum

Sarah L. Harris
Department of Electrical and Computer Engineering
University of Nevada, Las Vegas
Las Vegas, NV, U.S.A.
sarah.harris@unlv.edu

David M. Harris
Department of Engineering
Harvey Mudd College
Claremont, CA, U.S.A.
david.harris@hmc.edu

***Abstract*—Many electrical and computer engineering and computer science students take an introductory digital design course followed by a computer architecture course. This paper describes a curriculum to support these courses that culminates in designing a simplified ARM® microprocessor on an FPGA. We also wrote a supporting textbook to help other instructors interested in teaching such a course. The textbook includes supplementary hands-on labs and exercises. Our experience is that enabling students to understand a microprocessor from the underlying gates and microarchitecture all the way up to the assembly and programming levels empowers them to fully understand both computer architecture as well as the design of complex digital systems.**

Keywords—Computer Engineering Education, Digital Design, ARM Processor, Microprocessors, FPGA, Computer Architecture

I. INTRODUCTION

Most electrical and computer engineering and computer science departments include at least one digital design course that is followed by a computer architecture course. The course described here covers both topics and can be taught as a single-semester course or as a multi-semester sequence, depending on the needs of the curriculum. The course begins with an introduction to digital design, then proceeds to introduce more complex digital design problems and hardware description languages (HDLs), and culminates with introducing computer architecture and processor design. The final hands-on lab project is for the students to build an ARM processor on an FPGA using SystemVerilog.

This course is an adaptation of a MIPS-based course taught at Harvey Mudd College as a single-semester sequence and at the University of Nevada, Las Vegas as a multi-semester sequence. The authors also wrote an accompanying textbook to support the course called *Digital Design and Computer Architecture: ARM® Edition* [1].

This paper presents the underlying principles, structure, and content of the course and its accompanying labs. It concludes with a discussion of future improvements and a summary of the course goals.

II. GUIDING PRINCIPLES

We believe that teaching students about a computer processor from the ground up is both exciting and empowering for students. The proposed course teaches students about the underlying gates and architecture of an ARM microprocessor all the way up to the assembly and programming levels of the microprocessor. This helps them fully understand both computer architecture as well as the design of complex digital systems.

The ARM microprocessor is a timely example as it is currently found in 95% of hand-held devices. Thus, students come away from the course with both a broad understanding of digital design and computer architecture as well as a detailed understanding of this microprocessor and the implications of its architecture.

This course is accessible to a broad range of students because it begins at the fundamental level of digital design, starting with 0's and 1's and proceeding to multi-bit numbers, gates, and Boolean functions. Students require little, if any, prerequisites for the course, although an understanding of algebra and basic software programming is useful.

Students learn best by doing, so the accompanying labs are a key part of solidifying the principles taught in the course. The labs begin with the students building basic digital circuits using 74xx parts. Although these parts are archaic, we find that students benefit from seeing the logic circuits in action and this lab enhances their understanding of fundamental concepts such as input variables and equipotential wires, especially as it applies to input signals being connected to multiple

gate inputs. The labs proceed to build on this concept using schematic design. Only then do students learn design entry using an HDL, SystemVerilog or VHDL. We have found that using hands-on design and schematic entry first helps students think of the hardware they expect their design to produce even when they later use HDL design entry. This process of starting with schematic entry first, before jumping into HDL design, solidifies their understanding of hardware design. In contrast, if students start with HDL entry first, they tend to think of HDL programming as a programming (software) language instead of a hardware description language. All of the hardware labs are also implemented on an FPGA, which solidifies students' understanding and debugging skills.

After completing increasingly complex digital designs including finite state machines (FSMs) on an FPGA, the course then moves to the topic of computer architecture and the students complete several C and ARM assembly language programming labs. The labs culminate by bringing both topics together – digital design and computer architecture – with the students building their own simplified ARM processor using in an HDL, their writing an ARM assembly language program for that processor, and finally simulating that program running on their ARM processor. By starting with the fundamental principles of 0's and 1's and digital design and building up to computer architecture and microarchitecture, the students gain a complete understanding of the design, the design process, and of the hardware/software interface.

III. Course Structure

The course is typically taken by students in their sophomore or junior year. It is structured with weekly lectures, readings, written assignments, and labs. When taught as a single-semester (15-week) course, students complete fewer exercises on any given material but cover the same overall topics. The course may also be taught as a multi-semester, and typically a two-semester, sequence. In that case, the students may complete more exercises and hands-on projects but they will still cover the same topics overall as in the single-semester course. For example, at Harvey Mudd College, we have taught the course as a one-semester course because students earn a general engineering degree, so their schedule is compressed due to the demand of also taking courses in other engineering disciplines, beyond electrical and computer engineering. However, we have taught this course as a two- to three-semester sequence at the University of Nevada, Las Vegas to computer and electrical engineering students and computer science students where these students focus their major on computer and hardware design and use instead of general engineering.

The course also includes lectures and accompanying readings in the textbook. The weekly lectures, typically two 75-minute lectures per week, introduce material and work through examples. Students are to complete the readings before attending lecture. Thus, they are introduced to the material on their own first, at their own pace, and then they can ask questions and fill in gaps in their understanding during lecture.

The weekly written assignments then help students solidify fundamental concepts introduced in the lectures and reading, such as binary addition, simplifying Boolean equations, FSM design, ARM assembly programming, and ARM processor modifications. These written assignments are complemented by the students completing hands-on labs using the principles they have previously practiced in written assignments. The written assignments offer a lighter-weight means of practicing the material while the labs solidify understanding of the material. This allows the students to complete more exercises that would be too time-consuming to complete in lab. The assignments require relatively less work for any given design or exercise – for example, students can design an FSM on paper but not need to then complete the extra steps required in the lab of entering the design into the FPGA design software, compiling, simulating, synthesizing, and testing the design on the FPGA. The written exercises also prepare students for exams and future interviews and industry positions.

This complementary learning structure of lectures and readings with written exercises and hands-on labs actively supports various learning styles from hands-on learning to group learning to individual learning at a student's own pace. It allows students to work through mistakes in the lower-stakes environment of in-class examples, in-class group activities, and written exercises before they practice what they have learned and strengthen their understanding during the labs and exams.

The structure of including readings, lectures, written assignments, hands-on labs, and exams also encourages mastery learning of the material as the students practice and experiment with the topics in increasingly higher-stakes environments – first during the reading and in class during lecture, followed by working through the exercises, and finally in lab and on the exams.

Table 1 shows our course syllabus when taught in a single semester. When taught as a two-semester

Table 1. Course Syllabus (Single-Semester Version)

#	Date	Topic	Assignment
1	26-Aug	Introduction: digital abstraction; binary numbers; bits and bytes; logic gates	
2	28-Aug	Transistor-level implementation; truth tables, Boolean expressions	
3	2-Sep	Boolean algebra; K-maps	PS 1
4	4-Sep	X's and Z's; timing, hazards	Lab 1
5	9-Sep	Sequential circuits: SR latches, D latches, flip-flops, clocking	PS 2
6	11-Sep	Finite State Machines	Lab 2
7	16-Sep	*Optional: Dynamic discipline; metastability	PS 3
8	18-Sep	Introduction to Hardware Description Languages (HDLs): Verilog	Lab 3
9	23-Sep	More Verilog	PS 4
10	25-Sep	Building Blocks I: mux, decoder, priority encoder, counter, comparator	Lab 4
11	30-Sept	Building Blocks II: Arrays: RAMs, ROMs, PLAs, FPGAs	
	2-Oct	Midterm 1	
12	7-Oct	Number systems: fixed & floating point, unsigned & signed	PS 5
13	9-Oct	Arithmetic: addition & subtraction, multiplication	Lab 5
14	14-Oct	ARM instruction set and registers	PS 6
15	16-Oct	Branches & procedure calls	Lab 6
16	21-Oct	Addressing modes	PS 7
17	23-Oct	Linking & launching applications	Lab 7
18	28-Oct	Single-cycle processor datapath	PS 8
19	30-Oct	Single-cycle processor control	Lab 8
20	4-Nov	Multicycle processor	PS 9
21	6-Nov	Exceptions	Lab 9
	11-Nov	Midterm 2	
22	13-Nov	Pipelining	
23	18-Nov	Pipelining hazards and stalls	PS 10
24	20-Nov	Memory-mapped I/O	Lab 10
26	25-Nov	Memory hierarchy, latency & throughput Caches	PS 11
27	27-Nov	Memory system optimization, Virtual memory	Lab 11
28	2-Dec	Advanced architecture: a sampler	PS 12
29	4-Dec	Course Review	
	9-Dec	Final Exam	

sequence, we cover the same overall topics, but we spend additional time on most topics. For example, we spend 2-3 weeks each, instead of one, on topics such as Boolean algebra, number systems (binary and other bases), finite state machines, introducing the ARM instruction set, etc.

IV. Written Exercises

Students complete weekly exercises on the topics covered in lecture. This enables them to practice and experiment with the material before completing hands-on labs that require that understanding. Because a given week's material builds on topics learned in prior weeks, this also helps students solidify their understanding before building on that understanding in the material that follows. The weekly homework consists of a selection of exercises from the back of each chapter, and it can also include variations of those exercises.

V. Lab Assignments

The course includes 11 core labs, as shown in Table 2, that allow the students to experiment with topics ranging from digital design – starting with simple designs and then increasing in complexity – to ARM assembly language programming and then ARM processor design.

Students complete the labs using an FPGA board for the hardware design labs and, for the C and ARM assembly programming labs, using the Raspberry Pi, a single-board computer developed by the Raspberry Pi Foundation. The FPGA labs can be completed using either Verilog or VHDL, both of which are supported by the textbook. The labs are designed for the Intel Altera FPGA boards and design software, specifically the DE2 (or DE2-115) FPGA board and the Quartus design software. But instructors could readily adapt these labs to use Xilinx FPGA boards and design software, such as the Nexys4-DDR or Basys3 boards and the Vivado IDE (Integrated Design Environment).

In the first lab, the students design a simple digital circuit, a 1-bit full adder, in schematic by hand first and then using the Quartus software. The students then simulate their design using ModelSim and synthesize the design onto the FPGA on the DE2 board. The students then use 74xx parts on a breadboard to build

the 1-bit adder. Although the last step, breadboard design, is optional, it is recommended that the students complete that step because it helps them understand fundamental principles such as the importance of connecting power and ground, the principle of a wire being equipotential and connecting to multiple gate inputs, and the concept of 1 and 0 as 5V and ground (GND).

Table 2. Laboratory Assignments

Lab	Description	Method
1	1-bit Full Adder	Schematic
2	Seven-Segment Display	
3	Adventure Game Finite State Machine	
4	Turn Signal Finite State Machine (FSM)	HDL
5	32-bit ALU	
6	C Programming: Fibonacci numbers +	C
7	C Programming: Temperature control	
8	ARM Assembly Language Programming	Assembly
9	ARM Single-Cycle Processor	HDL & Assembly
10	ARM Multicycle Processor Control	
11	ARM Multicycle Processor Datapath	

In labs two and three the students build increasingly complex digital circuits, a seven-segment display and a finite state machine (FSM), using schematic entry and then simulating, synthesizing, and building their design in hardware on the DE2 board. We have found that it is critical for students to first create designs in schematic before entering their designs using a hardware description language (HDL). By so doing, the students become grounded in thinking of digital designs in terms of gates and, more broadly, in terms of combinational logic and registers. Without this foundation, students who begin their digital design by going directly to an HDL are at risk of viewing their HDL designs as software instead of thinking about the gates and hardware their HDL modules imply.

In labs four and five the students begin entering their designs using an HDL, either Verilog or VHDL. Lab 4 is a Thunderbird turn signal FSM. By proceeding from Lab 3, implementing an FSM in schematic, directly to Lab 4, students can clearly see the parallels between schematic entry and HDL entry. In Lab 5, a 32-bit ALU design, students also learn how to write an HDL testbench.

Labs six and seven introduce students to C programming using the Raspberry Pi, a single-board computer developed by the Raspberry Pi Foundation. This board includes the Broadcom BCM2835 system-on-a-chip (SoC), a processor based on the ARMv6 architecture. In Lab 6, students write three C programs: one that calculates the Fibonacci sequence, another that scrolls through lighting up the board's LEDs, and a third that is a number guessing game. These programs also interface with the Raspberry Pi's I/O (LEDs, switches, and the console). In Lab 7, the students write a temperature control program in C and build a custom temperature control circuit that they interface with the Raspberry Pi board.

In Lab 8, students practice their ARM assembly language programming skills by first writing an assembly program to calculate the Fibonacci sequence, one of the same programs they wrote in C in Lab 6, and then writing a second assembly program to compute floating point addition.

Labs 9-11 bring together the topics of digital design and computer architecture by guiding students in designing, building, and testing two simplified ARM processors in hardware. The students also write ARM assembly programs in both assembly and machine code to test their processors. In Lab 9, the students are given the HDL code for the simplified ARM processor that we discuss in lecture. That simplified processor performs the `ADD`, `SUB`, `AND`, `ORR`, `LDR`, `STR`, and `B` ARM assembly instructions only [5]. The students add their 32-bit ALU from Lab 5 to complete the design. After testing and examining the single-cycle processor using a provided testbench and ARM assembly program, they then expand the single-cycle ARM processor to include two additional instructions, `EOR` (exclusive OR) and `LDRB` (load register byte). They sketch their modifications by hand on the schematic from the textbook and then modify the HDL to include these new instructions. They then also translate an ARM assembly program into machine code, load it onto the processor in hardware, and write a testbench (i.e., modify the provided testbench) to determine whether the processor worked correctly.

In Labs 10 and 11, the students build on the knowledge they gained in all of the prior labs by building a simplified multicycle processor from scratch. Although the HDL design must be their own, they may use the building blocks provided in Lab 9 such as register files, memories, and multiplexers.

Throughout the labs the students also practice the guiding design principles of abstraction, modularity, hierarchy, and regularity. For example, throughout each lab, both hardware and software labs, students learn the importance of abstraction – that is, abstracting away unnecessary details – and modularity, having clearly defined interfaces and functions. In fact, as the labs increase in complexity, these design concepts become increasingly important. For example, a student using a multiplexer in Lab 5 must have, at that point, already gone through the work or completely understanding a multiplexer at the lower level and then be able to abstract away the details so that they can focus on its function. Abstraction, modularity, and regularity are also explicitly practiced as students build HDL modules that abstract away underlying details, have well defined interfaces, and that are then used and re-used in their design.

Table 3. Textbook Contents

1. From Zero to One	**4 Hardware Description Languages**	**7 Microarchitecture**
1.1 The Game Plan	4.1 Introduction	7.1 Introduction
1.2 The Art of Managing Complexity	4.2 Combinational Logic	7.2 Performance Analysis
1.3 The Digital Abstraction	4.3 Structural Modeling	7.3 Single-Cycle Processor
1.4 Number Systems	4.4 Sequential Logic	7.4 Multicycle Processor
1.5 Logic Gates	4.5 More Combinational Logic	7.5 Pipelined Processor
1.6 Beneath the Digital Abstraction	4.6 Finite State Machines	7.6 HDL Representation
1.7 CMOS Transistors	4.7 Data Types	7.7 Advanced Microarchitecture
1.8 Power Consumption	4.8 Parameterized Modules	7.8 Real-World Perspective: Evolution of ARM
1.9 Summary and A Look Ahead	4.9 Testbenches	7.9 Summary
	4.10 Summary	
2 Combinational Logic Design		
2.1 Introduction	**5 Digital Building Blocks**	**8 Memory Systems**
2.2 Boolean Equations	5.1 Introduction	8.1 Introduction
2.3 Boolean Algebra	5.2 Arithmetic Circuits	8.2 Memory System Performance Analysis
2.4 From Logic to Gates	5.3 Number Systems	8.3 Caches
2.5 Multilevel Combinational Logic	5.4 Sequential Building Blocks	8.4 Virtual Memory
2.6 X's and Z's, Oh My	5.5 Memory Arrays	8.5 Summary
2.7 Karnaugh Maps	5.6 Logic Arrays	
2.8 Combinational Building Blocks	5.7 Summary	**9 I/O Systems**
2.9 Timing		9.1 Introduction
2.10 Summary	**6 Architecture**	9.2 Memory-Mapped I/O
	6.1 Introduction	9.3 Embedded I/O Systems
3 Sequential Logic Design	6.2 Assembly Language	9.4 Other Microcontroller Peripherals
3.1 Introduction	6.3 Programming	9.5 Bus Interfaces
3.2 Latches and Flip-Flops	6.4 Machine Language	9.6 PC I/O Systems
3.3 Synchronous Logic Design	6.5 Lights, Camera, Action: Compiling, Assembling, and Loading	9.7 Summary
3.4 Finite State Machines	6.6 Odds and Ends	**Appendix A: Digital System Implementation**
3.5 Timing of Sequential Logic	6.7 Evolution of ARM Architecture	**Appendix B: ARM Instructions**
3.6 Parallelism	6.8 Another Perspective: x86 Architecture	**Appendix C: C Programming**
3.7 Summary	6.9 Summary	

The principle of hierarchy is also practiced as students build increasingly complex designs. For example, by dividing the final ARM multicycle processor design into two labs, Labs 10 and 11, the students learn to design with clearly defined interfaces and functions (modularity) and they practice and understand the use of hierarchy – building a processor from multiple layers of submodules and, perhaps most importantly, testing those submodules independently before combining them into the overall module.

Instructors who choose to run the course as a 2-semester sequence may either run the labs every other week or add other labs to give the students more practice with the material. For example, the first lab of building the 1-bit adder could be followed by another lab that builds additional simple digital circuits such as a majority circuit or a priority encoder. Likewise, the two finite state machine (FSM) labs, Labs 3 and 4, could be duplicated and then modified to include additional FSM implementations. In some iterations of the 2-semester version of the course, we have introduced a simpler FSM lab that preceded Lab 3, the Adventure Game FSM, and then also introduced a follow on FSM lab to Lab 4, the Thunderbird Turn Signal lab, so that the students gain more experience in building FSMs using an HDL. Additional C or ARM assembly language programming labs can also easily be introduced. In some iterations of our multi-semester course, we have also had the students follow Labs 9-11 with their building a pipelined processor.

The hardware required to support the labs are a DE2 (or DE2-115) FPGA board, an inexpensive Raspberry Pi board, and various electronic parts. The cost of these parts is listed in Table 4. Although the hardware labs are targeted to the DE2 or DE2-115 board, an institution that already has other FPGA platforms could readily adapt the labs to target that FPGA instead. The HDL programming portions of the labs would remain the same; only the instructions for targeting a given board would need to be changed.

Table 4. Laboratory Parts

Description	Cost
DE2-115 FPGA Board	$309 (academic price)
Raspberry Pi Model B	$35
Various electronics: 74xx parts, etc.	$1 - $5
Breadboard	Typically available

Even if an institution does not have the resources to purchase the hardware listed in Table 4, the majority of the labs can be completed using simulation as the final step. While the students would miss out on debugging

their design in hardware and learning from the hardware implementation, much of the learning goals can be met even without completing the hardware designs if necessary. However, if resources are available, the relatively inexpensive cost to support hardware implementation pays off as students transition to hands-on projects and to industry jobs.

The labs are supported by a TA who is available for 5-10 hours per week for a class of 45-60 students. The lab space can be run as an open lab, which is preferable if the resources are available, or as a scheduled lab with a fixed time for the students to complete the lab. The open lab format is preferable because it allows students to work on the material on their own and at their own pace and to hone their debugging skills. In that case, the TA hours are available to students typically after the students have tried it themselves first. The TA is then available to help them develop their design or debug their design or hardware.

VI. TEXTBOOK

The textbook that supports this course, *Digital Design and Computer Architecture: ARM Edition* [1], was written by the co-authors of this paper. In previous versions of the course [2][4], we have taught the material using another MIPS-based textbook, *Digital Design and Computer Architecture* [3], also written by this paper's authors. While the first half of the book, which focuses on digital design, is similar in both textbooks, the latter half of the new ARM edition focuses on the ARM processor instead of MIPS while still building on the same fundamental computer architecture principles as in the MIPS version.

The course transitioned to using the ARM processor because of that processor's prevalence in the current computer industry, particularly in embedded systems. ARM processors are in 95% of hand-held devices and are used world-wide.

The ARM computer architecture is also an interesting and timely case study because, although it is a RISC (reduced instruction set computer) architecture, it includes some features of CISC (complex instruction set computer) architectures that make it well-suited for embedded systems such as hand-held devices. In particular, the ARM architecture includes conditional execution and a large number of indexing modes, such as pre-indexing, post-indexing, and using an immediate offset or an optionally shifted register offset. These features increase the complexity of the hardware somewhat but enable smaller program size. ARM assembly programs are typically 75% of the size of other assembly programs, which makes them well-suited for the smaller memory sizes and the demand for lower power consumption of hand-held devices.

VII. FUTURE ENHANCEMENTS

Because the course offers hands-on instruction in designing, building, and testing digital circuits and in software (C and ARM assembly) programming, the tools that support these hands-on labs are often updated. However, forthe updates to the existing tools, such as compilers and FPGA design tools, are often incremental, so the lab instructions remain accurate. However, the lab instructions require incremental updates as the tools change.

Additionally, a parallel set of labs could be developed to include instructions for Xilinx-based FPGA boards, such as the Nexys4-DDR board, using the Xilinx design suite, Vivado.

VIII. CONCLUSION

The ARM architecture is a widely-used architecture in the computer industry, particularly in hand-held devices, and includes features such as conditional execution that enhance student understanding of both digital design and computer architecture. By including a broad range of learning tools, especially hands-on labs, the course proposed here enables students to learn digital design and computer architecture using the ARM-based processor as a case study. By showing students how to design an ARM processor from the underlying gates to the high-level architecture, students gain a thorough and clear understanding of processor design and the implications of design choices from both a hardware and software perspective.

IX. REFERENCES

[1] Harris, Sarah L., and Harris, David Money, *Digital Design and Computer Architecture: ARM® Edition*, Elsevier Publishers, May 6, 2015

[2] Harris, David Money, and Sarah Harris, *Introductory Digital Design and Computer Architecture Curriculum*, Microelectronic Systems Education Conference, June 2013, Austin, Texas.

[3] Harris, David Money, and Harris, Sarah L., *Digital Design and Computer Architecture*, 2nd Edition, Elsevier Publishers, August 1, 2012

[4] Harris, David Money, and Harris, Sarah L., *From Zero to One: An Introduction to Digital Design and Computer Architecture*, the First International Workshop on Reconfigurable Computing Education, March 1, 2006, Karlsruhe, Germany

[5] *ARMv4 Architecture Reference Manual*, ARM Limited, © 2005