Paper ID #17899

Improving Student Confidence and Retention using an Introductory Computer Engineering Course

Dr. Daniel W. Chang, Rose-Hulman Institute of Technology

Dr. Daniel W. Chang is an Assistant Professor in the department of Electrical and Computer Engineering (ECE) at the Rose-Hulman Institute of Technology. He is the faculty advisor for the student chapters of the Institute of Electrical and Electronics Engineering (IEEE) and the ECE honor society Eta Kappa Nu (HKN). His interests include computer architecture, digital systems, memory systems, and engineering education.

Improving Student Confidence and Retention using an Introductory Computer Engineering Course

Abstract

An introductory computer engineering course where students learn about combinational and sequential circuits is fundamental to any Electrical and Computer Engineering (ECE) curriculum. Many of these courses are taught using a hardware description language (HDL) such as Verilog or VHDL. However, younger students traditionally struggle with HDLs due to their abstract nature. The students are used to designing with traditional logic gates and structures, but are often confused by the software-like interface that an HDL provides. This creates a disconnection between the student's experience in the classroom where the students learn with one method (visually with gates and structures) and in labs or projects where they are asked to implement designs using text descriptors. Often times a student's frustration with HDLs leads to them being disinterested in digital systems or even computer engineering as a major. This paper will describe the transition of an introductory Computer Engineering course from primarily using Verilog for its assignments to instead using a combination of schematic capture (which is very similar to what they see in class) and Verilog. With this course's redesign, the author saw the student's self-reported confidence in their design skills improve by 44% (from 41% to 85%) and their interest in taking additional computer engineering courses improve by 10% (from 66% to 76%).

Introduction

The goal of this paper is to describe how an introductory computer engineering course was changed from primarily using the Verilog Hardware Description Language (HDL) for its assignments to instead using a combination of schematic capture and Verilog. A previous iteration of this course had been taught for years using Verilog as its primary means for design assignments. However, the students would often be confused by the software-like interface that a HDL provides since it looked nothing like the logic gates and structures that they saw in lectures, homework assignments, and exams. This created a disconnection between what the students learned in lecture and what they did on their lab assignments. The students would routinely complain about Verilog in their evaluations and their frustration with HDLs would often lead them to becoming disinterested in taking additional computer engineering courses or going so far as to leave the major entirely.

The new iteration of the course covers many of the same topics as the previous iteration of the course including binary arithmetic, logic gates, combinational structures, (multiplexers, decoders, etc.), storage elements (flip-flops and latches), and finite state machines. However, the students first few lab assignments are done in schematic capture, which is a visual design tool that uses the same gates and components the students see in class. After mastering a section of the course and completing the associated lab assignments in schematic capture, students implement one of the previous lab assignments in Verilog. This allows the students to see how the software-like statements in Verilog actually translates into the same hardware with the gates and structures they are used to and that Verilog is simply another way of implementing things. Since implementing the changes to the courses, the author saw a dramatic improvement in the student's self-reported

confidence in their design skills and an increase in student interest in computer engineering courses and the major as a whole.

The remainder of the paper is organized as follows: The following section will summarize related work. The third section will detail the methodology in implementing the new course. The fourth section will present self-reported student survey results and the last section will conclude the paper and present future work.

Literature Review

A number of studies and papers have been published on improving undergraduate computer engineering student's proficiency in Hardware Description Languages (HDLs) like Verilog and VHDL. Nestor *et. al.* introduced HDLs and Field-Programmable Gate Arrays (FPGAs) throughout the ECE curriculum at Lafayette College[1]. He began introducing HDL design in senior-level electives, but then propagated the material down into both a junior-level computer architecture course and two sophomore-level digital design courses. The results were generally positive with the students appreciating the convenience of HDLs over traditional breadboarding and the satisfaction of being able to implement a large, complex digital system. However, he also observed that students made assumptions that HDL code could be written like software leading to synthesis and debugging problems. To address these problems, Nestor created a HDL coding guideline intended to help his students avoid common pitfalls and instill good HDL practices that lead to reliable designs[2]. He observed a higher level of success in the student's projects and believes that the HDL guidelines were a factor in this. Nestor's observations on his student's difficulty with HDLs such as Verilog are in line with my own observations and are some of my motivation behind the creation of the introductory computer engineering course outlined in this paper.

Holland *et. al.* introduced FPGAs into a computer architecture course at the University of Washington to enhance the student's experience by allowing the students to implement actual hardware instead of relying on software simulations[3]. The processor design was broken into separate design projects. The students first designed a register file, tested if it could read and write simple values, and finally implemented it on the FPGA. The second design project was designing an ALU, which after implementation, the students could integrate it with their register file and observe the ALU performing the correct operations and storing the results in their register file. Their approach was similar to mine, but my students were given a finished digital system and week-by-week removed one of my designed components, implemented it themselves, and tested to see what their component did in the larger system rather than doing the entire digital system design all at once.

DeGroat designed a Theory and Design of Digital Computers course at The Ohio State University, which included a strong emphasis on HDLs[4]. Unlike prior approaches to teaching Verilog, she taught the students VHDL through various levels of abstraction. The students started learning VHDL at the gate level where the HDL descriptors were one-to-one with the hardware being modeled. In other words, each HDL statement correlated with a specific gate in the design. Throughout the course, the students continued to move up levels of abstraction until they were able to write VHDL that implemented entire algorithms. Although DeGroat and I agree that students should not start with complex HDL constructs, my work differs in that I start the students

with schematic capture to do gate-level and subcomponent design and only move to a HDL after they have mastered schematic capture and solidified their understanding of the course material.

Methodology

Introduction to Digital Systems (ECE 233) is a four credit course that meets four times a week with three hour-long lectures and a three hour lab session. The course is offered twice a year and is usually taken during a student's sophomore year. The course has been offered for two years and over 200 ECE students have now taken the course. Like other introductory computer engineering courses, the course topics include:

- Number systems (binary, 2's complement, octal, and hexadecimal)
- Binary arithmetic
- Forming basic digital circuits using logic gates
- Boolean Algebra
- Karnaugh maps
- Combinational structures like Multiplexers, Full Adders, Comparators, Decoders, and Encoders
- Designing Combinational Circuits such as an Arithmetic Logic Unit (ALU)
- Storage using Latches and Flip-Flops
- Finite State Machines (FSM)
- Designing a Sequential Circuit
- Timing and Propagation Delay
- Register Design
- Designing a Control and Datapath
- Operation of a basic Reduced Instruction Set Computing-like (RISC) Processor

The previous iteration of this course included many of these topics, but the current iteration of the course has added some basic computer architecture in the form of RISC since the department wanted all of our students to be exposed at least the basics of how a computer processor worked not just the computer engineers. The topic is also a good example that combines all of the topics in the course such as combinational logic (Datapath) and sequential circuits (Control) into one big, integrated design (a processor). However, the biggest change in the new course is the addition of a weekly three hour lab session and the manner in which those labs and Verilog are presented.

Like many weekly lab sessions taught at other schools, the students first learn the material used in lab at least one week before the actual lab session. They then use time outside of class and in lab to implement a digital design. For ECE 233, I used the concept of Backward Design to design the lab assignments. Backward Design is a method for designing educational curriculum where the goals are set prior to choosing the instruction methods, assessment, or assignments[5].

The students in the previous course were having difficulty with large and complex Verilog design projects and with this in mind, I used Backward Design to rebuild the lab materials. Specifically, I wanted the lab materials to reinforce the topics and for the students to have a less frustrating experience. Therefore, instead of having multiple unrelated projects in Verilog, I

created a quarter-long project where the students would see the final functional design during their first lab period and week-to-week would replace the components I designed with their own. By the end, the system would be made almost entirely of their components, not mine. This would enable the students to more clearly understand what their smaller components did in the larger digital system while also having a less frustrating experience since each lab had a smaller scope.

Currently the students implement the game Pong, which incorporates seven-segment displays for scoring, push buttons for the player's paddle, shift registers displayed on LEDs to represent the ball, and a finite state machine to implement the game logic (i.e. Control). Each of the listed concepts is tied both to lecture and an individual lab, which again, the students use to gradually build the entire system component-by-component. The students implement their designs on the Altera DE2 Field-Programmable Gate Array (FPGA)[6]. The Altera FPGA loaded with the Pong game can be seen in Figure 1.

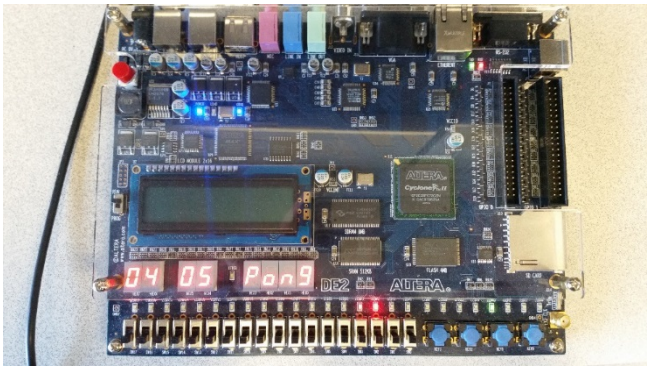

Figure 1: The Altera DE2 FPGA Board running the Student's Lab Project

Additionally, it well known that undergraduates struggle with hardware description languages such as Verilog and VHDL due to its software-like interface that looks nothing like the gates and structures students see in class[1]. Therefore, I designed the first two lab assignments to be done with schematic capture, which is a digital design method where students simply drag and drop logic gates and components and connect them with wires. This is much more consistent with what they see in lecture and allows the students to more easily tie their lab experience with the topics seen in lecture. My students use the Altera Quartus software to implement their digital circuits, but there are many tools that allow schematic capture. The students in the previous iteration of the course used Verilog and the Xilinx tool chain. An example of a simple digital circuit using Quartus' schematic capture tool can be found in Figure 2.

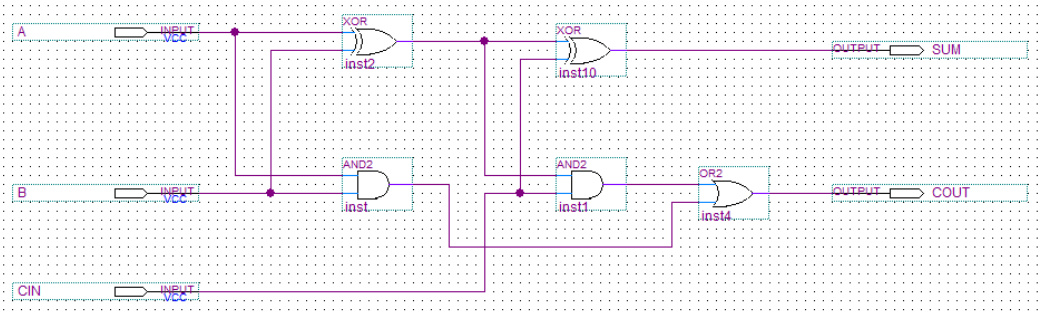

Figure 2: A 1-bit Full Adder using Schematic Capture

To help with the student's transition to Verilog, the first Verilog lab is re-implementing the second lab assignment, which uses Karnaugh Maps to display the player's score on the seven-segment displays. I chose this method since it allowed the students to more easily see the connections between what they did in a previous lab using schematic capture to the Verilog statements they were writing. Specifically, it gave them a one-to-one correlation between the gates in schematic capture and the corresponding software-like statement in Verilog. Additionally, I give the students a lecture on Verilog during the lab session before they start implementation. This lecture covers the basic syntax including gates, operators, modules, etc. and also how to make a test bench to simulate their designs. Figure 3 shows the segment B circuit of the seven-segment display that the students implement using schematic capture in Lab #2. Figure 4 shows the same segment B circuit that the students implement using Verilog in Lab #3. Viewing both together, one can see how the students translate the schematic capture circuit in Figure 3 into the Verilog logic operators and assign statement in Figure 4.

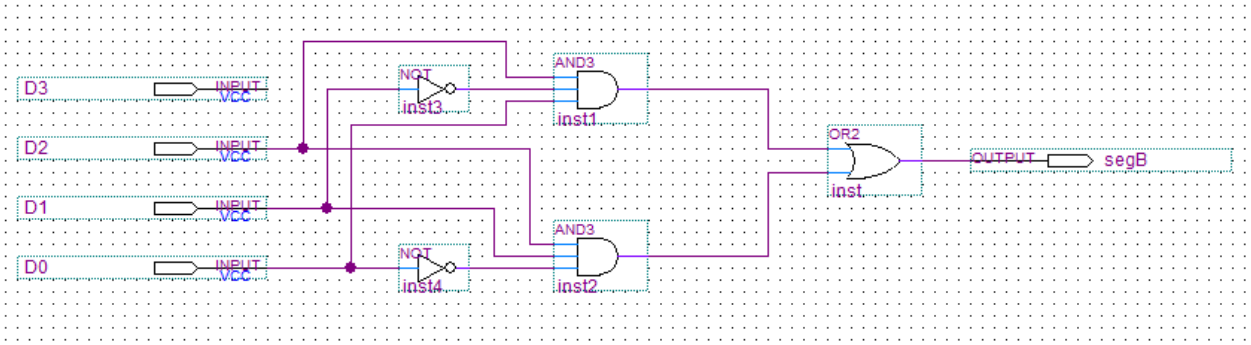

Figure 3: Segment B of a Seven-Segment Display using Schematic Capture

```
1  module segB
2  (
3      input D3, D2, D1, D0,
4      output segB
5  );
6
7  assign segB = (D2 & ~D1 & D0) | (D2 & D1 * ~D0);
8
9  endmodule
```

Figure 4: Segment B of a Seven-Segment Display using Verilog

The second Verilog lab is introduced much later in the quarter after the students have mastered all of the material regarding sequential circuits including finite state machine design. Like the first Verilog lab, I start the lab session with a lecture covering how to write a state machine in Verilog, clock generation, and exhaustive test methodologies for testing all transitions. Unlike before, the students do not implement the same finite state machine they did in a previous lab, but instead implement a simple sequence detector (i.e. detects the binary sequence 00100). This was chosen since the state machine implemented for their Pong game has a lot of states and outputs that might be confusing when introducing a new Verilog concept. A sequence detector is a much simpler state machine with very few inputs and outputs allowing them to focus on learning the Verilog syntax

for state machine design. Figure 5 shows a portion of the finite state machine that the students implement using Verilog.

```
1  module FSM
2  (
3      input X, clk, rst,
4      output reg detect,
5      output reg [3:0] CurrentState, NextState
6  );
7
8      // State Encoding
9      parameter State0 = 4'b0000, State1 = 4'b0001, State2 = 4'b0010,
10               State3 = 4'b0011, State4 = 4'b0100, State5 = 4'b0101;
11
12     // Current State Assignment
13     always @(posedge clk or negedge rst) begin
14         if (rst == 0)
15             CurrentState <= State0;
16         else
17             CurrentState <= NextState;
18         end
19
20     // Next State Logic
21     always @(CurrentState or X) begin
22         case (CurrentState)
23             State0:
24                 begin
25                 if (X == 0)
26                     NextState <= State1;
27                 else
28                     NextState <= State0;
29                 end
30             State1:
31                 begin
32                 if (X == 0)
33                     NextState <= State2;
```

Figure 5: Finite State Machine using Verilog

The previous iteration of this course had no dedicated lab sessions and instead had weekly design projects using Verilog. The course had no schematic capture and the students jumped right into Verilog early in the quarter. As described earlier, the students had difficulty with Verilog and would routinely complain about its difficulty and how their knowledge of Verilog was not indicative of their mastery of the course material and concepts. This frustration caused the students to have low self-confidence in their design skills, become disinterested in computer engineering, or even go so far as to change majors out of ECE.

Results

To evaluate the changes made to the new course, students from both the previous course (ECE 130) and the new course (ECE 233) were asked to take a survey that measured their self-reported confidence on a variety of digital design skills and also their interest in computer engineering. 24 students from ECE 130 responded and 83 students from ECE 233 responded. It should be noted that all sections of ECE 233 are taught the same regardless of instructor and multiple instructors have taught both courses. However, to isolate the results I only surveyed my students.

Figure 6 shows the student's self-reported confidence in implementing a single subcomponent of a larger system using Verilog or schematic capture. Based on the student survey data, 94% of the students in ECE 233 reported that they either strongly agreed or agreed that they could implement at least one subcomponent of a larger digital system. However, only 67% of the students in ECE 130 reported the same level of confidence in their digital design skills. Students of the new course were 27% more confident in their digital design skills over the students in the previous course. However, it should be noted students in both sections still had a reasonably high level of confidence in at least implementing one simple, digital circuit.

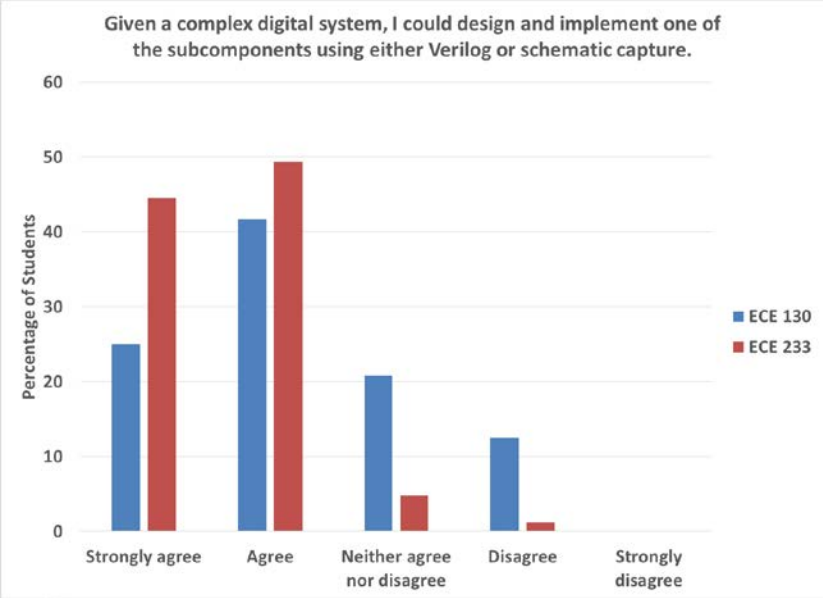

Figure 6: Student Survey Results on implementing a Subcomponent

Figure 7 presents the student's self-reported confidence in implementing an entire digital system using either Verilog or schematic capture. Understandably, the student's confidence in their design skills for both classes begins to degrade a bit here due to the complexity of designing an entire system versus only a single component. 74% of students in ECE 233 reported that they either strongly agreed or agreed that they could implement an entire digital system. Only 37% of students in ECE 130 reported the same level of confidence showing a 37% difference in student confidence between the current and previous iteration of the course. These results show a significant difference in confidence between the two courses.

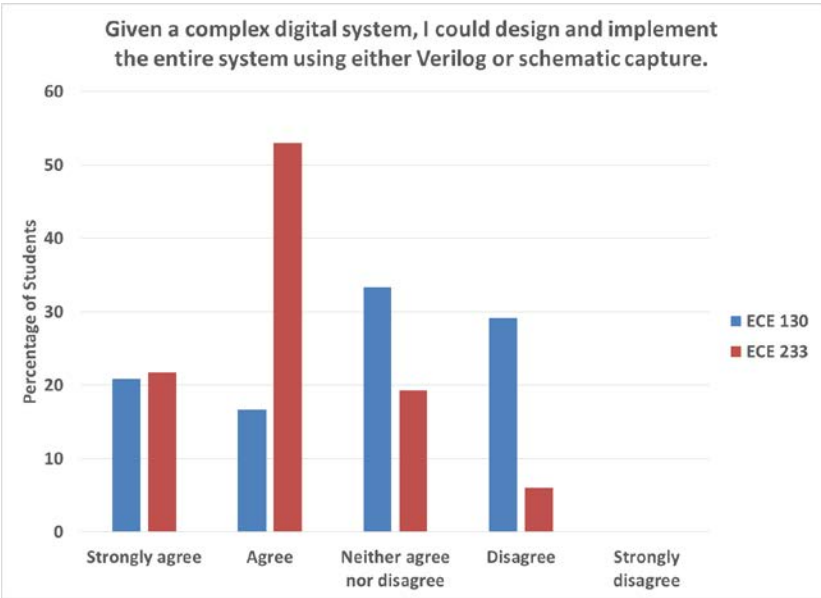

Figure 7: Student Survey Results on implementing an entire Digital System

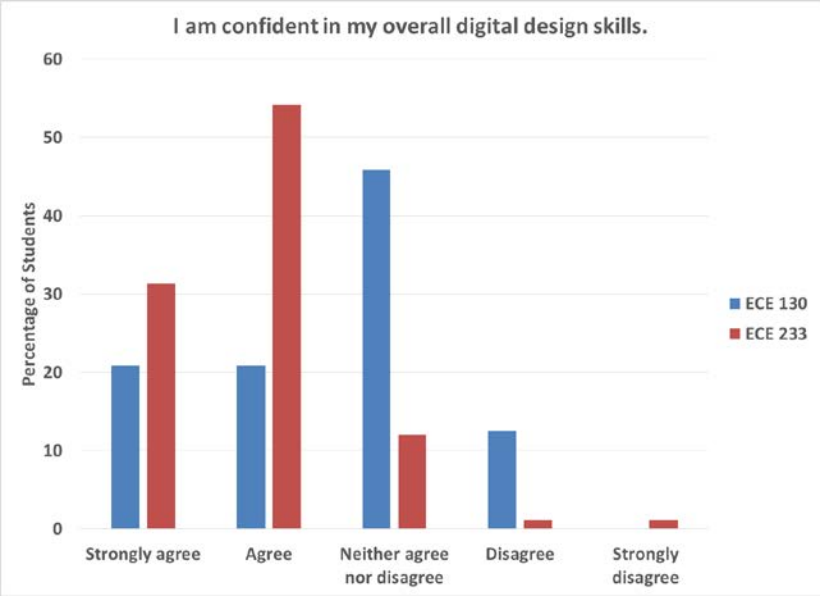

Figure 8: Student Survey Results on their Overall Design Skills

Figure 8 shows the student's self-reported confidence in their overall digital system design skills. Based on the student survey data, 85% of students in ECE 233 answered that they either strongly agreed or agreed that they were confident in their overall digital design skills. Only 41% of the students in ECE 130 had the same level of confidence in their overall digital design skills. Like the previous results, these results show a significant difference in confidence between students in the two courses. Students in ECE 233 demonstrated 44% more confidence in their overall design skills compared to students in ECE 130.

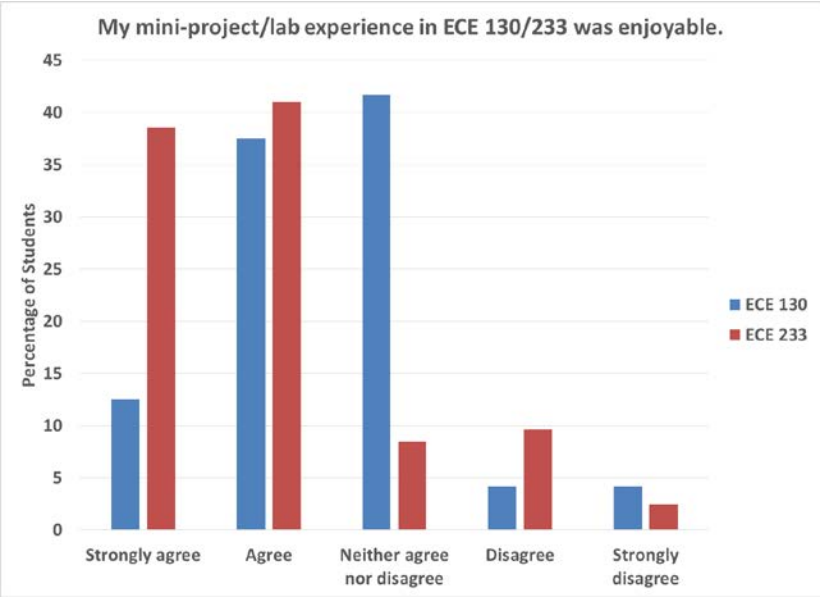

Figure 9: Student Survey Results on their Mini-project or Lab Experience

Figure 9 shows the survey results on the student's enjoyment of the ECE 130 Verilog mini-projects or ECE 233 labs using both schematic capture and Verilog. 79% of the students in ECE

233 enjoyed the labs using schematic capture and Verilog while only 50% of the students in ECE 130 enjoyed their Verilog mini-project experience. These results show that 29% more students enjoyed their lab experience in ECE 233 using schematic capture and Verilog compared to the students in ECE 130 using only Verilog. I believe the significant difference between the two courses in the last four survey questions correlates directly with the changes I've made to the new course.

Students in ECE 130 were given large mini-projects that they were expected to implement using Verilog while still learning the concepts and material. In other words, they were learning the material one way (gates, components, etc.), but then had to implement things another way. In a software-like language that was not very intuitive. However, students in ECE 233 first implemented their lab assignments using a tool that directly correlated with what they were seeing in lecture (gates, components, etc.), which helped to solidify their understanding of the material. Only after having mastered the material did they transition to Verilog. Furthermore, they were able to tie their previous schematic capture designs to the Verilog software-like constructs making it easier for them to tie together. I feel the significant improvement in the student's confidence and the enjoyment of the design experience can be directly tied to the changes I've made in ECE 233.

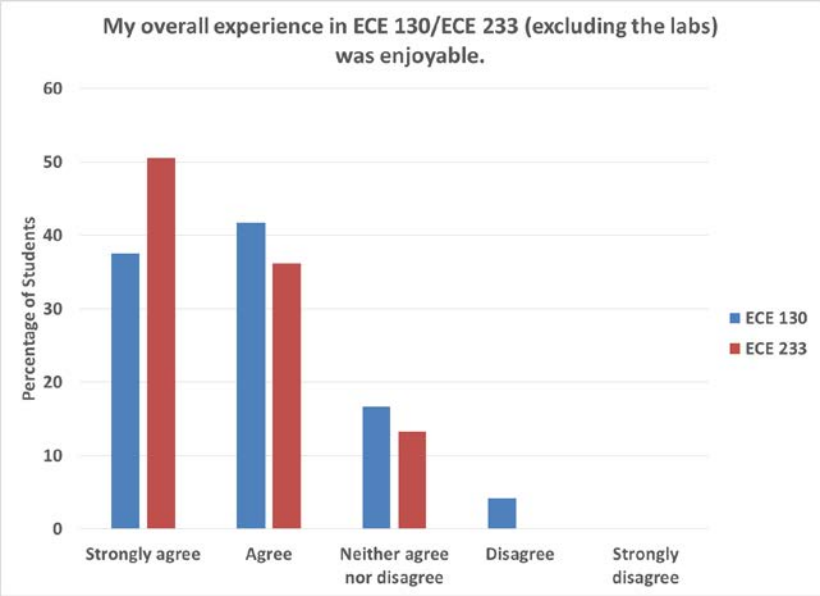

Figure 10: Student Survey Results on their Overall Experience in the Course

Figure 10 presents survey results on the student's enjoyment of the course outside of labs and Verilog mini-projects. The purpose of this survey question was to see if students could still enjoy introductory computer engineering material even if they disliked the design aspect. In other words, my hypothesis was that the material was still inherently interesting and the majority of students could still enjoy learning about it even if their experience was hampered by a negative Verilog or lab experience. 86% of students in ECE 233 answered that they either strongly agreed or agreed that they had an enjoyable experience with the course excluding labs while 79% of students in ECE 130 had a similar experience. These results correlate with my hypothesis that introductory computer engineering topics such as logic gates, sequential circuits, and computer organization

are still interesting to students and it is not the difficulty or the nature of the material that causes problems for students. It is the abstract nature of Verilog and how the material is reinforced.

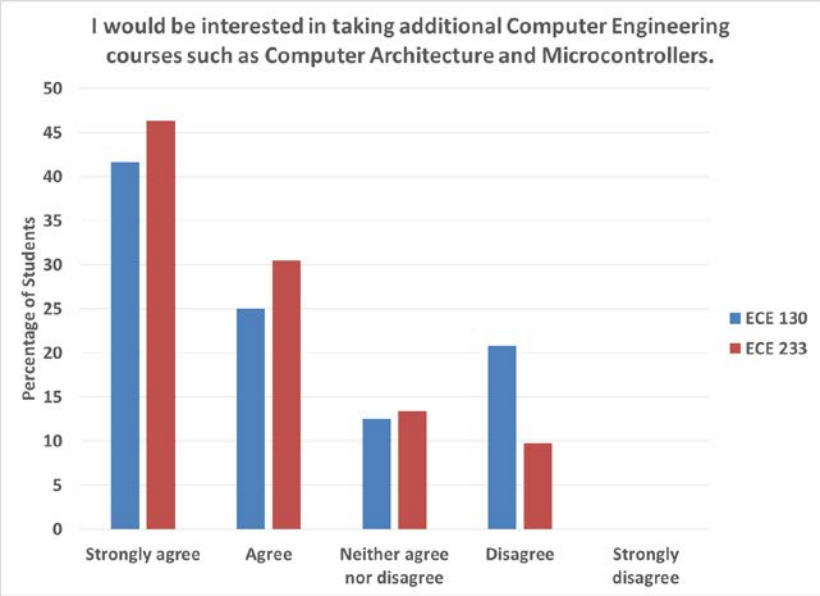

Figure 11: Student Survey Results on their Interest in taking Additional Courses

Figure 11 shows the student survey results on their interest in taking additional computer engineering courses. 76% of students in ECE 233 strongly agreed or agreed that they were still interested in taking additional computer engineering courses. Only 66% of students in ECE 130 expressed the same level of interest in taking additional computer engineering courses demonstrating a 10% difference between the two courses. The purpose of this survey question was to demonstrate that the student's experience with an introductory computer engineering course had an influence on their interest and willingness to learn more about computer engineering. The results do not show as significant of a difference as many of the previous survey questions, but still demonstrates that a number of students lose interest in computer engineering based on their experience in an introductory computer engineering course. In addition to the change from Verilog only mini-projects to schematic capture and Verilog labs, I believe adding the introductory computer architecture concepts near the end of the course also helped with student interest. A computer processor is a great example that ties all of the concepts the students learn throughout the course into something they know is useful and now have confidence in designing in the follow-on computer engineering courses.

These results show that it is imperative that students have a positive experience in an introductory computer engineering course to help with retention in the program. Our department has seen an increase in retention in the computer engineering degree program and a dramatic increase in percentage breakdown between the two ECE disciplines. In the past, our department has generally been 70 – 80% electrical engineering students and only 20 – 30% computer engineering students. Since introducing the new course our breakdown has been closer to 60% electrical engineering students and 40% computer engineering students with the latest freshmen class being 50% computer engineering. Admittedly the increase in the freshmen class cannot be

tied to the new course since they have not taken it yet, but the data shows dramatic improvements in our computer engineering recruitment and retention and I believe some of that can be tied to the changes made in my new course.

Conclusions

In this paper, a successfully implemented introductory computer engineering course was described where the students implemented digital systems in lab sessions using a combination of schematic capture and Verilog. The previous iteration of the course only used Verilog for out-of-class mini-projects and the students routinely complained about the difficulty of Verilog's software-like interface, which did not correlate with the gates and components they see in lecture. This led to the students having lower confidence in their digital design skills and a lack of interest in taking additional computer engineering courses. Additionally instead of separate and unrelated Verilog mini-projects, the new course has the students implementing a complex digital system over the entire quarter where each lab assignment is one of the smaller subcomponents. This allowed the students to more easily identify what each subcomponent that they designed was doing in a larger digital system.

With the changes in the new course, the author observed a 44% improvement in the student's self-reported confidence in their digital design skills and a 10% increase in the student's interest in taking additional computer engineering courses. These improvements have led to stronger retention in the computer engineering major and a more even distribution of students between the electrical and computer engineering disciplines. It is this author's hope that the new introductory computer engineering course will serve as a gateway into computer engineering and lead to stronger computer engineering graduates who have a better understanding of digital concepts and also higher confidence in their digital design skills and understanding of Verilog.

Future Work

Although the new course has been a great introduction to computer engineering and Verilog, I do not believe it covers enough Verilog for a student who graduates with a computer engineering degree. Graduates in computer engineering need to have a greater mastery of Verilog and be able to identify best practices, when to use certain ideas such as parallelism, blocking versus non-blocking statements, and most important, what their Verilog actually synthesizes into. In other words, what they write in Verilog directly influences how fast their designs are, how much area and power it will consume, and that there are techniques to improve all of these.

Currently the computer engineering students learn more Verilog in their Computer Architecture class. However, they do so without any formal lectures or assignments and simply out of necessity to make their implementation process easier. With that in mind, my colleagues and I are planning on designing an intermediate Verilog class focused on many of the concepts highlighted in the previous paragraph. Unlike the introductory computer engineering course described in this paper, the course will focus solely on Verilog. The lectures will cover Verilog syntax, best practices, synthesis, etc. and the students must implement all of the homework assignments and projects in Verilog to give them more and more practice. We plan to roll out this intermediate Verilog course in the near future and intend to study the course in a similar manner as the study described in this paper.

Bibliography

1. J.A. Nestor, J. Greco, I. Jouny, "HDL-based instruction across the ECE curricula," *2010 IEEE Frontiers in Education Conference*, pp. T3F-1, 2010.
2. J.A. Nestor, "HDL Coding Guidelines for Student Projects," *2011 IEEE International Conference on Microelectronic Systems Education*, pp. 86-89, 2011.
3. M. Holland, J. Harris, and S. Hauck, "Harnessing FPGAs for Computer Architecture Education," *2003 IEEE International Conference on Microelectronic Systems Education*, pp. 12-13, 2003.
4. J. DeGroat, "Teaching Hardware Description Languages," *2005 ASEE Annual Conference and Exposition*, 2005.
5. G.P. Wiggins and J. McTighe, "Understanding by Design," *Prentice Hall*, 2005.
6. Altera, "DE2 Development and Education Board, User Manual," 2006. [Online] Available: http://www.altera.com.