EE 201: Fundamentals of computer systems and engineering

(aka, let's build a microprocessor from first principles)

Spring 2024
T/Th 1:30-2:45pm, Pearson 112

Welcome to EE 201!

Any sufficiently advanced technology is indistinguishible from magic. – Arthur C. Clarke

There are few technologies for which Clarke's statement is more true than microprocessors. Today you can find a microprocessor in almost anything electronic, and yet even those of us who can program them often have very little understanding of their inner workings. In EE 201, we will study microprocessors — and digital electronics more generally — from the ground up. From your past courses, you know how computers work at the physical level (voltages and currents) and at the application level (C++ and other languages); the objective of this course is to connect the dots between the two.

To do this, we begin with the basics of manipulating zeros and ones with circuitry, and start building *combinational* circuits, which produce outputs based on some mathematical combination of their inputs. Then we make a major turn and explore *sequential* circuits, which can store information or step through a sequence of states. With these building blocks in hand, we will examine the ARM instruction set (which powers your cell phone and a few billion other devices), and build circuits which can actually interpret and execute software instructions. Finally, we'll discuss popular techniques to make processors fast, including pipelining (executing multiple instructions at once in assembly-line fashion) and caching (saving important data nearby for faster access).

When we're done, you'll have built a working microprocessor from scratch, and implemented it on an FPGA. You'll also be able to create hardware to search through genomes, analyze and manipulate network traffic, mine bitcoin, or play retro video games. In short, you'll be able to work magic.

After successfully completing this course you will be able to:

- Use modern digital development tools including HDL synthesis tools and logic analyzers to implement and debug combinational and sequential logic circuits on an FPGA.
- Describe how a microprocessor works in terms of intermediate digital building blocks. That is, you should be able to explain to someone halfway through the course how a pile of logic gates can read a sequence of instructions and operate on a block of data to produce results.
- Explain the purpose of pipelining and caching, analyze the performance of a system using pipelining and/or caching, and design the hardware components necessary to implement such a system.
- Use memory-mapped peripherals on a microcontroller, and design memory-mapped hardware for timing, general-purpose I/O, and SPI or I2C communication.

How will this help me?

The skills from this course will help you in several different ways:

First, you will have the skills to design digial hardware which is capable of outperforming software on a conventional CPU by a factor of 100 or more (seriously!). Custom hardware is popping up in all sorts of places, from tensor processors in datacenters to neural compute engines in cell phones, and you will be poised to be part of this.

Second, you will walk away with a deep understanding of how a microprocessor works, because you'll have built one from scratch! You'll be able to read and decipher those thousand-page microprocessor datasheets

from Microchip and NXP, and you'll be able to describe how the hardware and software interact at the most fundamental level.

Even if you never design digital hardware in the future, a solid understanding of how the hardware works will enable you to write higher-performance code. And "performance" isn't just speed: power efficiency is incredibly important for wearable and some IoT applications, and this too requires a solid knowledge of digital hardware, even if you're "just" writing code.

Perhaps most importantly, you'll have more tools in your toolbox: you'll know exactly what you can do with a CPU, what you can do with an FPGA, and when to use each of them.

Where should I look for information?

The **course schedule and materials** will be posted on the course website: http://www.ece.tufts.edu/ee/201.

Course announcements and Q&A will be on Piazza. If you have a general question about the course content or course logistics, please post on Piazza rather than emailing me so that everyone benefits from the answer. Sign up for this course on Piazza here: http://piazza.com/tufts/spring2024/ee201.

Reading checks will be on Canvas.

The **textbook** is "Sarah L. Harris & David Money Harris, *Digital Design and Computer Architecture, RISC-V Edition*. ISBN: 978-0128200650

The paper version is available through the usual channels for about $80.

The ARM edition of the book is available online through Tisch Library, with no limit on the number of concurrent users. As far as I can tell, it is nearly identical through chapter 5.

There are some free supplemental resources for the book, including an electronic-only chapter 9: https://booksite.elsevier.com/9780128200643/

There are also two MOOCs available on EdX which have a video lecture for each section of the book, which you may find helpful:

- https://www.edx.org/learn/design/harvey-mudd-college-digital-design
- https://www.edx.org/learn/computer-architecture/harvey-mudd-college-computer-architecture

Access is free for the duration of the MOOC (which is a little shorter than our semester, but long enough).

Assigment submissions and feedback will be done through Gradescope and VHDLweb (and possibly Github). We'll provide the signup links/credentials once there are assignments to complete!

There are several computer games that cover large portions of the course content. The best one I've found so far is **Turing Complete** (https://turingcomplete.game). At $19.99, it's way cheaper than a textbook and a lot more engaging!

There are also a number of great websites and videos about digital design. Some of these are posted in the "possibly helpful" and "just for fun" sections on the course website, but there are many more. If you discover other resources which are particularly helpful, please share them so we can all benefit!

Communication:

Steven Bell `sbell@ece.tufts.edu`

Curtis 001-C (Directions: http://www.ece.tufts.edu/en/1EK/finding_my_office.html)

Office hours:

- Tuesday and Thursday after class

- Wednesdays 1-4pm, Nolop makerspace
- I'm also available other times by appointment, just email me and we can find a time!

To minimize distraction, I generally only check email a couple times a day. However, I will make a strong effort to answer all messages within 24 hours on weekdays.

Prerequisites:

There are no prerequisites for this course except graduate standing. Some familiarity with C/C++ or assembly programming (such as CS 40 or EE 200) will be helpful.

Exams:

There will be one midterm assessment, tentatively scheduled for the week of March 6.

Our scheduled final exam slot is Tuesday, May 7 2024, 3:30-5:30pm (H+ block).

The format of these will be determined as the course progresses, but is likely to include some independent design work (e.g., create a circuit to compute a certain mathematical function) followed by a brief in-person discussion about your design.

Grading:

Grades will be assigned on an absolute scale (not curved), with the following components:

Readings and pre-class quizzes: 10 %

Homework: 60 %

Exams: 30 %

Homework:

There will be homework assigned roughly every week, and you'll have at least a week to work on each assignment. Homework will be a mix of traditional pen-and-paper problems, VHDL coding challenges where you build components of your microprocessor, and "lab" problems where you implement VHDL designs on an FPGA and wire up circuitry to support your design. The homework culminates in building and programming your own microprocessor.

Late policy:

Unexpected things happen during the semester, so if you need an extension on an assignment, send me an email **before** the deadline, and let me know when you will submit the assignment. I will grant all reasonable requests, but I will hold you to the deadline you set for yourself. In general, "reasonable" means not more than 3 days, except in the case of major illness or family emergency.

Laptops and phones in class:

I aim to make our classroom an interactive and collaborative environment. To facilitate this, all "devices" (cell phones, laptops, tablets, Palm Pilots, Furbys, GameBoys, etc.) should be put away during class, except when you are asked to use your device for an online quiz or poll.

Academic integrity:

One question guides decisions about academic integrity in this course:

Does this *help* me learn, or is it a way to *avoid* learning?

Collaboration: I encourage you to collaborate with your classmates on the homework assignments. But just like physical exercise, those who put in the work get the gains. Thus, **the work that you turn in must be your own**.

- For written work, you're welcome to collaborate on a whiteboard or other shared thinking space, but each student needs to write up their own solution. When you're done, you should be able to work a similar problem on your own.
- For code, you may work on the problem together, discussing approaches or looking at each other's code to debug errors. However, you should not be copying code, either by copy-pasting or by typing from another screen.
- For FPGA implementations, every student needs to get their own implementation of the lab project working, whether it is a circuit or FPGA design. You're welcome to sit side-by-side and build the circuits together, but you cannot borrow someone else's circuit or copy another student's code.

Internet resources: You can (and should!) look up related resources on the internet, but again they should be to supplement your learning, not replace your thinking. For example, looking up resources on how to do Boolean logic simplification is excellent, but using a simplification tool to solve the homework problems is not. Searching a VHDL error message for possible solutions is great; searching for the complete solution to a coding problem is not.

Generative AI: While generative AI systems such as chatGPT and Github Copilot have enormous promise to amplify a skilled developer's ability to write code, the emphasis in this course is on building your core digital design skills. **It is not acceptable to use an AI tool to generate code for any assignment**, even as a partial solution or starting point. These systems also have a tendency to generate utter nonsense in a very confident tone, which makes it difficult to discern whether a generated solution is even useful as a reference.

Any code you submit may be run through plagarism-checking software, and I may use AI-checking software on code or text. By the terms of my employment with Tufts, I am required to report any suspicion of academic misconduct to the student affairs office.

ADA accomodations:

If you need special accommodations (extra time on exams, larger type on handouts, etc.), please initiate the process with Tufts Student Accessibility Services (http://students.tufts.edu/student-accessibility-services/accessibility-policies-procedures). Accomodations cannot be granted retroactively, so please do this sooner rather than later.