Digital Logic without Compromise in a Quarter-Based EE Curriculum

Dr. Mehmet Vurkac, Seattle University

Mehmet Vurkaç is an assistant professor in the Department of Electrical and Computer Engineering at Seattle University.

Dr. Margarita D. Takach, Seattle University

Dr. Margarita Takach is an Associate Professor in the Electrical and Computer Engineering Department at Seattle University. She earned her PhD degree from the University of Washington. Her teaching interests include digital and analog circuits and systems and signal processing.

Shruti Singh, Seattle University

Dr. Shruti Singh is a Term Faculty in the Electrical and Computer Engineering Department at Seattle University. She earned her PhD degree from University of Denver specializing in renewable energy and smart grids. Her research focus is on renewable energy integration into smart grids, ensuring efficient energy management and grid stability, aiming for a sustainable impact. She is a member of IEEE, ASEE and SWE and has worked on several NSF and NREL funded projects.

Teaching Digital Logic in the Quarter-Based EE Curriculum

Introduction

Courses on digital logic are an essential part of all Electrical and Computer Engineering curricula. With the advent of FPGAs, the use of a hardware-description language (HDL) in digital-logic[1] courses has increased significantly. Augmenting a digital-logic course with an HDL has had positive results as well as presented challenges. The strengths and drawbacks of using discrete TTL devices as opposed to FPGAs in an introductory digital-design course have been discussed in the ECE-teaching literature. For example, according to Nickels [1], many institutions have incorporated—and in some cases, fully replaced labs using discrete TTL DIP parts with—programmable devices. In comparing TTL and FPGA design implementations in terms of wiring, debugging, and pedagogy, Nickels concludes that in an introductory lab, serious pedagogical drawbacks in using programmable devices may impede their use.

Almagambetov and Pavlina [2] compare three methods of teaching laboratories in a digital-logic service course for first-year students. The methods are wiring cookbook-style labs using off-the-shelf components, VHDL labs with virtual-wiring techniques, and hybrid labs combining the two (Ibid.). The hybrid approach was seen to produce better educational outcomes according to an extensive evaluation. Other educators have chosen a hybrid style as well. For instance, Areibi [3] has students start with breadboards, but after introducing FPGAs, observes that a student preference for FPGAs is established after a few labs. Nonetheless, Areibi has found that the use of VHDL in such a course causes confusion and has identified that the primary challenge was being able to teach enough VHDL in twelve weeks. Tabrizi [4], likewise, discusses *Digital Systems*, a second-year course with nine labs, seven of which are VHDL based. The course teaches a subset of introductory VHDL as well as the fundamental concepts of digital logic. Tabrizi presents the labs and the lecture slides of the VHDL subtopics covered, carefully ordered, and the lecture time spent on each subtopic. (To avoid confusing beginning students, variables are not covered.) Still other educators have also opted to cover a subset of introductory-level VHDL due to the challenges of introducing both logic concepts and HDL practices successfully and sufficiently in one course. Abidin *et al.* [5], for instance, use VHDL successfully to teach combinational logic by relegating sequential logic to a later course.

In another innovative take on teaching digital logic with HDLs, McCarthy *et al.* [6] had a former student who had taken the course previously create the lab exercises to improve student learning of Verilog. Zhao and Huang [7] improved student learning of VHDL by using inductive instruction. Some institutions teach HDLs in a separate course which has a course on digital logic as a prerequisite [8–9]. Our institution uses this approach; i.e., VHDL is taught in a

[1] We use "Digital Logic" (a proper noun) to refer to specific courses with that name at any institution. We use "digital logic" (the common noun) and "digital-logic course" (with the compound modifier) to refer to the discipline and to classes on this material in general, respectively. Our digital-logic course is titled *Digital Operations*.

separate course with the digital-logic course, called *Digital Operations*, as a prerequisite. The present paper describes our approach and its benefits.

Overview

Our curriculum requires three digital courses of all our ECE students: *Digital Operations*, *Programmable Devices*, and *Microprocessor Design*. The *Digital Operations* course is a prerequisite to *Programmable Devices*, which in turn is a prerequisite to *Microprocessor Design*. We are in a quarter system. *Digital Operations* is a four-credit lecture course taken in the first (fall) quarter of the first year of studies. *Programmable Devices* is a two-credit lecture–lab course taken in the second (winter) quarter of the second year. *Microprocessor Design* is a four-credit lecture–lab course taken in the third (spring) quarter of the second year. This paper addresses the advantages of this approach by providing details on the structure and management of our first two courses.

The *Digital Operations* course is taken by first-year students in the fall quarter. It has no prerequisites beyond admission into the university. The course is also taken by students who transfer as juniors from community colleges. The course covers Boolean algebra, combinational logic, and sequential logic. It does not address HDLs. Students use *Logisim*, a free digital-circuit simulator. Each student receives their own kit (to be returned at the end of the quarter). The kit contains everything needed to perform six take-home labs. Students must demonstrate each lab to the instructor or the grader. The *Digital Operations* course is followed by *Programmable Devices*. Students take it in the winter quarter of the second year or in their junior year if they are transfer students. By this time, students have taken a Python course and therefore have programming experience. The *Programmable Devices* course meets once a week for a 50-minute lecture followed by a three-hour lab. There are normally eight labs and one final project.

Digital Operations Take-Home Labs

The purpose of the take-home labs is a) to help students view digital components as real and to have them see that they work as presented in class, b) to have students learn about breadboards, and basic components such as diodes, resistors and integrated circuits, c) to teach students basic debugging skills, and d) provide them an introductory experience to programmable devices.

During the first week of the quarter, students receive the aforementioned kit, which is a convenient plastic container that holds discrete components, a breadboard, and a 5-V source operated from a 9-V battery. The discrete components include several ICs from the 7400 series as well as resistors, capacitors, LEDs, a seven-segment display, a resistor IC, and a clock display. After *take-home lab 4* is finished, each student receives a Cypress *CY8CKIT-044 PSoC® 4 M-Series Pioneer Kit*. Accompanying this is the *PSoC Creator*, an easy-to-use software that allows students to implement a digital system by drawing a circuit schematic using components from the Cypress component catalog. Students implement state machines on the kit using look-up tables. The advantages of using *PSoC Creator* is that students do not need to know a programming language to implement a complex digital circuit—they get exposed to programmable devices and implement a state machine in a short time, typically within an hour.

Each kit contains a 9-V battery and a PCB that provides 5 V to the breadboard, as shown in Figure 1 and Figure 2, respectively. To incorporate a short-circuit indicator, an LED with a resistor is connected to the end of the breadboard. The LED will not light when a short circuit occurs.

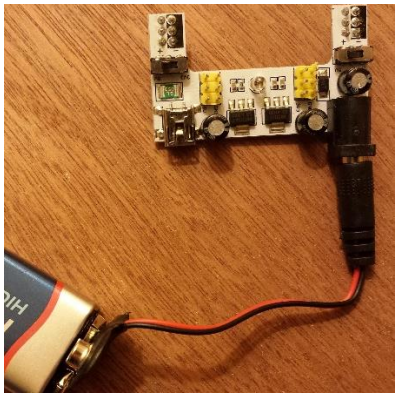

Figure 1: PCB powered with a 9V battery to generate 5 V

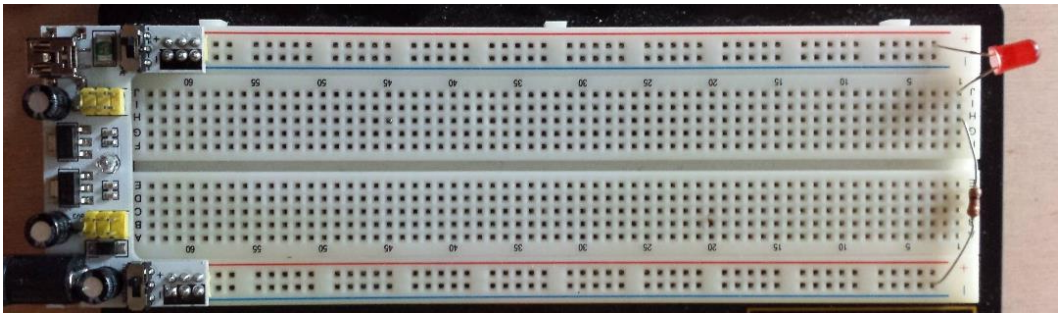

Figure 2: Short circuit indicator connected on the right side of the breadboard.

Take-Home Lab 1. This lab introduces the use of a breadboard, resistors, LEDs, and the 7400 IC. These components are new to most of the first-year students. They learn to test a NAND gate and to connect and test the four-NAND circuit shown in **Error! Reference source not found.**. The logic level at any node is tested using a logic probe consisting of a 330-Ω resistor in series with an LED.

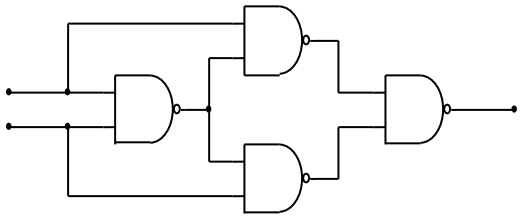

Figure 3: Circuit for lab 1.

Take-Home Labs 2 and 3. Students build and test a binary counter. The clock input of the counter is driven by a bouncy switch, a debounced switch, or a clock circuit. The output of the counter is displayed using four logic probes as well as a seven-segment display. The block diagram is shown in Figure 4. This is the largest circuit students build and it takes up all the space on the breadboard. Students get experience in debugging and in organizing their layout such that the components fit onto the breadboard.

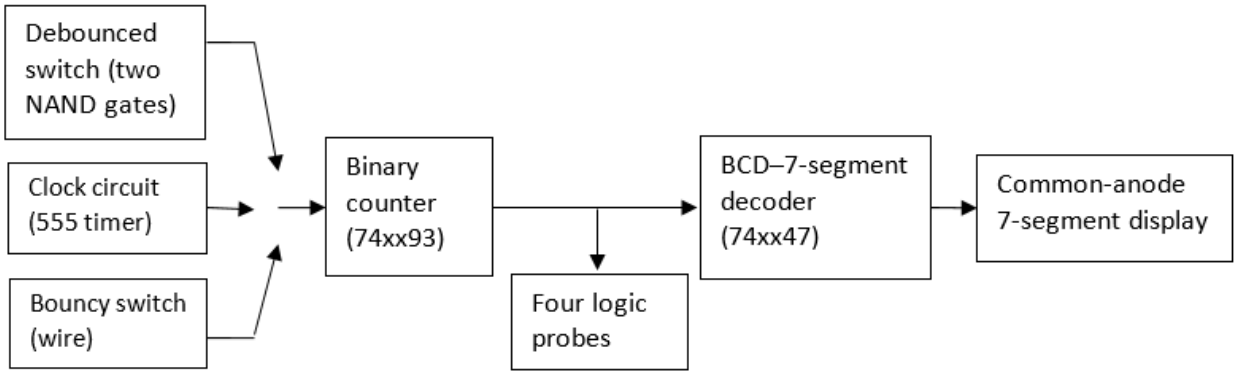

Figure 4: Block diagram for labs 2 and 3: The input to the binary counter can come from any of the three sources shown to the left of the diagram.

Take-Home Lab 4. Students are asked to design, build, and test a simple state machine. The circuit has an input X and two outputs. When $X = 0$, the circuit goes through the sequence 01, 10. When $X = 1$, the circuit goes through the sequence 00, 01, 11. This circuit uses the clock circuit from *lab 3* and ICs for the NAND and the D flip-flops.

Take-home lab 5. Students are introduced to *PSoC Creator* and the Cypress kit to implement the state machine from *lab 4*. Figure 4 shows the schematic used. Two external logic probes, implemented on a breadboard, are connected to the output pins. The frequency divider creates a low-frequency clock signal from the 1-kHz signal generated by the board. The lower frequency allows visual verification of the operation via witnessing LEDs turning on and off.

Students can typically finish this lab in less than one hour. We see further evidence in this to support the position that introducing programmable devices at this level of studies does not impose a significant time burden or, more importantly, conceptual challenge, to students. The instructions are carefully written so all students are successful and gain a positive attitude along with introductory skills, helping them look forward to working with programmable devices in the future.

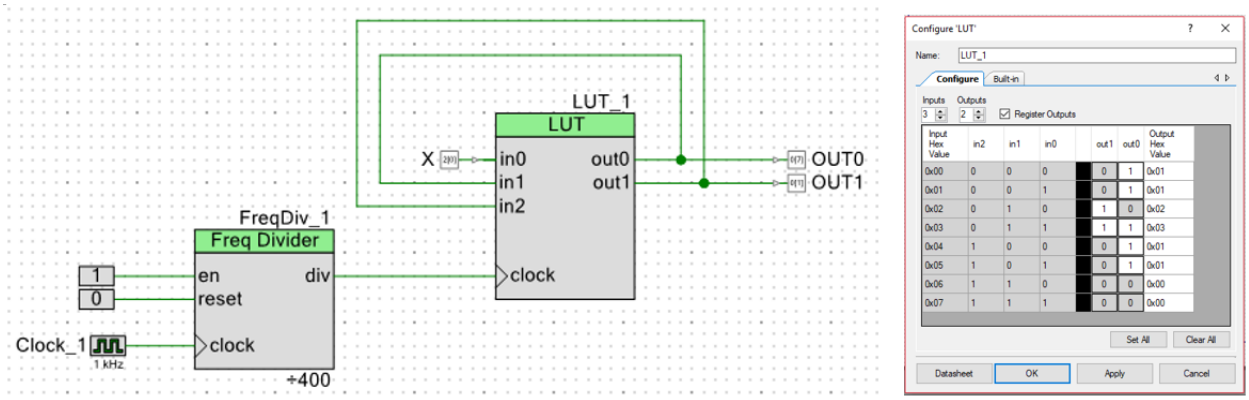

Figure 5:The schematic used to implement the state machine (left) and the contents of the LUT (right).

Take-Home Lab 6. This lab features two parts. In the first part, students learn to drive a clock display to show the output of a four-bit counter. The same decimal number is displayed simultaneously on the four seven-segment digits. Figure 6 shows the circuit schematic. The look-up table implements the seven-segment decoder. The output pins, D1 through D4, get connected to the common cathodes of each of the four digits of the clock display.

The two-bit counter is necessary for time-multiplexing the four digits, i.e., to turn on each digit for 1 ms (the period of the 1-kHz signal). The frequency divider again creates a low enough clock frequency so that the displayed digits can be visually distinguished.

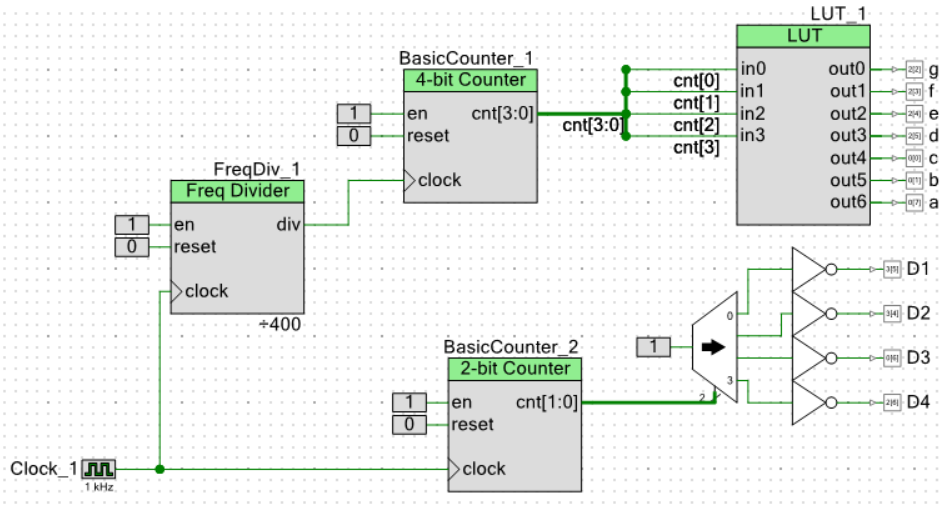

Figure 6: Schematic to display the output of a counter on a clock display.

Figure 7 shows the clock display, the resistor chip on the breadboard and the connection to the Cypress board. The IC, shown on the left, contains eight 470-Ω resistors. Many students finish the first part within an hour.

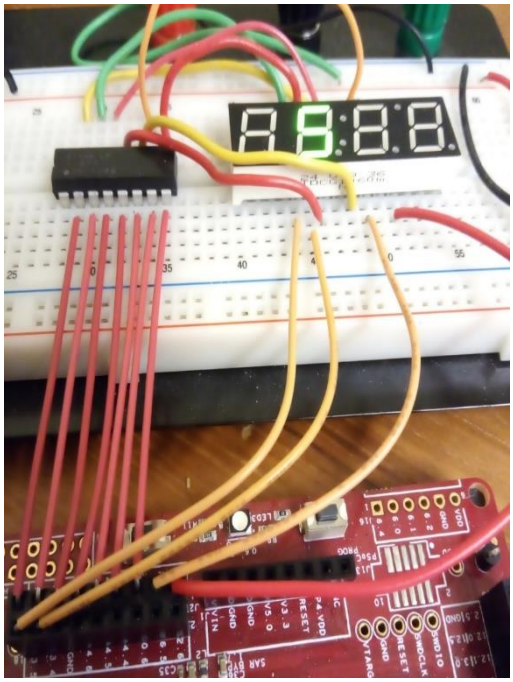

Figure 7: The clock-display circuit.

In the second part of the lab, students design the schematic necessary to display the output of a four-bit Johnson counter on the clock display. There are several possible design solutions. This part of the lab tends to take longer since students may go through several attempts in the design. Some students need help with the design.

We next describe the course on VHDL and programmable devices which follows the above logic course and re-exercises and reinforces logic concepts while introducing both programmable devices (with a focus on FPGAs and SOCs) and VHDL. This is a two-credit course, one credit for lecture–discussion and one credit for lab. We review the lab activities next.

Programmable Devices **In-Class Labs**

In this course, following an overview of programmable devices in terms of architecture, history, and design trade-offs, students are introduced to programming in VHDL while reinforcing concepts of digital circuits from *Digital Operations*. Each weekly 50-minute lecture time must be used very efficiently since there is key material on VHDL to be introduced as well as some digital circuits to review.

Because of the need to display numerous code examples, including disallowed, legal but non-ideal, and ideal VHDL-coding patterns, we use *PowerPoint* slides during the lecture portion of the class and make the content available to the students. The textbook used is the second edition

of "Circuit Design and Simulation with VHDL" by Pedroni [10][2]. Most homework is assigned from the textbook, along with some logic challenges designed by instructors.

Immediately following the lecture, students work in the lab for three hours. The time it takes to complete a lab range from two hours to three and a half, although there are times when a group gets stuck for much longer. The lab instructions are very detailed to ensure success while learning to use industry-grade synthesis and simulation tools. Students are expected to work on their own for most of the labs to maximize learning, but they are encouraged to work in pairs at certain points where the lab instructions prompt them to interrogate concepts or practical applications. At the end of each lab, each student is required to demonstrate their functioning results to the instructor or grader. Additionally, in most cases, each of several stages of the lab work must be demonstrated before students are checked off and may continue to the next stage.

The labs use *Altera*'s now *Intel*-owned *Quartus* for compilation and synthesis, the *Altera DE1-SoC* and the *DE2-115* boards, and *Siemens*' *Questa Sim* for simulation. To reinforce VHDL concepts covered in the labs, weekly homework is assigned to be done individually. Since the code needed to do the labs is provided (in its entirety in the early labs and partially in later labs), homework is students' opportunity to further practice on their own. An overview of the contents of each lab follows in order to show the progression of the material and the integration of VHDL with the reinforcement of logic circuits from the previous course.

Lab 1. Students are introduced to *Quartus*, and the *DE1-SoC* and *DE2-115* boards, and to the concept of concurrent code. A VHDL file describing a seven-segment decoder is provided, although it is a static image and students are asked to type it in to help familiarize them with VHDL syntax, keywords, and good style through the sensory feedback of reading and typing. In this lab, students learn how to assign input pins (slide switches) and output pins (LEDs in a selected seven-segment display) while becoming familiar with the requirements of our selected synthesis engine.

---

[2] We should note here that this edition of the textbook is from 2010 and that we have piloted using the third edition—dated 2020 and with a slightly different title—and pivoted back to the second edition. The former, titled "Circuit Design with VHDL" [11] and presented by the publisher as the next edition to [10] is a revamped book that starts with four chapters of digital-circuits review not found in the previous version and a different philosophy to its organization. We find [10] easy to teach with and learn from because the many sections are named after the VHDL keywords they address: '2.4 `ENTITY`', '2.5 `ARCHITECTURE`', '2.6 `GENERIC`', '5.3 The `WHEN` Statement', '5.5 The `GENERATE` Statement', '6.3 `PROCESS`', '6.8 `CASE` versus `SELECT`', '7.2 `SIGNAL`', '8.2 `PACKAGE`', '8.3 `COMPONENT`', '8.4 `GENERIC MAP`', and so on.

[11], on the other hand, deals with each overarching concept in its own chapter followed by a "practice" chapter of examples and exercises and in only a few cases ('The *if* Statement', 'The *case* Statement', 'The *wait* Statement', and 'The *loop* Statement') features the VHDL keywords in section titles. Although both books, of course, address the VHDL language and do so in a thorough and excellent manner, we have not identified a clear advantage in switching to the surprisingly more affordable third edition.

Lab 2. The purpose of this lab is to introduce *Questa Sim* (formerly *Model*Sim). Students use *Questa Sim* to simulate the priority encoder from their first homework assignment (due before this lab). Students learn about "force" commands and the *Tcl*-based "do" files in *Questa Sim*. They also implement a full adder by using a component and simulate it in *Questa Sim*. In the last exercise, students use the addition operation from the IEEE library to add two numbers and simulate this operation.

Lab 3. Students are introduced to basic sequential code in the pre-lab session. They first simulate a clock divider with the input-to-output frequency ratio of 6. The code is provided, but requires modification to be simulated using a `GENERIC` statement to create any arbitrarily chosen fraction of the input frequency. (`GENERIC` statements also help students associate general good coding practices with the realm of HDLs.) Afterwards, the clock-divider code is uploaded to the board, which takes a 50-MHz built-in clock signal and generates an output signal that turns on and off an LED. The output frequency is again selected to be low enough to see the LED blinking.

At this point, we find it imperative to emphasize that although instructions are provided with screenshots, these are not cookie-cutter labs. Students are empowered to understand and be able to explain circuit functionality. Two ways in which we build this into the labs are that (1) students are asked to get the synthesis result and verify that it makes sense, and that (2) in all but the first two labs, they are required to develop their own VHDL code (starting from some guide code that is provided).

Lab 4. This lab adds the 'package' feature in *Quartus*, a natural extension to components, accompanied by the ability to use block diagrams instead of explicit code. It builds on the one-digit counter students code on a homework assignment. The lab covers almost all the elements of VHDL seen in the course to date (synthesis, simulation, RTL view, components and port maps, signals vs. variables, exponent notation, sequential code, and concurrent code) as a review that precedes the change of direction in the next lab.

In this lab, students create a package with three components: the one-digit counter, the clock divider from *lab 3*, and the seven-segment decoder from *lab 1*. The main entity instantiates the three components using port maps, which they learn the previous week. Next, students implement the counter on the FPGA using the package along with the four other VHDL files. Lastly, students learn to create a symbol for each of the components and connect them in a block diagram in *Quartus*. The purpose of this lab is to introduce students to the principled use of multiple VHDL source files (enabled through the use of 'components' and the generation of a 'package') and the use of block diagrams, as shown in the figure below, for connecting components.

Lab 5. The purpose of this lab is to learn how to write VHDL code to implement a state machine. Students write the code to implement a simple holiday light generator using 10 LEDs with an input that controls the direction of the LED pattern. The simulation is done first followed by implementation on the FPGA. The lab material indicates how the typical diagrammatic view of state machines corresponds to a state-machine VHDL template from the textbook.

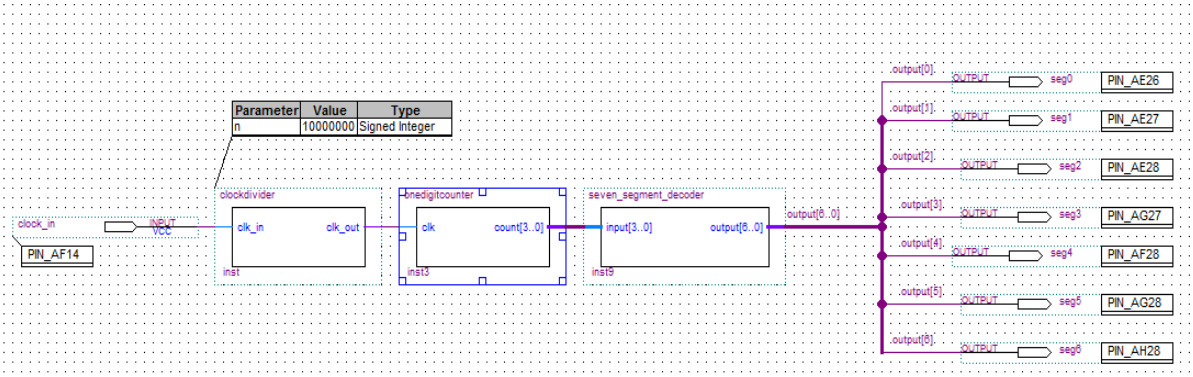

Figure 8: Block diagram for the one-digit counter.

Lab 6. Students are provided with the file necessary to control the display of a VGA monitor. They create a block diagram for it, named *vga_control*, as in Figure 9, and write the VHDL code to create various user-controllable images on the monitor.

Initially students are provided with the code to create horizontal stripes of different colors with the color of each stripe controlled by a switch. The code is then represented as a *make_image* block and connected to the *vga_control* block as shown in Figure 9. Students then modify the *make_image* program to display vertical stripes, and then to have a square able to move vertically, horizontally, and diagonally all across the monitor and at different speeds. Students also control the frame size for the range of motion of the square.

This use of VHDL to control graphics on a monitor serves three purposes. The least of those is that students get a sense of the range of capabilities of the language. Another is that students get to stretch their facility with VHDL in a particularly challenging task—challenging because VHDL is far from optimized to serve as a game-design language. Finally, and perhaps pedagogically most importantly, this graphical ability of the language allows us to boost student engagement in the course by making game design the most salient, or visible, outcome of the course. Gaming being a common popular pastime, this becomes a motivating factor for many students in recent generations to be driven to invest time and feel ownership in the course.

Lab 8. Students create a 1D×1D array—a bit map—that holds the image they want displayed on the monitor as the "ball" in a game akin to *Pong*™. They also create a "paddle" that moves vertically (or horizontally, depending on the game design), controlled by a pushbutton. When the moving image ("ball") comes across and hits the paddle, it bounces back. The main pedagogical benefit in this lab is that there are multiple processes running simultaneously.

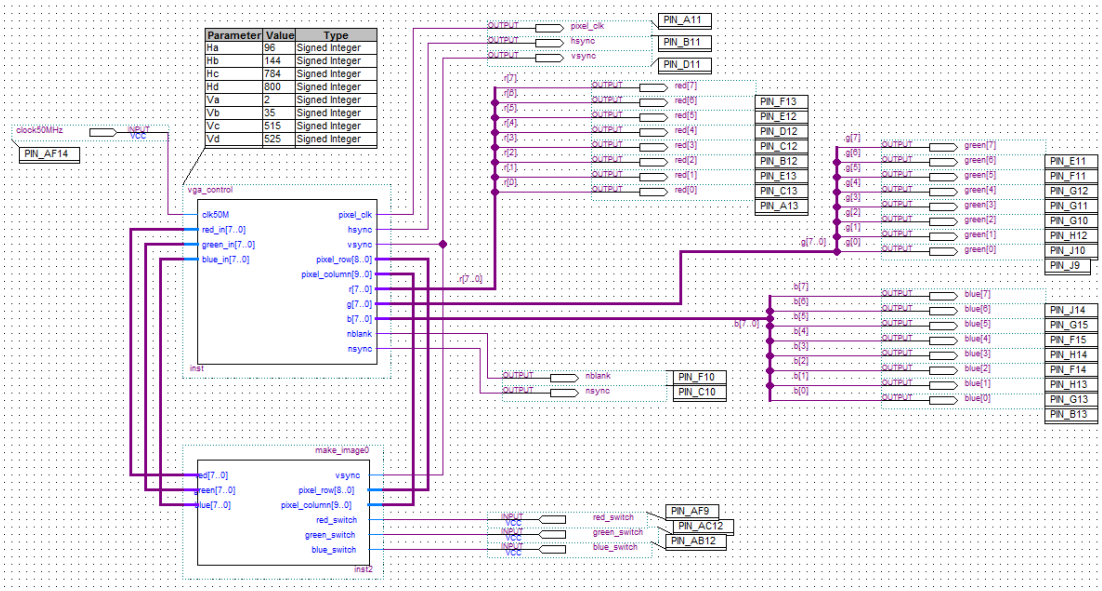

Figure 9: Block diagram for creating an image on a VGA monitor.

Project. During the last two weeks of the quarter, students work in groups of two on a self-selected project. This often involves the use of the monitor to implement a game. Creative extensions to the minimum requirement of a *Pong*™-like game are encouraged. Some examples of regular and more advanced games by students are shown in Figure 11 and **Error! Reference source not found.**.

Making a video game (or computer game) in VHDL is analogous to running on sand when preparing for a marathon: Although VHDL contains VGA and DVI controls, the language was not intended for game design. Hence, while we *require* only the bare minimum for the project to be considered complete, several students each year choose to go beyond the minimum. Below are images from several projects, some ordinary *Pong*™-style games (Figure 10), a couple of more advanced games (a simplified *Duck Hunt*™-style game with a *Raspberry Pi*-controlled joystick and a simplified, vaguely *Asteroids*-style game, respectively, as shown in Figure 11 and Figure 12) and a working multi-level recreation (from scratch) of a *Super Mario Bros.*™-type game that a student developed as an independent project immediately following the VHDL class *Programmable Devices*.

Figure 10: Three Pong™-style games, the basic project.

Figure 11: Starting screen for a Duck Hunt™-style game with a Raspberry Pi-controlled joystick.

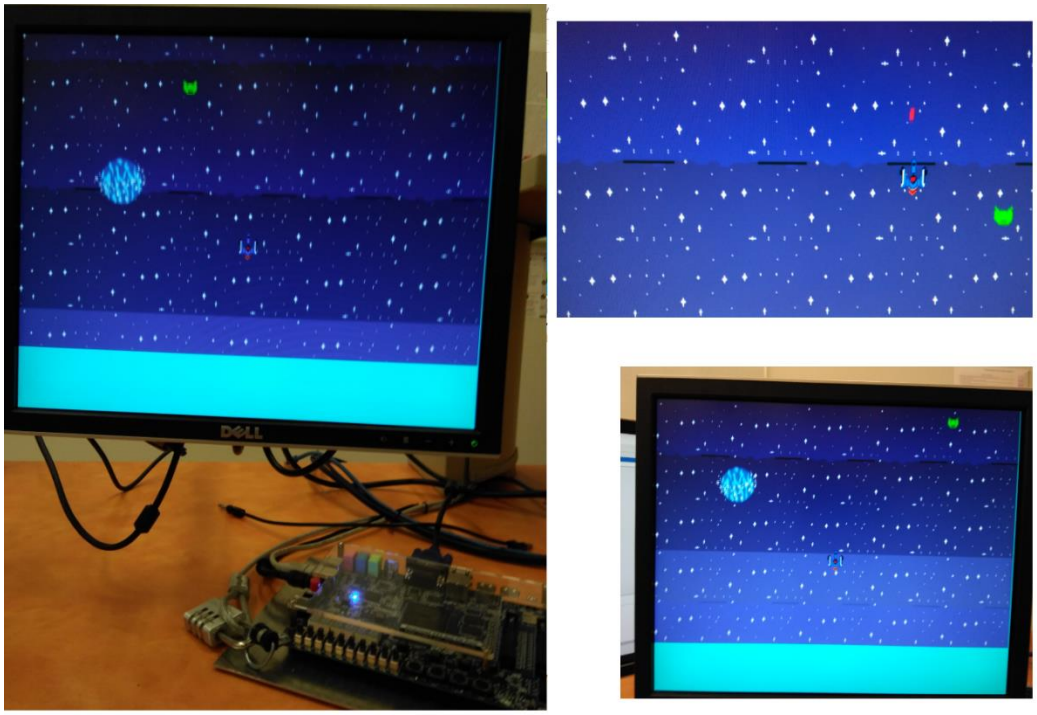

Figure 12: Three views of an asteroid-and-alien game featuring a spaceship that can fire as well as move up, down, and side to side with separate controls.

We have had success in teaching the course effectively. The weekly lectures are only 50 minutes, so they must be used very efficiently. There is emphasis on synthesis to induce students to think of the hardware that is produced and help them understand concurrent and sequential code.

Reflections on Teaching the *Digital Operations* Labs

We have taught the *Digital Operations* course with take-home labs for over 26 years. We have never included HDLs in this course. Hence, we cannot assess how including HDL affects students' learning. What follows are some reflections on the take-home labs.

Students earn points only if they can demonstrate a working circuit. On occasions there might be one or two students who fail to successfully demonstrate one out of the six labs in *Digital Operations*. This, we have consistently observed, happens if a student starts working on a lab at the last minute, not due to lack of skills, instruction, or comprehension.

All students learn how to use a breadboard and learn some debugging skills. Initially, many students rely on the instructor to debug their circuit, but after a few weeks, most students are capable of helping each other. A bonus is that the take-home labs often serve as a way for students to get to know each other by working together in residence halls and the departmental labs and form a community of learners.

In the first two labs, there tends to be a struggle among students to learn how to use a breadboard in connecting up a circuit. Consequently, we provide "paper breadboards" on which students

draw the components and all the connections that need to be made. Students then translate those graphics onto the real breadboard. This has helped considerably with student comprehension of circuit connectivity. For the rest of the labs, students demonstrate success with breadboarding, and their cognitive efforts are turned to learning debugging skills instead.

Every course taught is evaluated in terms of what students felt was especially helpful. Our quantitative assessment to date has not been fine-tuned to the matters addressed in this paper, but we are able to offer a selection of student comments indicating engagement, experience, and understanding:

The hands-on portions with the labs is a ton of fun and really helps understand the real-world applications of what we are learning.

The labs were the most enjoyable and interesting part of the class, as they allowed for a hands-on application of the lecture material.

The take home labs were by far my favorite aspect of this class. They provided an immersive experience where not only could I demonstrate my knowledge in the class content, but I was also given an opportunity to gain real-life skills in building and designing a circuit.

Reflections on the Labs

We have taught *Programmable Devices* for the last nine years. There are VHDL topics we do not have enough time to cover, such as creating functions and testbenches. Trade-offs are inevitable and this compromise provides the time needed to focus on synthesis from both concurrent and sequential statements.

Depending on enrolment, we teach one or two sections of *Digital Operations* during the fall quarters. A section typically has on the order of two dozen students. A great deal of time outside of class is put into helping students debug their circuits. Additional faculty time is used for circuit demonstrations even when the course grader can help with this task. Consequently, take-home labs are best implemented in small classes with the help of a grader or teaching assistant and practical implementation in larger classes could be facilitated by adding an in-class lab component and having students work in groups of three.

Programmable Devices is a prerequisite for our *Microprocessor Design* class and is taken immediately following the *Digital Operations* class. One of the main challenges featured in our *Microprocessor Design* class is to create a microprocessor in VHDL. To successfully complete this challenge requires that students build a strong foundation of VHDL syntax, tools, simulation, and synthesis.

Conclusion

Digital logic is an ideal subject to teach first-quarter ECE students because it easily lends itself to simulation and hardware labs, without any prerequisites. We use *Logisim* for simulation; the hardware portion is accomplished through take-home labs. These labs teach students about breadboards, ICs, passive components, and debugging skills. They also make the material more interesting—more real—to students. Keeping HDLs out of the *Digital Operations* course makes

the course more accessible to first-quarter pre-engineering majors as well as ECE majors. We have never taught a hardware-description language in our *Digital Operations* course; thus, we cannot assess student outcomes with and without using an HDL in that course.

A year later and after acquiring some programming background in other courses, our students take the *Programmable Devices* lecture–lab class. Some mature students have successfully taken *Programmable Devices* without any prior programming experience. We have not found any unusual difficulties in students' learning of VHDL. Having a separate course dedicated to VHDL seems to avoid some of the difficulties described in the recent ASEE, IEEE, and other publications discussed in our introduction. In addition, it provides a strong foundation for the follow-up *Microprocessor Design* class.

We are happy to share our labs for either course with interested educators.

References

[1] Nickels, K. M. (2000, June), *Pros And Cons Of Replacing Discrete Logic With Programmable Logic In Introductory Digital Logic Courses.* Paper presented at 2000 Annual Conference, St. Louis, Missouri. https://peer.asee.org/8648

[2] Almagambetov, A., & Pavlina, J. M. (2017, August), *Cross-sectional study of engineering student performance across different types of first-year digital logic design laboratories* Paper presented at 2017 FYEE Conference, Daytona Beach, Florida. https://peer.asee.org/29404

[3] Areibi, S. (2001). A first course in digital design using VHDL and programmable logic. In *Frontiers in Education Conference, 2001. 31st Annual* (Vol. 1, pp. TIC-19). IEEE.

[4] Tabrizi, N. (2017, June), *First Course in VHDL Modeling and FPGA Synthesis of Digital Systems* Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. https://peer.asee.org/28360

[5] Abidin, H. Z., Kassim, M., Othman, K. A., & Samad, M. (2010, December). Incorporating VHDL in teaching combinational logic circuit. In Engineering Education (ICEED), 2010 2nd International Congress on (pp. 225-228). IEEE.

[6] McCarthy, D., & Wright, C., & Barrett, S., & Hamann, J. (2010, June), *Student Created Laboratory Exercises For A Digital Systems Design Course Using Hdl And Plds* Paper presented at 2010 Annual Conference & Exposition, Louisville, Kentucky. https://peer.asee.org/16532

[7] Zhao, Y., & Huang, S. (2017, June), *Improving Student Understanding of Digital Systems Design with VHDL via Inductive Instruction* Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. https://peer.asee.org/28496

[8] Newman, K. E., Hamblen, J. O., & Hall, T. S. (2002). An introductory digital design course using a low-cost autonomous robot. *IEEE Transactions on Education*, *45*(3), 289-296.

[9] Hanna, D., & Haskell, R. E. (2007, March). Learning digital systems design in vhdl by example in a junior course. In *Proceedings of the ASEE North Central Section Conference, Charleston, West Virginia*.

[10] Pedroni, Volnei A. (2010). *Circuit Design and Simulation with VHDL*. Second edition. MIT press.

[11] Pedroni, Volnei A. (2020). *Circuit Design with VHDL*. Third edition. MIT press.