Integrating Digital Logic Design and Assembly Programming Using FPGAs in the Classroom

William M. Jones* and D. Brian Larkins
Department of Computer Science and Information Systems
Coastal Carolina University
P.O. Box 261954, Conway, SC 29528-6054

ABSTRACT

Rising Field Programmable Gate Array (FPGA) market volumes combined with increasing industrial popularity have driven prices down and improved capability to the point that FPGA hardware and development environments afford academia the unique opportunity to embrace this technology not only in specialized graduate-level courses, but also across many of the courses of a traditional undergraduate computer science curriculum.

We have begun adapting several of our undergraduate computer science courses and associated laboratories to make use of FPGAs as a common platform of instruction. In this paper, we illustrate how to make use of FPGAs in courses that cover digital logic design and assembly programming while discussing some of the pro's and con's of their use. We also provide a detailed discussion of a laboratory project that integrates both assembly programming as well as digital logic design in such a way that allows the student to perform a trade-off analysis between using software in the place of a purely hardware-based solution to a common interfacing problem.

We conclude with an analysis of preliminary data gathered via student surveys and find that the results support the use of FPGA-based platforms in the undergraduate classroom. By making use of FPGA-based systems, not only are students exposed to a technology that is becoming much more prevalent in industry, they also benefit from the dovetailing of concepts and shorter learning curves between courses that come from making use of a common target platform.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education

*Corresponding author.

©*ACM, 2011. This is the author's version of the work. It is posted here by permission of the ACM for your personal use. Not for redistribution. The definitive version was published in ACM SE 11, ACM 978-1-4503-0686-7/11/03.*

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are a type of integrated circuit that contains a lattice of configurable logic blocks and a hierarchical interconnection fabric that allow the user to "program" the underlying logic structure and functionality after manufacturing. An FPGA can generally be configured to implement any logic function, and its overall capability is primarily limited by the the number of available logic blocks and size of communication fabric.

Due to their re-programmability, FPGAs have become integral components in a variety of fields ranging from rapid system prototyping to high-performance distributed computing. Improvements in technology have given rise to FPGAs with massive amounts of reprogrammable gates numbering in the millions. This increased gate density has moved the achievable design space from simple combinational and sequential circuits to full-blown computing systems. Rising FPGA market volumes combined with increasing industrial popularity have driven prices down and improved capability to the point that FPGA hardware and development environments afford academia the unique opportunity to embrace this technology not only in specialized graduate-level courses, but also across many of the courses of a traditional undergraduate computer science curriculum [1, 2]. Not only have these forces made the physical hardware more accessible, the prevalence of quality development suites and IDEs are making using FPGA-based prototyping systems a robust and powerful teaching tool in the classroom [5].

We have adapted portions of our undergraduate computer science curriculum and associated laboratories to accommodate making use of FPGAs as a central platform of instruction. At the present time, we currently make use of FPGA-based platforms in our courses that deal with digital logic design, assembly programming, computer architecture, and compiler construction.

Our intention is that by making use of an FPGA-based system across multiple courses, not only are our students exposed to a technology that is becoming increasingly prevalent in the industry, they also benefit from the dovetailing of concepts and shorter learning curves between courses that come from making use of a common target platform. In addition to this general goal, of particular interest in this paper is the notion of providing an integrated instructional platform that enables students to devise multiple solutions to a problem and to subsequently analyze the tradeoffs among

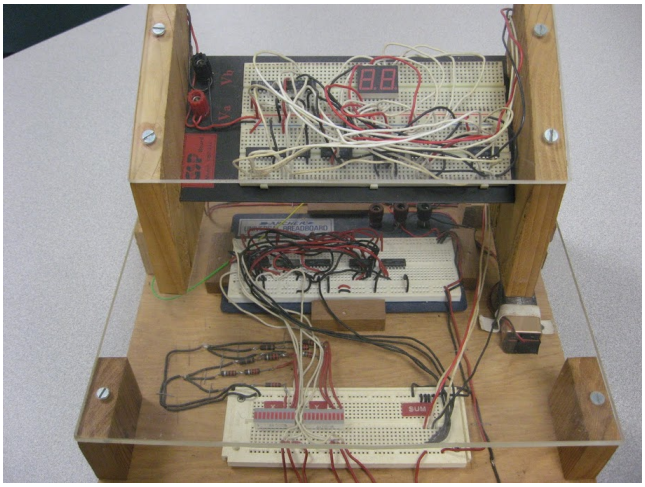

Figure 1: The use of breadboards and discrete 7400-series IC chips to implement an unsigned 3-bit ripple carry adder with binary and dual seven-segment displays.

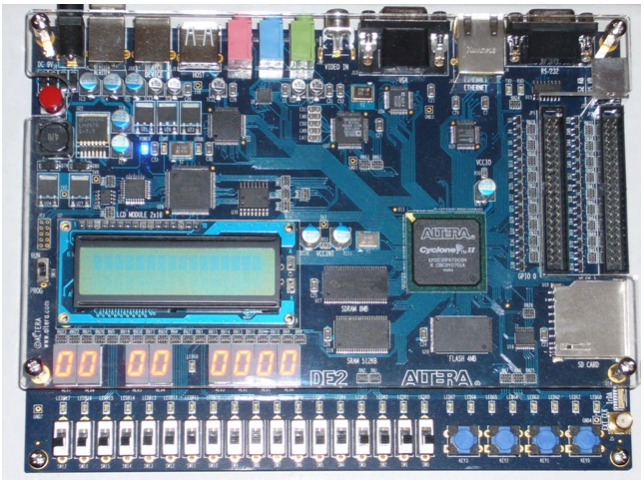

Figure 2: The Altera DE2 educational prototyping board. ***Note the wide range of available peripheral devices.***

them, a goal central to computer science education.

In this paper, we briefly focus on how to make use of FPGAs in courses that cover digital logic design and assembly programming while discussing some of the pro's and con's of their usage. We also provide a detailed discussion of a laboratory project that integrates both assembly programming as well as digital logic design in such a way that allows the student to perform a trade-off analysis between using *software* in the place of a purely *hardware*-based solution to a common interfacing problem. This experience allows the student to conceptually evaluate the solution to the problem along many dimensions and it also provides the opportunity to explore the challenges faced through *actual implementation*. We conclude with an analysis of preliminary data gathered via student surveys and find that the results support the use of FPGA-based platforms in the undergraduate classroom.

2. DIGITAL LOGIC DESIGN

A traditional approach typically employed and one that we previously used in the study of both combinational and sequential digital logic is the use of breadboarding and discrete logic gates (Figure 1) to implement designs of concepts that arise during lecture. This process generally begins with a qualitative description of the behavior of the the underlying circuit and then evolves to take on quantitative characteristics as the student develops truth tables for the circuit or device in question. Where possible, circuit minimization is attempted manually via the judicious use of Karnaugh maps or other techniques. Once a satisfactory set of Boolean expressions is obtained, the student will typically set about implementing the circuit using a breadboard, a low-voltage DC power supply, as well as a myriad of discrete logic gates including but not limited to ANDs, ORs, NOTs, and flip-flops. Where necessary, I/O devices such as single-pole double throw switches as well as LEDs (and potentially resistor banks as well) are used in order to drive and observe the behavior of the implemented circuit.

While this approach is certainly satisfactory in many cases, several problems generally arise during its use. In particular, students often suffer the consequences of hastily wired circuits by experiencing difficulties due to mis-wiring. Mistakes such as polarity reversal and short circuits can lead to the destruction of the integrated circuit chips used. Additionally, even with careful assembly during the breadboarding phase, a single misplaced wire can lead to hours of frustrating manual debugging.

Making matters worse is the prevalence of physically damaged parts such as breadboards with an open-circuit backplane channel, burned-out LEDs and IC chips that invariably become mixed in with those that function properly. The burden of maintenance and inventory control can become overwhelming for the department and infuriating for students.

2.1 Transition to FPGA Platform

In order to overcome some of these challenges while simultaneously leveraging a powerful common development platform, we have adopted the Altera DE2 FPGA-based prototyping board (Figure 2). This platform centers around an FPGA that is sufficiently large as to enable all manner of projects ranging from simple sequential and combinational digital logic designs all the way to full-blown RISC embedded soft processors (*more on this in Section 3.1*) [3, 6]. The vast array of peripheral devices common to these platforms allows them to be used in a variety of situations where interfacing with external devices is desired. Although we selected the Altera DE2 board (around $270), many such educational boards exist that feature FPGAs not only from Altera, but also from other manufacturers, such as Xilinx [8], ranging in capabilities and prices from around less than $100 to thousands of dollars.

After making this transition, we still focus lecture time covering the fundamental concepts of Boolean algebra, truth-tables, Karnaugh maps, minimization, and sequential state machine design; however, instead of implementing the re-

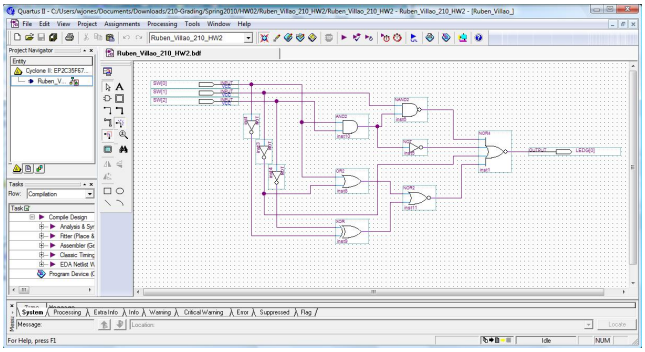

Figure 3: Altera's Quartus II IDE in schematic block diagram mode. Here the student has drawn a schematic that mirrors the design equations derived from pencil-and-paper techniques. The circuit is receiving input from switches and is outputting to an LED on the board. *Courtesy of CS student Ruben Villao.*

sulting expressions manually on a breadboard, we have the students make use of a freely available manufacturer-supplied IDE (Figure 3) to schematically enter their designs and to check for common wiring mistakes such as unused pins, short circuits and open circuits. The students then compile their designs targeting the FPGA fabric and load their designs onto the prototyping board.

One of the major benefits of using the development board is that designs are more easily debugged and changes that are made in the IDE can immediately be recompiled and reloaded into the FPGA. Aside from schematic input, students quickly find that making use of hardware description languages (HDLs) such as VHDL or Verilog can simplify the tediousness of design entry. Although HDL languages are extremely powerful in that behavioral descriptions can, in many cases, be automatically compiled into synthesized logic, we require students to make use of only Structural HDL here, so that the learning process of fundamental digital logic remains.

3. ASSEMBLY PROGRAMMING

The typical approach we had previously been employing for teaching assembly programming was through the use of software simulators, in particular, either for the PEP/8 or SPIM MIPS [7, 4]. One of the obvious benefits of using a simulator for this purpose is the fact that free simulators abound and can be run without any additional hardware directly on the students' own laptops. We have found that although students are comfortable with this process, they lack the realistic feedback of running their programs on an actual piece of hardware where interfacing with simple external devices such as displays and switches can provide a truly kinesthetic experience.

3.1 Transition to FPGA Platform

By leveraging the same development platform and environment in assembly programming as we have for the digital logic course, we are able to build on a technology with which our students are already familiar. In doing so, not only do we shorten their learning curve, we are able to provide the

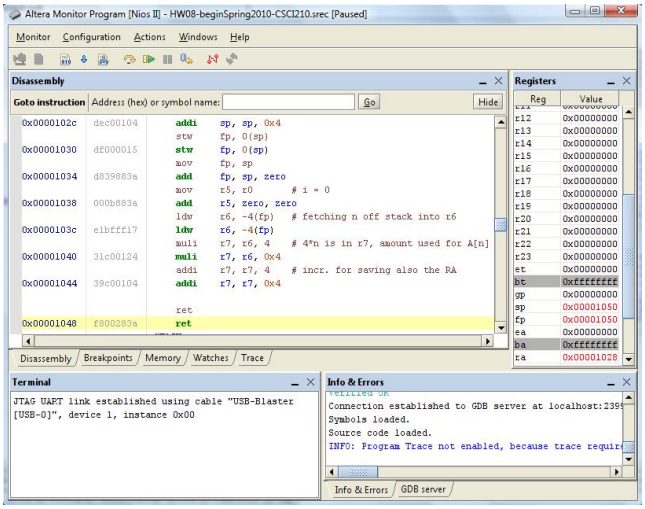

Figure 4: Altera Monitoring Program in use debugging student code.

opportunity to integrate concepts they have learned in their digital logic course with those they are learning in their assembly programming course. This is an interesting proposition because it joins together two disciplines that are often quite disjoint in introductory laboratory settings.

Altera's University Program provides an intellectual property core for the NIOS processor, a 32-bit RISC MIPS-like embedded processor that can be instantiated on the Altera DE2's FPGA [3]. By using a cross compiler and monitoring program, the students are able to load and execute their assembly programs remotely on the FPGA step by step from their connected host laptops. As with the typical simulators, the monitoring program allows the students to inspect the contents of registers and memory, while also being able to debug their programs using familiar GDB commands.

4. EXAMPLE DESIGN PROJECT

As part of departmental ABET accreditation, we have identified eleven overarching student learning outcomes (SLOs). One of these key SLOs is as follows:

"*The student will be able to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems through the* ***critical analysis of the tradeoffs*** *involved in design choices.*"

As such, we firmly believe that being able to devise multiple solutions to a problem and subsequently to analyze the tradeoffs among them is central to computer science education. One such design choice that inevitably surfaces is whether a software- or hardware-based solution to a given problem is "better." In order to allow us to explore this question at an early point in the curriculum, we have devised a project that allows our students to design two competing solutions to the problem of driving two seven segment displays (SSDs) as the primary output device of an embedded computing system.

The students are tasked with the responsibility of taking the

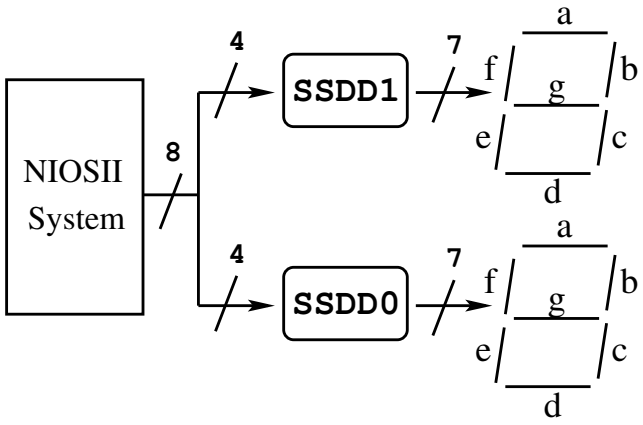

Figure 5: Using minimized combinational logic to implement the binary-to-HEX seven-segment display drivers (SSDDs). Here an 8-bit parallel I/O port is instantiated on the NIOS system, where the upper and lower nibbles map to their respective drivers. In this case, the contents of the lower byte of register R can be written directly to the I/O port using an assembly primitive corresponding to "store byte I/O."

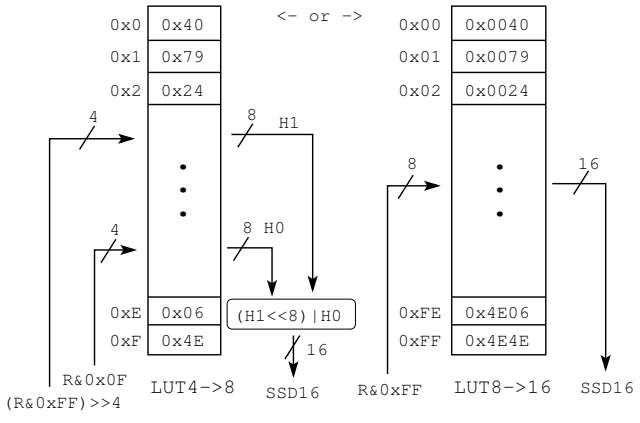

Figure 6: Using lookup tables (LUTs) to map the lower byte of register R to the appropriate 16-bit active-low bit patterns of the dual seven-segment display. Although only 14 bits are required to drive both displays, we use 16 since this matches the width of both the parallel I/O port on the NIOS as well as the half-word "sthio" assembly instruction.

lower eight bits of the contents of a register and displaying them in hexadecimal format on the two SSDs. As such, the system should be able to correctly display all 256 possible numbers ranging from 0x00 to 0xFF. The students are then provided assistance to lead them toward the following two potential solutions.

In the case of both the hardware and software approaches, the students make use of the NIOS embedded soft processor that is also instantiated alongside their designs in the FPGA fabric. Using a cross compiler and monitoring program on their laptops, the students are able to write the necessary assembly code to output the contents of the register to the parallel I/O port, whether it is connected directly to the SSDs or via their display driver logic (Figures 5 and 7).

4.1 The Hardware Approach

In the hardware approach, the students are expected to explore the fundamental underlying issue here, which is to derive expressions that will correctly drive all seven independent LEDs that make up the SSDs. Although such chip models as the 7447 are already available in the IDE library that will perform binary to BCD or binary to HEX display decoding, we do not allow the students to make use of them in this case. Furthermore, they must come to the realization that the same hardware they derive for one display will also work for the second display (Figure 5). This helps reinforce the concept of design reuse.

Solving this problem amounts to populating a 16 row truth table with the appropriate 1's and 0's that will assert the correct LEDs within the SSD. After the minimization phase, the students input their design schematics into the IDE, compile and then load their designs onto the prototyping board.

4.2 The Software Approach

After having finished the purely hardware-based implementation of the SSD drivers, the students have now seen a functioning project and are familiar with the challenges associated with building these structures from the ground up as well as the relative simplicity of the assembly code that, once acquired, merely loads the least significant byte of a given register onto the SSD displays. At this point, they are presented with the reminder that now they must implement the *same behavior* without the use of their hardware SSD drivers by connecting the NIOS parallel I/O ports directly to the individual LEDs that constitute the SSDs.

A common software technique that is used to map numbers to symbol data is through the use of a lookup table. In this case, the least significant byte of register R must be properly displayed in hexadecimal format on two independent SSDs. The students are shown how to determine each of the bit patterns required for each of the possible 16 HEX characters for a single display, taking into account that the SSDs present on the Altera DE2 are active low. They are then encouraged to sketch out physically how the operations will work (Figure 6) as well as to write a sample C program (Listing 1) that mimics the behavior of the assembly code they will write in order to ensure they understand the mechanics of the bit-masking and shifting in conjunction with accessing the LUT. In this way, issues such as function prologue and epilogue are omitted from the thought process temporarily, as this is a concept students are generally learning at the same time in the associated course.

This transition in emphasis from hardware to software design generally presents a challenge for the students. Although the concept of lookup tables is not foreign to them at this stage of their education, judiciously using bit-wise and shifting operators generally is. In particular, although they have been exposed to these operators in their introductory programming classes (in C) and have been refreshed as

```
1  hword retrieveSSD16(int R) {
2     X = R & 0x0F     // get first nibble
3     Y = R & 0xF0     // get second nibble
4     Y = Y >> 4       // move into position
5     LOW  = LUT[X]    // get pattern for SSD1
6     HIGH = LUT[Y]    // get pattern for SSD2
7     HIGH = HIGH << 8   // move into position
8     SSD16 = HIGH | LOW // merge patterns
9     return SSD16
10 }
```

Listing 1: C-like structure of function to take contents of register R, retrieve the correct bit patterns from the lookup table, and coalesce them into a 2-byte result. ***Although this excerpt is given in C, the students are required to write the corresponding assembly code***

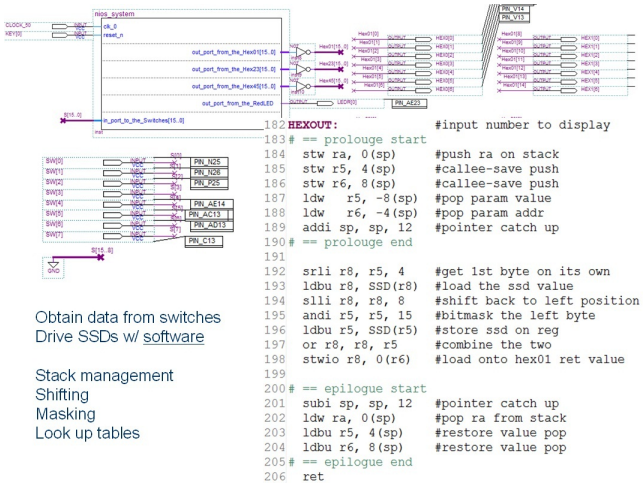

Figure 7: IDE in schematic mode, with the NIOS parallel I/O ports directly connected to the seven segment displays and switches. A fragment of student code is shown that accesses the lookup table once. ***Courtesy of CS student Yosi Benezra.***

to their meaning and behavior in the assembly programming class, they are unfamiliar with how to apply them to solve this problem.

Although this project presents a number of concurrent challenges that cover both basic hardware design as well as fairly detailed assembly programming, many students have expressed that this project allowed them to tie together concepts that had previously seemed disjoint. In addition, our intention is to also allow the students to make comparisons between the two solutions and to critically analyze the tradeoffs involved.

4.3 Tradeoff Analysis

After completing both parts of this project, the students are asked to comment on the benefits and drawbacks of each approach. This typically evokes responses related to how difficult each part was relative to the other in terms of their own effort. In most cases, students feel that the hardware solution is "easier" to implement because it avoids the intricacies of dealing with the lookup table and bit-wise operations. However after we move past this initial response, when pressed further to evaluate each design, they begin to critically analyze the issues involved. Chief among the ideas discussed are:

- **Flexibility** The LUT approach is more flexible in that the characters displayed on the SSDs can easily be changed by simply altering the contents of the array. This is in stark contrast to the challenge of re-deriving Boolean expressions for a similarly altered truth table.

- **Speed** The hardware approach allows the display to be driven with far fewer instructions because there are fewer operations to perform on the contents of register R and there are no memory accesses to any LUT. Furthermore, there is no on-chip memory usage for storing a LUT, which leaves more real estate on the FPGA for other hardware to be instantiated, if desired.

- **Time versus Space** Using a larger LUT allows for half as many assembly operations; however, in doing so, we use 16 times as much memory to store the array.

This sort of tradeoff analysis reinforces the concept of matching a design with a specific set of requirements. For example, in this case, the requirements they were given did not include the need to provide for a changing set of characters to be displayed on the SSDs; therefore, the hardware solution might seem to be the "better solution" given that it is both faster and requires less on-chip memory. However, the software solution provides a modicum of flexibility that more easily accommodates incomplete or changing specifications, an unfortunately common occurrence in the field.

5. SURVEY RESULTS

During the Fall 2010 semester, a short, anonymous, and voluntary survey was given to students in two different courses, CSCI 210 and 310, using BlackBoard. CSCI 210 is a course that covers digital logic design, computer organization, and introduces assembly programming while CSCI 310 further covers assembly programming and advanced logic design with a significant focus on computer architecture.

Of interest here are three questions from each surveyed course. Question 1 was different in each course; however, questions 2 and 3 were the same for both courses.

- [**Q1-210**] Compared to lecture alone, has the usage of the NIOS processor and Altera Monitoring Program helped you better understand how assembly programming works?
  (a) none, (b) not much (c) somewhat (d) a lot

- [**Q1-310**] Has your prior experience with the Altera DE2 and the NIOS processor helped you to better understand how hardware and software interact?
  (a) none, (b) not much (c) somewhat (d) a lot

- [**Q2**] Given the choice, do you feel it is more rewarding to see your programs execute on the Altera DE2 boards

Survey Results					
Course	Question	(a)	(b)	(c)	(d)
CSCI 210	Q1	0	2	5	2
	Q2	7	0	2	NA
	Q3	4	1	4	NA
CSCI 310	Q1	0	0	2	8
	Q2	6	0	4	NA
	Q3	5	0	5	NA

Table 1: Summary of the results of the Fall 2010 survey of both CSCI 210 and 310 students. Across both courses, of the students that had a preference, 100% preferred the use of the Altera DE2 board over a simulator for learning assembly programming while 90% preferred the Altera DE2 over breadboarding.

or on a software simulator?
(a) Altera DE2, (b) simulator, (c) no preference

- **[Q3]** Given the choice, would you prefer to implement and debug your digital logic designs using a breadboard and discrete logic gates or by using the QuartusII IDE and Altera DE2?
(a) Altera DE2, (b) breadboard, (c) no preference

The results of the survey are presented in Table 1. Nine of the 14 students enrolled in CSCI 210 responded to the survey, while 10 out of the 17 enrolled in CSCI 310 responded. *Note that all enrolled students in CSCI 310 were previously exposed to the Altera DE2 and NIOS processor in the prerequisite course, CSCI 210.*

The general trend across both courses is that, of the students that had a preference, 100% preferred the use of the Altera DE2 board and the Altera monitoring program over the use of a simulator for executing their assembly programs. To a lesser extent, 90% of those that responded preferred the use of QuartusII and the Altera DE2 over the use of breadboards and discrete logic gates for implementing their digital logic designs. Additionally, 78% of CSCI 210 students indicated that using the NIOS processor combined with the Altera Monitoring program played a significant role in helping them to better understand assembly programming over lecture alone. Moreover, a full 100% of CSCI 310 respondents indicated that the new FPGA platform played a significant role in helping them understand how software and hardware interact.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we illustrate how to make use of FPGA-based platforms in courses that cover digital logic design and assembly programming while discussing the pro's and con's of their use. We also provide a detailed discussion of a lab-based project that integrates both assembly programming as well as digital logic design in such a way that allows the students to perform a trade-off analysis between using software in the place of a purely hardware-based solution to a common interfacing problem.

We also provide the results of a survey across two different courses where students have been exposed to the FPGA-based platform for the purpose of learning both digital logic design as well as assembly programming. The results are encouraging and, based on our experiences, we plan to expand the use of these platforms across other courses in our curriculum. In fact, we have already begun expansion into our compiler construction course by making the NIOS processor the target architecture.

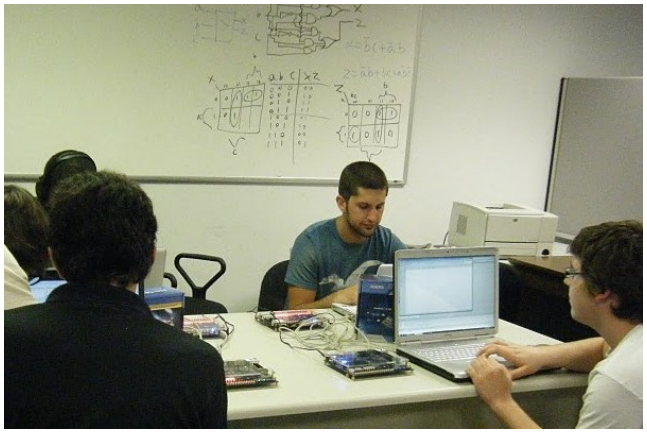

Figure 8: Students making use of a schematic-based IDE to design digital logic circuits for instantiation and deployment on FPGA development boards.

By making use of these powerful FPGA-based systems across multiple courses, not only are students exposed to a technology that is becoming much more prevalent in many different areas in industry, they also benefit from the connecting of concepts and shorter learning curves between courses that come from making use of a common target platform.

7. REFERENCES

[1] T. S. Hall and J. O. Hamblen. Using fpgas to simulate and implement digital design systems in the classroom. In *American Society of Engineering Education: Southeast Section Conference*, 2006.

[2] M. Holland, J. Harris, and S. Hauck. Harnessing fpgas for computer architecture education. In *Microelectronics Systems Education*, pages 12–13, 2003.

[3] Nios ii embedded processors for education. http://www.altera.com/education/univ/software/nios2/.

[4] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface*. Elsevier Morgan Kaufmann, 4 edition, 2008.

[5] T. E. Salem, R. Rakvic, R. Voigt, and S. Firebaugh. Curricula enhancement and thematic learning via undergraduate design projects. In *The 36th IEEE Annual Frontiers in Education Conference*, pages 1–5, 2006.

[6] A. A. Thompson and R. A. Ebel. The military academy reduced instruction set computer. In *American Society of Engineering Education: Mid-Atlantic Section Conference*, 2009.

[7] J. S. Warford. *Computer Systems*. Jones & Barlett Learning, 4 edition, 2009.

[8] Xilinx university program. http://www.xilinx.com/university/.