Digital Logic Course Themed Around Arithmetic

Firas Hassan, Ahmed Ammar, Ziad Youssfi
ECCS Department, Ohio Northern University, Ada, OH, USA
f-hassan@onu.edu, a-ammar@onu.edu, z-youssfi@onu.edu

Abstract—**Work in Progress—In-depth coverage of arithmetic operations in an introductory digital logic course equips students with the background knowledge necessary for high-level Computer and Electrical engineering and Computer Science courses. In this paper, we propose a new method of teaching digital logic based on including a module on fixed-point and floating-point arithmetic operations during the first four weeks of the course, and then theming the whole course around this module without sacrificing the recommended course outcomes. The paper shows that theming the whole course around arithmetic operations helps students to achieve the course outcomes, makes the course interesting, ties the course to real-life applications, and teaches students how to implement and compare alternative designs.**

Keywords——Digital logic, arithmetic operations, hardware implementations.

I. INTRODUCTION

Digital logic is one of the major courses in Electrical and Computer Engineering and Computer Science programs. It is an introductory course taken by freshman and/or sophomore students, and it introduces students to the basic concepts of digital logic. Digital logic is important and required for some advanced courses that students take in the next years [1]. Therefore, the course needs to be taught effectively, efficiently, and comprehensively.

Different strategies, approaches, and methods of how to teach digital logic have been proposed in the literature. One method to teach digital logic is by using practical laboratory work on breadboards [2]. However, according to [3], this method is not viable because it only teaches students how to wire small scale integration devices and how to test them. It does not expose students to the real design, or give them the chance to practice some basic concepts of digital logic. Hence, an alternative method can be used, i.e., teaching digital logic by using Computer Aided Design (CAD) tools [3]. Teaching digital logic using CAD tools allows students to observe, explore, and manipulate device characteristics [4]. Also, it allows students to implement and test complex components and systems. Thus, students will have more chance to practice various concepts of digital logic. Note that with CAD tools, students can implement and test systems on programmable logic devices such as Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD) [5].

One method to modernize digital logic is by incorporating Very High Speed Integrated Circuits Hardware Description Language (VHDL) in teaching the course [6, 7]. Incorporating VHDL in teaching digital logic enhances students understanding of combinational and sequential logic circuits and their thinking design skills [7]. Also, it equips students with the skills that would meet the industry requirements.

Another approach to teach digital logic is by using practical examples [8] (i.e., real-life applications) instead of only using standard example circuits such as binary adders, decoders, multiplexers, etc. The standard circuits are the essential parts in designing larger systems, however, they might not capture students interest as much as practical examples [8]. In addition, using practical examples ties the material to real-world systems more than the standard circuits. Besides, if a practical example is selected at the beginning of the term and the laboratory experiments are themed around it, then by the end of the term students will learn two more concepts, i.e., reusability and system integration [9]. Lastly, making the end project of the course themed around games is another approach to teach digital logic, which can make the course more interesting and practical [10].

In this paper, we propose a new method of teaching digital logic, i.e., based on booting the course with a module on fixed-point and floating-point arithmetic operations, then theming a big portion of the laboratory experiments around this module. Different arithmetic operations are described in VHDL, simulated using a discrete-event simulation tool, synthesized using CAD tools, and implemented on an FPGA device, then tested on a developing board. The experiments on arithmetic operations makes the course interesting to students. Also, all course outcomes that are recommended by the Computer Engineering (CE) body of knowledge [1] can be linked to the arithmetic operation experiments. In addition, through the coverage of arithmetic operations, extra high-level course outcomes can be added to the course. Finally, the in-depth coverage of arithmetic operations in this introductory course is very useful for higher level computer engineering courses.

The rest of the paper is organized as follows. Section II describes how the proposed lab experiments are implemented and assessed. Section III lists the course modules and their associated arithmetic operation experiments. Section IV connects the outcomes of these experiments to the CE body of knowledge, and lists the high-level course outcomes that are also connected to these experiments. Section V discusses the relationship between the arithmetic operations covered in this course and high-level courses in computer engineering. Section VI analyzes students' feedback and evaluation of the course. Section VII concludes the paper.

II. LABORATORY TEACHING AND ASSESSMENT

The most recent offering of the course has 4 sections and a total of 76 students (students from Computer and Electrical Engineering and Computer Science programs at the freshmen and sophomore levels), and each section has 18 to 20 students. Our digital logic laboratory has 12 bench stations. Each bench station contains an oscilloscope, FPGA board (BASYS 3), and Windows desktop computer equipped with ModelSim, Vivado, and Matlab. The laboratory is intended for 24 students. Therefore, students in each section conduct the laboratory experiments in groups of two.

Over a 15-week semester, the laboratory meets for two hours and 45 minute sessions 14 times. In each laboratory session, we spend the first hour in introducing the objective of the experiment, and explaining how the hardware can be designed theoretically and described in VHDL. We provide students with most of the VHDL files required to implement the hardware. After that, in the remaining time, we ask students to simulate that hardware using ModelSim, synthesize it using Vivado, implement it on FPGA, and then test it, following the procedures provided on the laboratory handout. As an exercise, students sometimes are asked to answer conceptual questions using the simulation and implementation results, or modify the VHDL files in order to meet some design constraints or requirements. Most often, we finish the laboratory session by a class discussion on alternative ways of implementing the hardware, and what is the advantages and disadvantages of each design.

Students are required to write a formal laboratory report for the odd numbered laboratories. A formal laboratory report must be typed and titled, and contain at least the four following sections: objective, procedure, results, and conclusion. Students are given instructions of what to include in each section. Formal lab reports are assessed such that each section is worth 25% of the report total grade. For the even numbered laboratories, students are required to submit an informal report; one report per group. Informal lab reports are assessed differently depending on the laboratory experiment. For example, in Experiment 8, students were asked to convert the 4- bit comparator to a 3-bit comparator and submit that as an informal report. In that informal report, students were asked to include the VHDL files, simulation results, synthesizing results, and a paragraph to comment on the results. The assignment was graded out of 10 as follows. VHDL files are worth 30%, simulation and synthesizing results are worth 50%, and the comment on the results is worth 20%.

III. COURSE MODULES AND ASSOCIATED EXPERIMENTS

The proposed course consists of six modules that are supported by twelve arithmetic operation experiments, and two introductory experiments that are essential for the implementation of the remaining twelve experiments. This section gives the title and brief description of each experiment in the order of their offering based on the course modules.

Experiments 1 to 4 are implemented to support the first course module that covers arithmetic operations. Because this module is given before the introduction of gates, all these experiments do not include hardware implementation and testing and are only based on simulation.

Experiment 1: **Hardware description and simulation of a parametrized large numbers comparator**. Students simulate a high-level description of a comparator that first subtracts B from A and check the sign of this subtraction to find out if $A \geq B$ or $A < B$. If the result of the first subtraction is positive, it is subtracted from 0 and the sign is checked to figure out if $A = B$. The bit-width of A and B is kept as a parameter and students are asked to play with large bit-widths that goes beyond the capability of the IEEE floating point double precision, covered in their programming course.

Experiment 2: **Fixed-point hardware description and simulation of $sinx$ and $cosx$ using Taylor series**. Students use Matlab to figure out how many terms of Taylor series should be included to get a good approximation of $sinx$ and $cosx$. Then, they describe/simulate the arithmetic operations required to implement the series using fixed-point number representation. Students learn how to implement constant multiplications using shift-and-add operators and how to keep only the significant bits from a general multiplication which includes dropping both most and least significant bits.

Experiment 3: **Hardware description and simulation of 4-bit floating point division**. This lab experiment focuses on the hardware description and simulation of floating point division. Both the dividend and divisor are given as floating point numbers with mantissa and exponent. The mantissa of the result is normalized and its exponent is checked for overflow. Students learn how to first find the reciprocal of the divisor mantissa from a look-up-table. Then, they use floating point multiplication to multiply the mantissa of the dividend with the reciprocal of the divisor mantissa. Finally, students use floating point addition to subtract the exponent of the dividend from the exponent of the divisor.

Experiment 4: **Matlab simulation of a new integer division algorithm**. This experiment serves as a teasing exercise for a later lab where the algorithm described in [13] is implemented in hardware, utilizing a large number of combinational and sequential circuits that are covered in the class. It also includes a discussion on modulo operators and finite fields and their use in encryption techniques.

Experiments 5 and 6 supports the course module that covers Boolean algebra and basic logic circuits. Although both experiments do not involve arithmetic operations, they are very crucial for the remaining laboratory experiments that are based on arithmetic operations. For example, the multiplexed display that is designed in Experiment 6 is utilized as a component in multiple laboratory experiments to display the results of the arithmetic operations.

Experiment 5: **Synthesis, implementation and testing of the basic 2-input gates on an FPGA board**. This experiment is an introduction to the CAD tool that is used for synthesizing hardware description and implementing it on an FPGA device. Students learn how the FPGA device can be reconfigured to different 2-input gates. Students also find the truth table of every gate by testing the different input combination on the board. This lab also includes a discussion of gates and their applications that includes the following topics.

- Using 2-input gates as input inhibitors by changing one of their inputs to a control signal.
- Generalizing all gates to m-inputs.
- Using gates as input decoders.
- Using the m-input Xor and Xnor gates as parity generators and checkers.

Experiment 6: **Hardware implementation and testing of multiplexed seven-segment display**. Students learn how to utilize time division multiplexing to display a 16-bit input as four hexadecimal digits on the four seven segment displays. First, students utilize k-maps to simplify the expressions of the seven-segment decoder. Then, they use two 4-to-1 multiplexers controlled by a slow 2-bit counter to time-multiplex the display of the different hexadecimal digits while deceiving the human eye. An n-bit fast counter is used to divide the clock and drive the 2-bit slow counter. Students experiment with the length n of the fast counter and find its effect on the display of digits on the seven segment display. Students find the optimum frequency for deriving the slow counter to deceive the human eye and make it see the four digits displayed at the same time.

Experiments 7 to 9 support the course module that covers combinational circuit design. They focus on implementing and testing basic arithmetic operations such as addition, subtraction, absolute, comparison, and multiplication.

Experiment 7: **Hardware implementation and testing of Ripple Carry Adder (RCA)**. Students implement an 8-bit adder and 8-bit subtractor using RCA configuration, and test them on the board. Students test different 8-bit unsigned input combinations using switches, and find their sum and difference displayed in hexadecimal on the seven segment display. Two extra LEDs are used to display the most significant bits of addition and subtraction. Students modify the hardware of the subtractor by sending a clock to the carry-in of the least significant full adder, then use this new hardware to measure the propagation delay. Using an oscilloscope, students measure the delay between the clocked carry-in and the carry-out from the final full adder. A class discussion is held on the need of some applications for less propagation delay. Such applications may require more complicated hardware implementations of addition to achieve the goal of less propagation delays such as carry-look-ahead-adder and carry-select-adder.

Experiment 8: **Hardware implementation and testing of a 4-bit absolute comparator**. This experiment uses two hardware components. The first component finds the absolute values of two signed 4-bit inputs. The second component compares the absolute values and produces an AGB flag and Eq flag. To create the absolute component, students start from the truth tables, find the Boolean expressions, simplify the expressions using k-maps, and describe them in VHDL. To implement the comparator, students use an algorithm that compares bit slices and ripple the results of the comparison from the most significant to the least significant bit. Students test and verify different combination of inputs controlled by switches, and check the two flags that are displayed on LEDs.

Experiment 9: **Hardware implementation and testing of 8-bit signed array multiplier**. Students modify an unsigned array multiplier to perform signed multiplications. To achieve this, students sign extend all partial products and intermediate sum results, and 2's-complement the final partial product. Students verify that their modified multiplier is working by testing different 8-bit multiplier/multiplicand inputs controlled by switches, and checking the results displayed on the four seven segment displays in hexadecimal.

Experiments 10 and 11 support the course module that covers sequential circuit design. They focus on designing binary counters and using them in different applications such as stopwatch and traffic light controller. One approach of designing binary counters is based on taking the previous state and incrementing/decrementing it depending on whether the counter is counting up or down. Truncated counters can be implemented using comparators that can check the maximum/minimum value and reset/preset the counter to the initial state. Both of these experiments can serve as an application of RCAs and comparators.

Experiment 10: **Design, implementation, and testing of a stopwatch**. The design of this stopwatch requires two types of counters. The first type is an n-bit regular counter used to divide the clock, and achieve the 100Hz frequency required to measure the 100th of a second. The second type is a 4-bit truncated counter that is used to display decimal numbers on the seven segment displays. Students consider different techniques to implement the two types of counters including: designing the 4-bit truncated counters on Delay Flip Flops using state tables, designing the n-bit regular counter using AND gates and Toggle Flip-flops, and utilizing RCAs and comparators to implement both types of counters. Students instantiated one copy of the n-bit counter to divide the clock and four copies of the 4-bit truncated counter to display the output of the stop watch from 100th of a second to 10 seconds.

Experiment 11: **Design and test a cross-section traffic light controller**. The controller has different modes that includes flashing red on both directions, flashing yellow on one direction and red on the other, and the standard cycle that moves between red, yellow, and green on both directions. The different modes have priorities with the flashing red being the highest priority and the standard cycle being the lowest priority. Students use different outputs from an n-bit counter to achieve different clock frequencies because the flashing light modes work at a different frequency than the standard cycle mode. Also, students implement a simple finite state machine that tests the most significant bit of a 3-bit slow truncated counter, and use it to generate the outputs of the standard cycle. Students prioritize modes through the use of cascaded two input multiplexers.

Experiments 12 and 13 support the course module that covers control and data path design. They focus on combining different combinational and sequential logic components to achieve complex hardware implementation of arithmetic operators such as pipelined signed array multiplier, serial multiplier, and integer division.

Experiment 12: **Implementing and testing an 8-bit signed serial multiplier and comparing its performance against a pipelined version of the 8-bit signed array multiplier**. Serial multipliers require a controller and a data path design. On every clock cycle, the controller checks one bit of the multiplier from least to most significant, and orders the datapath to shift, add shift, or complement add and shift. The datapath is created out of a shift register, a RCA, and inverters. The content of the shift register after 8 clocks is the output of multiplication. The time performance of this serial multiplier is compared to a pipelined version of the signed array multiplier, implemented in Experiment 9, by constraining both designs in time before synthesis and implementation. Students add a component to both multipliers to transform their outputs from two's complement to sign and magnitude. This component contains a new method for finding the absolute value that is based on locating the least significant high bit and inverting all the bits after it. The magnitude of the multiplication is displayed on seven segment display and the sign is displayed on an LED.

Experiment 13: **Hardware description and simulation of a 32-bit integer divisor**. This experiment is based on the new algorithm for integer division that is introduced through Matlab simulation in Experiment 3. The importance of this algorithm is that it utilizes most of the combination and sequential components that are covered in the class to achieve integer division. The components are priority encoders, decoders, multiplexers, subtractors, comparators, and registers. Students integrate these components to build the hardware divider.

Experiment 14 supports the last module of the course that covers the computer revolution, Moore's law, different types of memories, the importance of memory hierarchy in processors, and an introduction to basic Instruction Set Architecture (ISA) processors.

Experiment 14: **Design, implementation, and integration of a 32-bit register file and Arithmetic Logic Unit (ALU)**. This experiment focuses on two basic components of the ISA, i.e, the register file and ALU. The hardware of a 32-bit register file and ALU is described and simulated. Students understand how these two components can be integrated to run basic assembly language instructions.

IV. COURSE OUTCOMES LINKED TO LAB EXPERIMENTS

Table 1 summarizes the link between the lab experiments and the course outcomes that are recommended by CE body of knowledge [1]. The argument behind linking the different experiments to the first five recommended course outcomes can be simply deduced from the description of the labs in the previous section. In fact, most of these experiments are offered to support course modules that cover these course outcomes. Since experiments 5 to 12 are all implemented and tested on an FPGA board, they are linked to the design with programmable logic outcome and the system testing outcome. System design constraints are typically power, area, and speed. The area constraint is covered through the utilization reports of the different FPGA resources such as look-up-tables and Flip-Flops, which are generated by the CAD tool in experiments 5 to 12 as well as 13 and 14 that are not tested on the board, but are still synthesized to check their utilization. Experiments 7 and 12 focus specifically on time constraints. Although the CAD tool can generate power estimates, this is not yet implemented in the class. Instead, the power constraint is covered through class discussions during the lab time.

Table 1. Link between course outcomes and lab experiment

Number systems and data encoding	1 to 4
Boolean algebra and basic logic circuits	5 and 6
Combinational circuit design	7 to 9
Sequential circuit design	10 and 11
Control and data path design	12 to 14
Design with programmable logic	5 to 12
System testing	
System design constraint	7 and 12

The above experiments can also be linked to high-level course outcomes such as, fixed-point and floating-point arithmetic operations, modular design, pipelining, parallelism, hardware accelerators, latency, throughput, and criteria for comparing alternative designs. These high-level course outcomes are an added value to the course, and this is due to the fact that the course is themed around arithmetic operations. Fixed-point and floating-point arithmetic operations are clearly covered in Experiments 1 to 4. Modular design which is based on replicating the same hardware for every bit slice is covered in the RCA design of Experiment 7, the comparator design of Experiment 8, one counter design from Experiment 10, and the ALU of Experiment 14. Pipelining and parallelism is covered using the array multiplier that is designed in lab 9 and pipelined in lab 12. Note that the latency of the pipelined array multiplier is different than its throughput. While the array multiplier can accept inputs on every clock cycle, there is a latency of several clocks between its input and output. On the other hand, the serial multiplier of lab 12 and the integer divider of lab 13 cannot accept an input on every clock cycle. Hence, their latency is equal to their throughput. Both serve as a good example of hardware accelerators that require some handshaking signals such as ready, start, and done. While the latency/throughput of serial multiplier is totally predictable, the latency/throughput of integer divider is unpredictable because it is based on the high bits in the divisor.

The proposed experiments present a number of alternative designs of arithmetic operations. Three alternative designs of comparators are implemented in Experiments 1, 8 and 14. Alternative designs of adders are discussed in Experiment 7. The serial and array multipliers are two alternative designs of multiplication. The new integer division algorithm is compared against alterative techniques in lab 13. Two techniques for finding the absolute of numbers are discussed in lab 8 and 12. All these alternative designs are assessed based on time, power, area, latency, and throughput.

V. ADVANTAGES FOR HIGHER LEVEL COURSES

The new CE body of knowledge has put more emphasis on parallel architectures, implementation of complex embedded systems, and low power operations. In-depth coverage of arithmetic in digital logic course gave our program more time to cover these changes in upper level courses. We have been using the time in Computer Architecture course to cover parallel algorithms on architectures such as multicores, single-instruction-multiple-data (SIMD), vector processors, simultaneous multithreading (SMT), Graphical Processing Units (GPU) architecture. Moreover, this coverage enables our students to design and test a single-cycle processor that includes an arithmetic ALU at a higher abstraction level. We also used the time to cover the newly emphasized software techniques for embedded systems in our embedded real-time applications, including complex input/output, communication, and multiple thread synchronization and buffering for real-time operations. The freed time also allows us to address in-depth topics related to robotics and automation. The in-depth coverage of arithmetic operations also improves student programming skills in C++ and Java by helping them understand the underlying principles for variable types such as float, double, and integer. Students are then able to make informed choices for variable types and limitations early in their programming experience.

VI. STUDENT FEEDBACK

Students were asked to fill in a questionnaire to get feedback on the laboratory experiments. The questionnaire consists of the following five statements:

1. The lab exercises helped me understand the different course outcomes.
2. The lab exercises made the course more interesting.
3. The lab exercises connected the course outcome to practical applications.
4. The lab exercises showed me that all arithmetic operators can be built using basic combinational and sequential logic components that are covered in the class.
5. The lab exercises compared alternative designs of some arithmetic operations based on power, area, and speed.

Each statement can be rated from 1 to 5, where 1 = Strongly Disagree and 5 = Strongly Agree. Out of 76 students, 63 students responded to the questionnaire. The weighted average of students' feedback is 3.6, 3.6, 3.9, 4, and 3.6 for statements 1 to 5, respectively. Overall, the survey indicates that majority of students are satisfied with theming the laboratory experiments around arithmetic operations.

Arithmetic operations have been gradually incorporated to the course over the last three years of offering the course. The students' evaluation from these three years shows the positive impact of the arithmetic operations experiments on the students' experience with the course. When asked how much do they agree with the statement "Over all the course was organized", the weighted average of their response has increased from 4.17 to 4.18 to 4.36. When asked how do they agree with the statement "The instructor overall teaching of the course was effective", the weighted average of their response has increased from 4.04 to 4.18 to 4.56. Overall, students mentioned in their comments that they could piece all the course material together through arithmetic operation.

VII. CONCLUSION

This paper proposes an in-depth coverage of arithmetic operations in digital logic. The proposed approach adds extra high-level course outcomes while still covering all recommended outcomes. The proposed approach also equips students with the background knowledge necessary for future courses in computer engineering curriculum, and makes the course more interesting. In the future, entrepreneurial aspects will be added to the course to engage the curiosity of the students, and help them establish connections between the course outcomes and real-life applications to be able to create value and solve future problems.

REFERENCES

[1] E. Durant *et al*., "CE2016: Updated computer engineering curriculum guidelines," *2015 IEEE Frontiers in Education Conference (FIE)*, El Paso, TX, 2015, pp. 1-2..

[2] Show them NAND gates and they will come, Barrett, S.F.; Hamann, J.; Coon, D.; Crips, P.M.; Pierre, J., Computers in Education Journal, v 17, n 2, p 26-36, April/June 2007.

[3] M. Johnson and B. Craig, "Computer systems pedagogy using digital logic simulation," *IEEE International Conference on Computers in Education, 2002. Proceedings.*, Auckland, New Zealand, 2002, pp. 703-704 vol.1.

[4] Z. Stanisavljevic, V. Pavlovic, B. Nikolic and J. Djordjevic, "SDLDS—System for Digital Logic Design and Simulation," in *IEEE Transactions on Education*, vol. 56, no. 2, pp. 235-245, May 2013.

[5] A. Alasdoon, P. Prasad, A. Beg, A. Chan, "A recent survey of circuit design tools for teaching", *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2013.

[6] R. Rodriguez-Ponce and J. Rodriguez-Resendiz, "Integrating VHDL into an undergraduate digital design course," *Proceedings of 2013 IEEE International Conference on Teaching*, Assessment and Learning for Engineering (TALE), Bali, pp. 172-177, 2013.

[7] H. Z. Abidin, M. Kassim, K. A. Othman and M. Samad, "Incorporating VHDL in teaching combinational logic circuit," *2010 2nd International Congress on Engineering Education*, Kuala Lumpur, pp. 225-228, 2010.

[8] J.P. Hoffbeck,"Using practical examples in teaching digital logic design", *121st ASEE Annual Conference & Exposition,* American Association for Engineering Education, june 15-18, 2014.

[9] L. A. Slivovsky, B. J. Mealy and A. A. Liddicoat, "Work in progress - a unifying set of experiments for digital design I," *Proceedings Frontiers in Education 35th Annual Conference*, Indianopolis, IN, 2005, pp. S2G-29.

[10] Kharma N. and L. Caro., "Magic Blocks: A Game Kit for Exploring Digital Logic," *American Society for Engineering Education Annual Conference*, 2002.

[11] F. Hassan, J. L. Magalini, V. de Campos Pentea, and J. E. Carletta, "A Booth-like modulo operator," *58th IEEE International Midwest Symposium on Circuits and Systems*, August, 2015.