Reimagining the digital lab with $30 FPGAs

Steven Bell, Tufts University

Steven is an Assistant Teaching Professor in Electrical and Computer Engineering at Tufts University, where he teaches a mix of courses including digital design, introduction to engineering, and embedded systems. He has a BS in Computer Engineering from Oklahoma Christian University, and MS and PhD in Electrical Engineering from Stanford University.

Reimagining the digital lab with $30 FPGAs

Introduction

Introductory digital logic is one of the most exciting courses in a typical electrical or computer engineering program — or at least it should be! Coming into the course, students have both learned to code and to analyze basic electrical circuits, but the workings of a computer remain a mystery. Digital logic is our opportunity to connect those dots by showing how a collection of circuit elements can execute code, and by enabling students to build systems with their newly developed engineering skills that they hardly could have dreamed of at the beginning of the semester. At its best, the digital logic course pulls back the curtains on the magic of microprocessors, and invites each student to imagine becoming the magician themselves.

The course also occurs at a critical point in the curriculum: at Tufts, students take the course (ES 4) in the fall semester of sophomore year and it forms part of their core conception of what electrical and computer engineering is. In general, their courses up to this point have been generic across engineering, and many students see the course as a way to confirm whether an electrical or computer engineering major is right for them. As a result, we have both an opportunity and an obligation to inspire and motivate students in addition to helping them develop prerequisite skills for other courses.

Digital logic labs

As at most universities, our offering of the course has a substantial laboratory component, where students put in the hard (and rewarding) work of translating pencil-and-paper logic designs into working systems.

There are several traits we desire in a digital design laboratory:

- First, labs should be *tangible*; concrete rather than abstract. Tangible does not necessarily imply that students can touch the result, but ideally the logic design would exist in the real world and be subject to its constraints. We would like to probe it with normal lab equipment and perhaps even connect it to other bits of real circuitry. Even better if students have the feeling of creating something they can show off to others.
- Second, labs should be *accessible*, meaning that the necessary workspace and development environment can be replicated easily and inexpensively. In the best case, the labs would be completely portable, allowing students to work anywhere. This was critically important during the pandemic, but it continues to have benefits. Tools like the Arduino microcontroller boards are popular in large part because they are accessible: the software is free and runs on nearly any computer, no lab equipment or additional circuitry is required, and the boards themselves are inexpensive.
- Third, labs should represent *professional practice*. Students should gain experience working with professional tools and techniques. This is obviously important in terms of preparing students to work in industry, but it is also a critical factor for motivation: students want to know that their work is not merely a toy exercise, but they are using professional-grade tools and can put their course experience on their resume.

These objectives are often in tension, and achieving all of them is a tall order. The remainder of this section surveys the popular approaches to digital labs and discusses their relative strengths.

The historical choice for digital design labs was discrete 74-series logic chips on a breadboard. Although these chip lines are nearly 50 years old, they remain an important contender for introductory labs.

74-series logic chips provide a deeply tangible learning experience. A NAND gate is not just a truth table, it's something you can hold in your hand. Inputs and outputs are provided by real circuit components. Power and ground are not invisible, they have to be wired up. Nothing is a black box; most of the circuitry is exposed on the breadboard and the chip datasheets generally show the internal workings down to individual gates if not individual transistors. Every logic signal can be probed, gate delay can be tested, and power consumption measured.

Given that our course emphasizes building complex logic from scratch, discrete logic chips provide excellent emphasis on fundamentals. However, it has been decades since digital logic systems were actually built like this. While design with discrete logic chips may still be pedagogically useful, it no longer represents professional practice.

A second approach is to use graphical simulators, such as CedarLogic [1] and LogiSim Evolution [2]. Because these tools allow direct editing and simulation of a logic diagram, no mental translation is needed from the diagrams and models used in the textbook or in class. The simulation directly reinforces the logic diagram models, making it a useful tool for investigating and understanding course concepts. The simulation can be single-stepped or paused at will and any signal can be inspected, making it easier to get an intuitive sense of how the circuit is functioning (or malfunctioning!).

Moreover, because most of these tools are lightweight and cross-platform (and in many cases free), it is straightforward for all students to install the software and run it in class, in the lab, or remotely. Some are even web-based [3], reducing the installation burden to zero and making it easy for instructors to share starter designs or examples.

But these strengths are also weaknesses. While the simulation is far more portable than a box full of breadboards and chips, it is also less tangible. Power supplies are generally abstracted away, along with real-world issues such as timing, clock frequencies, and metastability. It is generally not possible to interface with real peripherals, so the simulator's I/O tends to be restricted to some simulated buttons and LEDs.

And while the visual nature of the logic diagram is a powerful pedagogical tool, it does not represent professional practice any more than filling a dozen breadboards with logic chips: digital logic design today is primarily done with hardware description languages (HDLs).

Accordingly, a third approach to the intro digital lab is to use HDL simulation. This is traditionally (System)Verilog or VHDL, although other languages are sometimes used (such as the custom language used by the "NAND to Tetris" tools [4]). This directly exposes students to professional practice, which prepares them for future courses, internships, and full-time jobs.

The downside is that this is even less tangible than graphical simulation. Inputs and outputs simply become testbench variables, and intermediate signals become digits (1 and 0) rather than voltages. Almost any sense that there is an actual circuit at play is lost.

This is a perfectly reasonable abstraction for practicing experts, but it can be hopelessly confusing for novice learners. Specifically in our context, nearly all of our students have some software development experience coming into the course, so they naturally equate "code" with "software". Without some tangible experience to ground the abstraction in actual circuitry, students easily slip into treating the HDL as nothing more than an obtuse software language with very frustrating behavior.

To bring back the tangible element, the final popular approach is to design with an HDL and implement it on an FPGA. This pushes professional practice one step further, giving students experience with professional-grade toolchains and the FPGA design flow. It opens opportunities for students to learn about PLLs, timing closure, metastability, and other essential aspects of implementing a design.

Sadly, FPGAs also introduce enormous overhead, pedagogically and practically. Most FPGA development boards cost a few hundred dollars apiece, and the toolchains from the two leading vendors have hefty system requirements and only run on Windows or Linux. Datasheets run into the thousands of pages, making them impenetrable for all but the most fearless of students.

And while typical FPGA development boards have nearly unlimited capabilities with a wide array of built-in peripherals and pluggable PMOD modules, this also means that students rarely get to make their own design choices and the peripherals become part of the black box. As a result, students' first experience with an FPGA is less like a playground sandbox and more like a guided nature walk through the scary woods: you'd better stay on the trail or risk getting very lost.

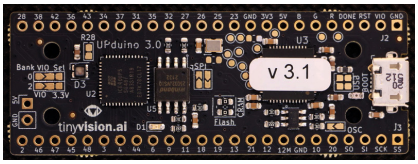
Each of these lab models has important benefits, pedagogically and practically. The following section introduces low-cost FPGAs, which combine many of the benefits of these different approaches.

Low-cost FPGAs

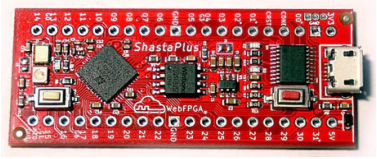
Not long ago, "Low-cost FPGA" was an oxymoron. That is no longer the case: There are now numerous FPGA development boards under $50, including the UPduino 3.1 [5], WebFPGA [6], and the tinyFPGA family [7].

Lattice Semiconductor defined this category with the iCE40, a lineup of simple and low-power FPGAs with a few thousand logic elements and a few dozen I/O pins [8]. The IceStorm project [9] publicly reverse-engineered the iCE40 architecture and bitstream format, and developed a complete open-source toolchain from Verilog synthesis through bitstream flashing. The combined result has been a proliferation of open-source development boards and and substantial user community around the iCE40.

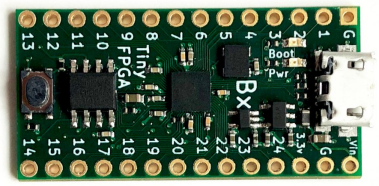
In the introductory digital lab setting, these low-cost FPGAs achieve the trifecta of tangibility, accessibility, and professional practice. In our offering of ES 4, we have used a sequence of UPduino boards, starting with the UPduino 2.0 from Gnarly Grey (now discontinued) and continuing with the UPduino 2.1, 3.0, and 3.1 from TinyVision.ai.



(a) UPduino 3.1
($30, TinyVision.ai)



(b) WebFPGA ShastaPlus
($38, WebFPGA.io)



(c) TinyFPGA BX
($38, TinyFPGA.com)

Figure 1: Several breadboardable FPGA development boards based on the Lattice iCE40 series of FPGAs.

Because the UPduino plugs into a standard breadboard, it retains much of the tangibility of 74-series logic. Students wire up switches and LEDs as I/O, making choices about pin assignments and whether signals are active high or active low. Arbitrary digital peripherals can be easily connected, from rotary encoders to seven-segment displays to MIDI keyboards. Signals can easily be probed with an oscilloscope or logic analyzer.

Second, the UPduino and its cousins enable uniquely accessible and portable labs. With a price point between $20 and $50, it becomes feasible for every student to have their own FPGA. It is also possible for students to run the toolchain on their personal computers. Lattice Radiant (the Lattice-supplied toolchain) has a free license for iCE40 devices [10], and the IceStorm toolchain is open-source and runs on Windows, Mac, and Linux. Radiant requires an order of magnitude less hard drive space than the Xilinx and Intel toolchains, and the IceStorm tools are even lighter. Add to the UPduino a simple lab kit containing a breadboard and a handful of digital circuit components, and it suddenly becomes possible for students to take their projects anywhere and work on them outside of the lab.

Finally, low-cost FPGAs enable professional practice. The iCE40UP5K powering the UPduino has 5280 logic elements, enough to implement interesting designs such as ARM or RISC-V processors [11], neural networks [12], audio synthesizers, or arcade games. Students develop their design with Verilog or VHDL (or the high-level HDL of your choice, such as Chisel, SpinalHDL, myHDL, etc.) and work through the complete synthesis/P&R/bitstream flow. Like larger FPGAs, the iCE40 contains block RAMs and a PLL, so students can experiment with inferring memories and managing clock distribution and timing.

At the same time, the iCE40 chips are simple enough that even introductory-level students can get a general grasp of how the FPGA works internally. In contrast to FPGA datasheets from the leading vendors, the iCE40 datasheet is a mere 54 pages and reasonably comprehensible. We show excerpts from the datasheet in class, demonstrating that the FPGA isn't just a magical device for implementing arbitrary logic — it is itself a digital circuit, which can be fully investigated and understood with the skills from the course!

To borrow our earlier analogy, the iCE40 is not quite an idyllic sandbox of exploration and free-form construction, but it's close. The following section describes our course in more detail, and then discusses how the UPduino FPGA enables a new paradigm in our lab exercises.

Implementation in an introductory course

The purpose of the introductory digital design course has subtly shifted over the past thirty years or so. Today, a 32-bit Cortex M processor costs less than a 7408 AND gate, and processors have taken over an enormous number of tasks for which we would have previously designed custom logic. Accordingly, only a very few of our students will go on to *design* digital logic in their careers — but many of them will be programming embedded systems or otherwise writing code that interacts with hardware at a low level.

As a result, our course is focused on digital design as a link to computer architecture, and not as an end in itself. We use the ARM architecture[1] as a reference, showing students that the concepts they are learning apply both to the Cortex-based embedded systems they will use in the following semester and to billions of computing devices deployed in the real world.

We use the ARM edition of the textbook "Digital Design and Computer Architecture" [13], which succinctly presents digital design fundamentals and then builds up to a simple microprocessor for a subset of ARMv7 instructions.[2]

Figure 2 shows a map of our path through the course as we travel from the basics of 0 and 1 all the way to building a processor. Our journey takes a number of twists and turns, so we refer to this map frequently in class to orient ourselves and stay focused on the big picture.

We begin with combinational logic, and after spending just enough time skirting the edge of "how things used to be done" (Boolean algebra and Karnaugh maps) we introduce VHDL.[3] Next we introduce flip-flops and sequential circuits, which is the point where the course gets difficult. Then we take a second pass through VHDL to discuss `process` blocks and the implementation of sequential logic. From there, we begin building larger pieces which ultimately are assembled into a computer.

As Figure 3 shows, the labs follow a day or two behind the lecture content. Because labs generally run four days a week and prelabs are due before students arrive in lab, we need a few days of buffer between students' first exposure to the material and using it in lab.

Lab kits

Every student receives a lab kit containing the FPGA along with a breadboard, USB cable, and all of the discrete components they will need to complete the labs. The kit fits in a small container about the size of a pencil box, which is easy for students to carry and bring each week to lab. The box is also just deep enough for an assembled circuit to be packed away and remain intact. An itemized list of the lab kit contents is included in the appendix.

The cost of the lab kit is approximately $15, plus the cost of the FPGA (which has ranged from $15 to $30). In our case, we are fortunate that the department covers this expense, but it could also be defrayed by requiring students to pay a small lab fee or to return their materials at the end of the semester.

[1] Technically ARMv7, but at the level of detail we cover in the course, the differences between the various 32-bit versions are irrelevant.

[2] With the growing popularity of RISC-V in industry and the release of a RISC-V edition of Harris & Harris [14], this is a compelling alternative.

[3] We use VHDL simply because other courses at Tufts use VHDL.

Figure 2: Map of the course, showing the progression through combinational logic, VHDL, sequential logic, and finally toward the fair lands of computer architecture. Inspired by Elecia White's "Memory map" [15] and created with Inkarnate.

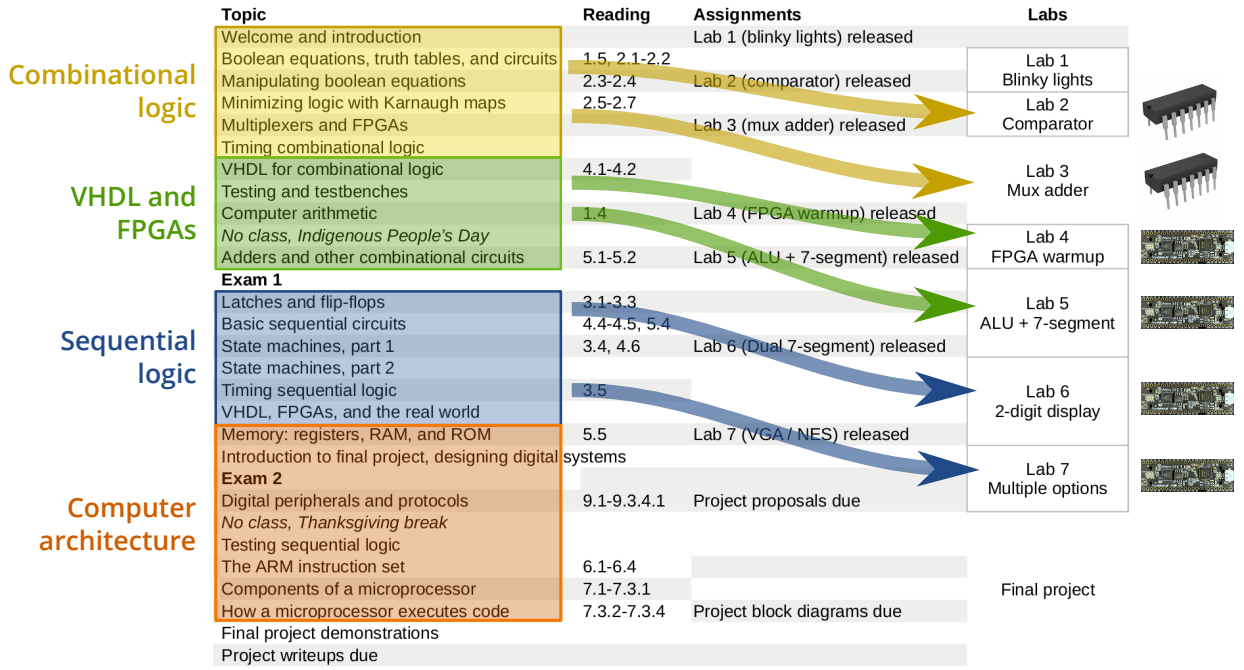


Figure 3: Flow of the course over the semester, and the correspondence of class topics and weekly labs. Readings are from [13].

Lab sequence

Lab 1 introduces students to the lab equipment (power supply and multimeter) and the process of building circuits on a breadboard. Since many students have never used a breadboard before, this lab is structured as a tutorial with detailed instructions interspersed with small investigations.

In **labs 2 and 3**, students design combinational circuits and construct them using discrete gates on a breadboard. Lab 2 asks students to build a circuit which compares two 2-bit numbers entered with a DIP switch, and light up an LED if $A > B$. This problem is approachable with only the first week's worth of course material (either by using intuition, Boolean algebra, or a Karnaugh map). At the same time, it has a basic (if simplistic) real-world meaning and multiple solutions.

In lab 3, students build a 2-bit adder using 8:1 multiplexers by wiring each mux input to a constant high or low value to create a 3-input look-up table. This is intended both to give students practice reading datasheets and working with a more complex logic component while helping them form a conceptual bridge to understand how an FPGA can implement arbitrary logic.[4]

During the Spring 2021 offering of the course, we replaced labs 1–3 with an FPGA-based alternative which was more amenable to remote work. Instead of discrete chips, students used Icestudio [16], a graphical logic design tool that targets iCE40 FPGAs. Icestudio allows a user to

[4]Whether it does this successfully is questionable. Since students do not see or use LUTs directly once we start using the FPGA in earnest, the other end of this conceptual bridge is never properly constructed.

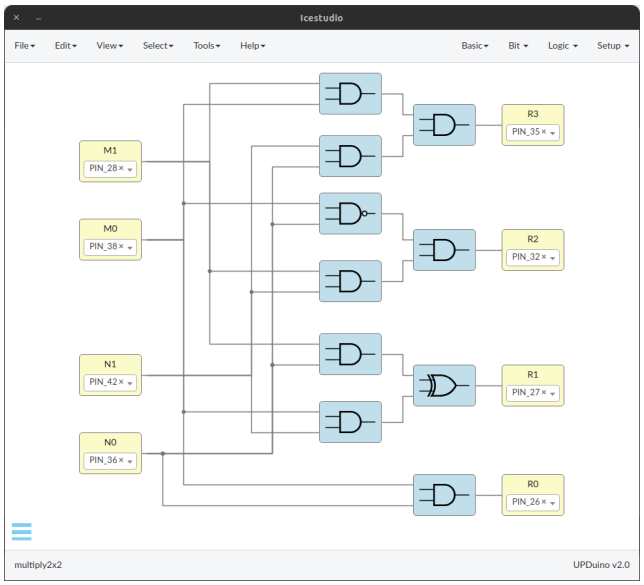


Figure 4: Example solution for lab 2 (2-bit multiplier) created in Icestudio. The yellow blocks represent FPGA I/O pins, while the blue blocks and wires represent the logic implementation.

draw a logic block diagram much like the logic simulation tools described earlier, except that the inputs and outputs are mapped to pins on the FPGA. Icestudio can then synthesize the design and flash it to the UPduino with the click of a button.

This had the immediate advantage that no additional lab equipment was needed, and students could complete the labs in their dorms, in quarantine, or at home on the other side of the world. Moreover, since the design existed primarily on a computer screen with only the I/O on the breadboard, it was relatively easy to debug via screensharing. It also got students used the FPGA's physical interface and design flow (design, synthesize, flash), which slightly simplified the transition to Radiant in later labs.

Because wiring up a logic diagram on a computer screen is substantially faster than physically constructing a circuit, students were able to complete larger (and therefore slightly more useful and interesting) circuits. For lab 2, students built a 2-bit multiplier (giving results between decimal 0 and 9) by writing a logic equation for each of the four output bits in terms of the four input bits. For lab 3, we extended the 2-bit adder to 3 bits.

There were also disadvantages to the Icestudio approach. Because the design exists entirely within the FPGA, intermediate signals cannot be immediately probed and debugging is a little more difficult. It is still possible to tweak the design and break out intermediate signals to output pins, but this extra step biases students toward visually checking their design over and over hoping to find an error rather than following a rational debugging process. Icestudio itself had a few frustrating quirks and we sometimes spent as much time debugging those as actual circuit issues.[5] For these reasons, we went back to using 74-series chips for Fall 2021, but Icestudio — and this approach more generally — remains a compelling option for labs.

[5]We used Icestudio version 0.6, and there have been many bugfixes and improvements since then. We have not tested it extensively since the 0.6 release.

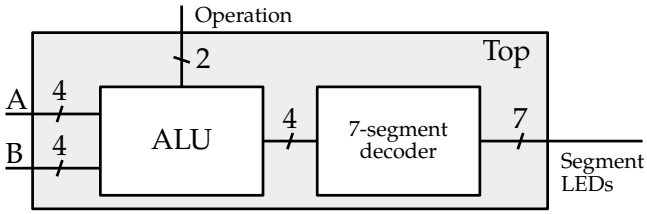


Figure 5: Block diagram of lab 5, an ALU and 7-segment decoder.

Lab 4 introduces students to the FPGA, in the same way that lab 1 introduces them to the benchtop equipment. By this time, students have already completed a number of VHDL exercises and are familiar with the basics of the language, so the goal of the lab is simply to walk them through the tool flow, starting with an empty project and working all the way through flashing the FPGA.

In **Lab 5**, students begin building substantial designs with VHDL and their FPGAs. They implement combinational modules for a 4-bit, 4-operation ALU and a 7-segment decoder, and then link these together to produce the complete design. For the hardware, they must wire up a DIP switch to provide the ALU operands and the seven-segment display to show the result.

Lab 6 is the first project to use sequential logic: students extend their work in lab 5 to two seven-segment displays by multiplexing the displays and using persistence of vision.

This requires instantiating the onboard 48 MHz oscillator, dividing it down to a suitable frequency for the display (on the order of 100 Hz), and writing combinational logic to switch between the digits as shown in Figure 6.

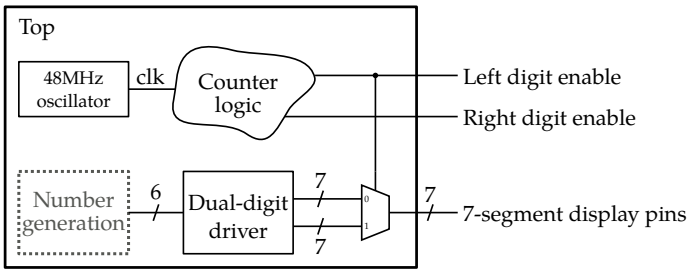


Figure 6: Block diagram for lab 6, which drives a multiplexed dual seven-segment display using persistence of vision. Students can choose how they want to generate numbers to feed into the display.

For **Lab 7**, students choose one of the following options:

- A controller to drive a standard VGA monitor at a resolution of 640x480 with 6-bit color. This requires instantiating the onboard PLL to obtain the VGA pixel clock frequency, plus counters to track the horizontal and vertical position and logic to generate the HSYNC and VSYNC signals. A small breakout board with 3 resistor DACs is used to drive the color signals (design link included in the appendix).

- The interface for an NES gamepad. The original NES controller essentially connects its 8 buttons to a shift register, so the students' task is to trigger the latch signal and transmit 8 clock pulses while shifting data into the FPGA.
- An animation using frames stored in the FPGA's embedded ROM. This project explores how to infer a ROM (and how to read the netlist diagrams and utilization reports to confirm it actually worked!), and how to generate addresses on the fly with a counter.

We are actively working on developing more options for lab 7 to cover additional peripherals (PS/2 keyboards and I2C devices) and FPGA features (e.g,. RAM).

Final project
For their final projects, small teams of students build interactive projects with any number of real peripherals, including PS/2 keyboards, NES gamepads, GuitarHero controllers, VGA displays, MIDI keyboards, I2S audio amplifiers, and more.

The project is very open-ended — our only hard requirement is that it use the FPGA to read some sort of input, do something interesting with it, and produce some output. However, we encourage students to build an arcade game: what took an entire cabinet of digital logic (and some very experienced engineers) fifty years ago now easily fits on the FPGA and can be accomplished in a few weeks. Each year we have also had teams build audio synthesizers, and a couple teams have worked on cryptographic accelerators. Students submit a brief proposal and we work with them to refine the idea and scope to something practical. We also assign a TA to be their first contact and advisor for the project; generally this is a student who completed a similar project in a previous offering of the course.

The course culminates in a demonstration party where we set up all of the games in an open room, bring in some food, and invite friends and colleagues to come play for an hour. Figure 7 shows a sampling of some past projects.

During the Spring 2021 semester, we offered an alternative final project option where teams built a processor to execute a basic subset of ARMv7 instructions. This required teams to work a few days ahead of the class topics, but could be completed by remote or distributed teams.

We supplied a set of handouts that described the interface for each functional component, along with a guide suggesting a sequence for building, testing, and integrating the components into a complete processor. Although the processor's functionality could be verified entirely in simulation, several teams went as far as synthesizing their processor in Radiant and testing it on the FPGA.

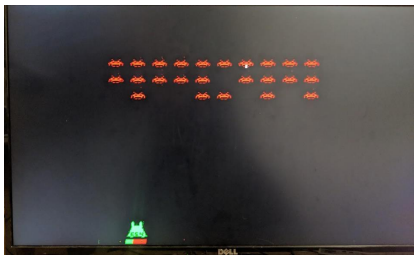
Response

Starting with 0 and 1 and working all the way up through VHDL and FPGAs to a functional microprocessor is a tall order for a single semester, but we have repeatedly found that students are more than capable of meeting the challenge. Each year, student teams turn out impressive projects with hundreds of lines of functional (if often inelegant) VHDL running live on their FPGAs.

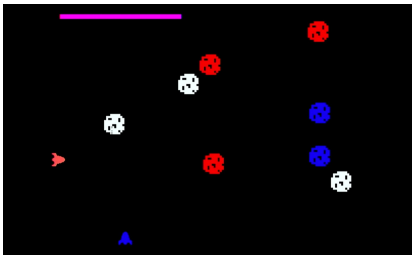
Learning the digital design content will always require focused effort, but we believe that effective design tools can dramatically reduce the cognitive load on students. We introduce

(a) “Smack Buds” platformer

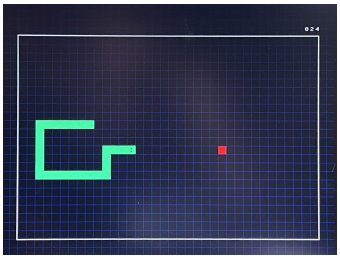
(b) Atari racer



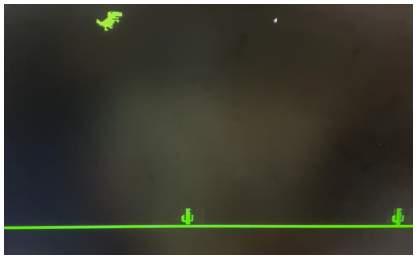
(c) Space Invaders



(d) 2-Player asteroids



(e) Snake



(f) Google Chrome dino game

Figure 7: Sampling of some final projects from the past few years.

VHDL using a web-based simulation environment developed specifically for the course, which allows us to do VHDL exercises in class and for homework without any of the traditional overhead of learning to navigate byzantine simulation tools. Likewise, the UPduino provides an entry point with a straightforward physical architecture and accessible development tools, reducing much of the overhead of traditional FPGA boards and allowing students more time and mental resources to focus on their designs.

To help gauge the value of the lab kits, we sent an optional survey to the students who were in the three most recent offerings of the course. Of the 30 students who responded, 67% indicated that they did some sort of work on their lab projects outside of the lab, such as modifying the circuit or flashing their FPGA. All but one student (97%) indicated that they used their lab kit and projects outside the lab as a reference while writing their lab reports.

Overall, we succeeded in capturing the interest of our target audience — and then some. After redesigning the course, we began to have interest from students outside of ECE and CS, as shown in Figure 8. These have included students from mechanical engineering, biomedical engineering, and engineering physics.

Additionally, we have drawn a significant number of computer science majors. The CS department dropped the course as a hard requirement for CS majors in 2021, and instead made it one option out of about eight “systems electives”. Nonetheless, we retained a large number of CS majors.

Every year, we allow students to keep their FPGAs if they “think they will do something interesting with them.” Only a very few students return them, suggesting that the vast majority of students see value in what they have learned and can imagine using their digital design skills in the future. However, most students report not having used them for any projects after the course,

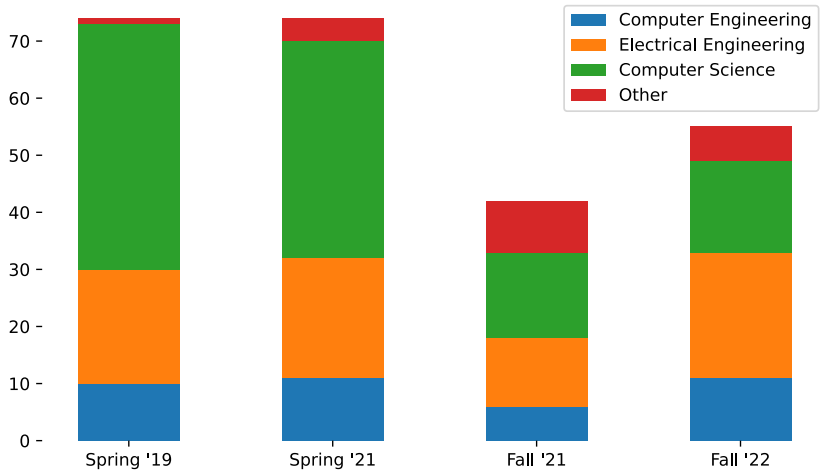


Figure 8: Enrollment for the course since the FPGA-based redesign. We experienced a jump in enrollments from non-majors, and CS enrollment remained strong even after ES 4 was removed as a CS degree requirement in 2021.

so perhaps we still have a ways to go!

Lessons learned and future directions

The UPduino hardware itself has performed very well. Version 3.0 fixed silkscreen and signal integrity issues with the 2.0 version, and the UPduino 3.1 includes a built-in PTC resettable fuse on the power supply in response to our students burning up boards by shorting power and ground. Failures are now very rare; we generally only experience one or two genuine FPGA failures out of 50-80 students.

Supporting Macbooks continues to be a challenge. Our initial solution was to provide a Linux virtual machine which every student could run with VirtualBox or VMWare, regardless of their host operating system. This established a uniform environment with Radiant and other software tools, but many computers had trouble with the virtual machine and some of these were unable to be resolved. When the ARM-based Macbooks arrived in late 2020, an x86 virtual machine was no longer feasible, making the open-source toolchain the only option on Macbooks. We are currently developing a browser-based IDE (similar to WebFPGA [6]) which we hope will provide a cross-platform solution with no installation requirements.

As with any course, there is never time to explore every possible avenue that we would like. One weakness of the final projects is that students do not have practice breaking large designs into modular pieces, and tend to pile nearly all of their logic into one or two modules. They also tend to use questionable constructs (such as multiple clocks or negative-edge-triggered flip-flops) which make sense theoretically but do not produce reliable designs.

We have only begun to tap the potential of the “FPGA everywhere” lab model. One of the prelab exercises invites students to test their code on the real FPGA before coming to lab, but there are

many more opportunities to do this in prelabs, homework, and perhaps even live in class. Likewise, while a majority of students already do some work on their lab projects at home, we imagine a time when FPGA development is as accessible and portable as microcontroller development. We are continuously iterating, and we look forward to what low-cost FPGAs will enable in the future.

Acknowledgments

I am grateful to numerous undergraduate and graduate TAs who have enthusiastically helped develop parts of the course, worked tirelessly with students in the lab, and stayed up until unreasonable hours of the night helping with final projects. This course would not be possible without them.

Special thanks to Venkat Rangan (TinyVision.ai) for creating and supporting the UPduino 3.0/3.1, and for supplying us with many boards despite ongoing supply chain issues. I am also thankful to Lattice for making licensing a simple process, and occasionally providing technical support when we encountered issues with Radiant. I have no affiliation with either TinyVision.ai or Lattice other than being a satisfied and enthusiastic customer.

References

[1] J. Richardson, K. Shomper, M. Lewellyn, B. Sprague, and C. Kohl, "CedarLogic Digital Logic Simulator," https://github.com/CedarvilleCS/CedarLogic.

[2] Logisim Evolution Developers, "Logisim Evolution," https://github.com/logisim-evolution/logisim-evolution.

[3] CircuitVerse Contributors, "CircuitVerse Digital Logic simulator," https://circuitverse.org.

[4] N. Nisan and S. Schocken, *The Elements of Computing Systems: Building a Modern Computer from First Principles*. MIT Press, 2005.

[5] V. Rangan, "UPduino v3.1," https://tinyvision.ai/products/upduino-v3-1.

[6] R. Jacobs, "WebFPGA," https://webfpga.io/.

[7] L. Valenty, "TinyFPGA," https://tinyfpga.com.

[8] Lattice Semiconductor, "iCE40 – Low-power, High-performance FPGA," https://www.latticesemi.com/Products/FPGAandCPLD/iCE40.

[9] C. Wolf and M. Lasser, "Project IceStorm," http://bygone.clairexen.net/icestorm/.

[10] Lattice Semiconductor, "Software licensing," https://www.latticesemi.com/Support/Licensing.

[11] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The MicroRV32 framework: An accessible and configurable open source RISC-V cross-level platform for education and research," *Journal of Systems Architecture*, vol. 133, p. 102757, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122002429

[12] G. G. Lemieux, J. Edwards, J. Vandergriendt, A. Severance, R. De Iaco, A. Raouf, H. Osman, T. Watzka, and S. Singh, "TinBiNN: Tiny binarized neural network overlay in about 5,000 4-LUTs and 5mw," *arXiv preprint arXiv:1903.06630*, 2019.

[13] S. L. Harris and D. M. Harris, *Digital Design and Computer Architecture, ARM edition*. Morgan Kaufmann, 2015.

[14] S. L. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V edition*. Morgan Kaufmann, 2021.

[15] E. White, "Memory map land image," https://github.com/eleciawhite/MapFiles/, 2021.

[16] FPGAwars community, "Icestudio," https://icestudio.io/.

Appendix: Part list

All of our course materials are available on our course website: http://www.ece.tufts.edu/es/4

Items in the lab kit (1 kit per student)

Item	Quantity	Part number/source
UPduino 3.1	1	TinyVision.ai
Micro USB cable	1	
830-tie breadboard	1	
Jumper wires	60	
LED, assorted colors	12	
8-way DIP switch	1	
Dual 7-segment display	1	Lite-ON LTD-4608
Tactile pushbutton switch	4	
Resistors for LEDs	20	330 Ω or similar
Pull-up resistors	10	10k Ω or similar
Quad AND	1	74HC08
Hex inverter	1	74HC04
Quad OR	1	74HC32
Quad XOR	1	74HC86
8:1 Multiplexer	4	74HC151
Project box	1	

Due to electronic component supply chain issues, the exact part numbers have changed from year to year. For the 74-series chips, we have replaced these parts with similar ones which allow students to complete the same projects with slightly different designs. For example, the 2022 kit included a 74HC00 quad NAND, 74HC4075 3x3 OR, and 74HC266 quad XNOR in place of the AND, OR and XOR listed above.

Shared items provided in lab

We typically allocate one of each of the following per lab bench (10 students working simultaneously during a lab section), and about 1 NES and VGA setup per final project team.

Item	Cost each	Part number/source
USB-C adapters (for laptops with only USB-C)	$2	Amazon or Ebay
Digital Discovery logic analyzer	$229	Digilent
NES controllers	$7	Ebay
0.1” Dupont connectors	bulk	Ebay or Digi-Key
VGA breakout boards	$5	OSH Park https://oshpark.com/shared_projects/u8Pw1fpw
VGA cables and monitors	—	Borrowed from lab
VGA to HDMI converters	$10	Amazon or Ebay
USB HDMI capture dongles	$10	Amazon or Ebay

In the fall of 2021, many of our lab computers were replaced with “all-in-one” machines which did not have a dedicated VGA input. Our fallback solution was to use a VGA→HDMI converter plus an HDMI capture dongle which plugs into a standard USB port and appears to the computer as a webcam. Although direct VGA→USB capture dongles exist, they cost several times as much ($100 or more).

The downside of this approach is that the converters and webcam preview software introduce noticable latency in the video signal. Some games were still playable; others were not.