CUSTOM GRAPHICAL SIMULATORS FOR VHDL LOGIC DESCRIPTIONS AND THE ALTERA NIOS II PROCESSOR

Naraig Manjikian¹ and Valerie Sugarman²

¹Dept. of Electrical and Computer Engineering, Queen's University, Kingston, Ontario K7L 3N6

²David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1
naraig.manjikian@queensu.ca

Abstract – *This paper describes the rationale for developing custom graphical simulators for courses in digital logic and computer architecture, discusses key features of these simulators in relation to the laboratory hardware for the two courses in question, and explores the learning benefits that can be obtained. The digital logic simulator models switch/pushbutton inputs and LED outputs, and it allows a custom circuit to be specified rapidly in a subset of the VHDL language using predefined signal names and flip-flop elements. The computer architecture simulator provides higher-level modeling of the execution behavior of a 32-bit processor with graphical depiction of register and memory contents. It simulates character input/output, and it models the switch, pushbutton, and LED features of laboratory hardware. When students in the two courses were surveyed about the custom simulators, a majority of them indicated a degree of enhancement to their learning.*

Keywords: Simulation, graphics, digital logic, computer architecture, field-programmable logic devices.

1. INTRODUCTION

This paper describes custom graphical simulators for core second-year courses on digital logic and computer architecture in the Department of Electrical and Computer Engineering at Queen's University. The primary motivation for the development and utilization of these simulators is to enable students to pursue laboratory preparation and independent study in a convenient and efficient manner for enhanced learning benefits, particularly for the practical application of the course concepts.

Computer simulation is commonly used in a variety of engineering courses, and it can aid students in their learning, and also prepare them for professional practice when commercial-grade simulation software is employed. With respect to educational use of simulation for teaching of digital logic and particularly computer architecture, there are a variety of examples with varying complexity and with varying degrees of accuracy in system modeling.

Larus [9] developed the *SPIM* simulator as an educational assembly-language-level computer simulator for the MIPS instruction set architecture. Garcia et al. [4] describe the graphical *p88100* tool that is a unified computer simulator framework to cover a broad spectrum from a simplified processing unit to more sophisticated superscalar processors with caches. Donaldson et al. [3] describe the *DLsym* toolkit that allows gate-level logic to be combined with higher-level modules in schematic form to specify and then simulate computer systems.

At Queen's University, simulators have been used for nearly two decades in computer-related courses. An entirely original simulator, called *QEVBI1* and developed by S. Simmons [8], provides a cycle-accurate model of the Freescale 68HC11 microcontroller and its input/output features. Until recently, it was used in an introductory computer architecture course and a microprocessor interfacing course. In advanced undergraduate and graduate courses on computer architecture, the *SimpleScalar* simulator [2] has been used for many years, with enhancements by N. Manjikian [6,7] for multiprocessor modeling and visualization of cache coherence. The experience with such simulators has influenced the development of the simulators discussed in this paper.

The problem considered in this paper is to enhance student learning in applying concepts for digital logic and computer architecture. Formal laboratory activities provide valuable learning experiences, but further support outside the laboratory is also desirable for laboratory preparation and other independent learning. In addition, reflection of laboratory hardware in learning outside of the laboratory is also desirable.

Possible approaches to address the aforementioned problem include additional tutorial guidance, increased laboratory hours, and open-ended projects beyond formal structured laboratory activities. For existing curricula that are characterized by a reasonably large number of formal contact hours and a reasonably heavy workload, such solutions to further increase content and contact hours may not always be feasible. An alternative approach is to provide additional learning resources, including educa-

tionally-oriented custom simulators, and allow students to informally manage the balance of their time between using such resources and pursuing other learning activities.

This paper reflects the decision to develop and utilize simulators as learning aids for digital logic and computer architecture. The most significant aspect of this decision is to closely reflect laboratory equipment based on field-programmable chips for implementing logic circuits and single-chip computer systems. The graphical displays and interactive features of the simulators are therefore familiar to students. The remainder of this paper provides the background for the courses in question, then describes the rationale, relevant features, and learning benefits of the simulators. A summary is also provided for results of surveys in the courses to obtain feedback from students.

2. BACKGROUND FOR COURSES

The two courses – *ELEC 271 Digital Systems* and *ELEC 274 Computer Architecture* – are in the curricula of five programs in the Faculty of Engineering and Applied Science at Queen's University: Electrical Engineering, Computer Engineering, Mathematics and Engineering (Computing and Communications Option, Systems and Robotics Option), Engineering Physics (Electrical Option and Computing Option), and Mechanical Engineering (Mechatronics Concentration). They are core courses for all programs/options except Mechanical Engineering.

ELEC 271 is a lecture-only course covering topics in digital systems such as Boolean algebra, optimization of combinational logic functions, latches and flip-flops for sequential logic, and state-machine design [1]. The course emphasizes the implementation technology of field-programmable gate-array (FPGA) chips and uses commercial hardware and software tools from Altera Corp. as primary examples. Practical application of course concepts (specification, simulation, and implementation of logic circuits using FPGA design software and chips) is pursued in a separate laboratory course using the hardware shown in Fig. 1. Practical aspects are also discussed and demonstrated in ELEC 271 lectures and tutorials using the same hardware and software.

ELEC 274, on the other hand, has a dedicated laboratory to complement lecture content on computer architecture. The subsequent course – *ELEC 371 Microprocessor Interfacing and Embedded Systems* – also has a dedicated laboratory. Previously, ELEC 274 and ELEC 371 used the 8-bit Freescale 68HC11 processor architecture. Now, ELEC 274 and ELEC 371 have been revised together to use a 32-bit architecture and a common textbook. The Altera Nios II processor [11] has been selected as the primary learning example. The common textbook [5] encompasses instruction set design, assembly-language programming, basic input/output, and processing unit design for ELEC 274, and advanced interfacing and embedded-

system design issues for ELEC 371. In both courses, software-oriented laboratory activities involve programming for a Nios II implementation in the FPGA chip on the board shown in Fig. 1, supported by the open-source GNU software tools adapted by Altera. Hardware-oriented laboratory activities involve using a custom VHDL description of the logic for a simplified 32-bit processor that students simulate, modify, and implement in the FPGA chip on the board shown in Fig. 1. In this manner, there is continuity in the application of contemporary FPGA technology for digital systems and computer architecture across the three relevant courses, ELEC 271, 274, and 371.

Fig. 1. Altera DE0 board with Cyclone III FPGA chip.

The practical application of concepts in ELEC 271 and ELEC 274 is closely linked to laboratory hardware and relevant CAD software. For digital logic, the full-featured Altera CAD software is available at no cost and can be used by students to synthesize and simulate circuits on their personal computers. Thus, independent practical learning for digital logic is certainly possible outside of the laboratory without direct use of the hardware shown in Fig. 1. Nonetheless, the sophisticated CAD software does require time and effort to learn and use. Section 3 describes a custom simulator as an additional aid to students in acquiring practical knowledge of digital logic.

For computer architecture, on the other hand, the Altera educational software tools are oriented toward code generation and interfacing with the hardware shown in Fig. 1. Hence, there is a greater reliance on the hardware to support practical learning. Altera does provide an instruction-set simulator for the Nios II processor, but it does not directly reflect the input/output features found in the hardware of Fig. 1. Furthermore, it is primarily intended for use within a more complex software development tool. For enhanced independent learning, Section 4 describes a custom simulator for ELEC 274 that is easy to use and that closely reflects the laboratory hardware.

3. DIGITAL LOGIC SIMULATOR

In ELEC 271, students must acquire knowledge of digital logic concepts and their practical application to be adequately prepared for subsequent courses. A challenge to this aim is the fact that many students have limited background on low-level technical details related to digital logic and computer organization, in contrast to the more significant exposure to the established areas of mathematics and physics that underpin other engineering courses. Another challenge is the use of sophisticated commercial hardware and software that places additional learning demands on students. To address these challenges, a custom logic simulator has been developed to aid students in studying logic circuits with a view towards providing a simple means for circuit specification and a simple yet accurate model of the laboratory hardware.

Fig. 2: On-screen appearance of digital logic simulator

Figure 2 shows the on-screen appearance of the logic simulator that has been developed for ELEC 271. It is called the *miniVHDL* simulator for the Altera DE0 board because it accepts logic specifications expressed in a subset of the full VHDL language, and it presents students with a direct representation of the hardware shown in Fig. 1. The simulator uses the graphics library of the X Window system, which is supported by Mac OS and also by Cygwin for Microsoft Windows. Because the custom graphics code has been prepared in a structured manner with parameters governing the number and position of objects, it could be modified to accurately depict other similar laboratory hardware. The software for the simulator is entirely original with one exception. Processing of logic specifications expressed in a subset of the VHDL language is supported by an adaptation of public software implementing the shunting-yard algorithm for parsing arithmetic expressions [10]. The original public software is simplified to accept only single-character identifiers

and arithmetic operators (e.g., '+'). It has been extensively modified to support multi-character identifiers and VHDL logic operators (e.g., 'xor').

To use the simulator, students must specify a logic circuit in a text file, then type a command with the filename to execute the simulator. The software will then read the file and report any syntax errors. If there are no errors, the software will construct an internal representation of the specified circuit and await user input through the graphical interface, i.e., clicking the mouse on a switch or a pushbutton. In response, the software will model the input change to the logic circuit and generate corresponding output changes on the simulated LED displays.

For accelerated independent learning, students are given a template VHDL file with many predefined features that are modeled by the simulation software. The simulated pin connections for the switches, the green LEDs, and the pushbuttons are reflected by predefined port names: `sw0` to `sw9`, `ledg0` to `ledg9`, `push0` to `push2`. The simulator also models the behavior of ten predefined internal flip-flops that each hold one bit (0 or 1). All flip-flops share predefined internal clock (`clk`) and active-low reset (`reset_n`) input connections. Individually, the flip-flops have dedicated data input connections `d0` to `d9`, dedicated data outputs connections `q0` to `q9`, and dedicated load-enable control signals `le0` to `le9`. Finally, there are predefined single-letter signal names `a` to `z` for use in specifying internal combinational logic connected to the simulated pins and internal flip-flops.

To specify a logic circuit, students copy the template VHDL file and insert a modest number of VHDL signal assignment statements at a well-defined location in the file. The simulator parses the lines at this location. This approach maintains a clear focus on conceptual and practical aspects that are most essential for an introduction to digital logic, while also providing gradual exposure to more syntax details. For example, the logic circuit in Fig. 3(a) is specified by the lines of VHDL code in Fig. 3(b).

The template file structure is such that it can be used with no modification to implement a circuit in the actual FPGA chip on the board for laboratory work. All that is necessary is to incorporate the customized file into a project for the Altera CAD software that specifies the correct target chip and appropriate hardware pin assignments for the board. In this manner, students can easily prepare and simulate circuits without relying on the full CAD software before they use the actual hardware in the laboratory. Use of the Altera logic-design and simulation software by students within (and away from) the laboratory is still important, particularly to learn about timing diagrams that reflect logic circuit behavior. Nonetheless, the custom graphical simulator can have a complementary role in the initial introduction of, and then the subsequent extension and reinforcement, of concepts.

```

clk <= push0;
reset_n <= push1;
x <= sw0;
y <= sw1;
g <= x and y;
d0 <= g;
le0 <= '1';
f <= q0;
ledg0 <= f;

```

(b) VHDL representation

Fig. 3: Sample logic circuit specification for simulator

The custom simulator reduces the elapsed time from entering a circuit specification to verifying functionality. It takes approximately 1 minute to type the code in Fig. 3(b). With the commercial CAD software, another 6 minutes is required for a student to prepare a project with a complete VHDL source file, perform compilation, graphically draw input waveforms for a functional simulation, and then perform a simulation to view the output waveforms. With the custom simulator, in contrast, a student need only take 10 seconds to type a command to open the simulator program after entering the code in Fig. 3(b). Then, the student can rapidly click on simulated switches and pushbuttons to step through different input combinations. By observing the resulting output changes on the LEDs, truth tables for combination logic functions can be verified, and the behavior of sequential circuits with flip-flops can be traced.

Fig. 4: Automatic on-screen labeling of inputs and outputs

Finally, the simulator automatically labels simulated elements on the screen with corresponding internal signal names, based on the circuit specification provided by the student. Figure 4 shows an enlarged portion of Fig. 2 to

highlight the labels that are automatically drawn by the software for the circuit specification in Fig. 3(b).

In summary, the *miniVHDL* simulator provides students in ELEC 271 with learning benefits involving exposure to, or experience with, the following aspects:

- relating schematic and VHDL representations,
- mapping external connections to internal signals,
- implementing combinational logic functions,
- using flip-flops and combinational logic together.

The simulator has less complexity, provides an interactive model of the hardware, and reduces time to obtain results.

4. COMPUTER ARCHITECTURE SIMULATOR

For ELEC 274, a custom simulator has been developed to aid students in acquiring knowledge of instruction set design, assembly-language programming, and basic input/output. For many students, this course provides their first exposure to these detailed topics, notwithstanding their constant exposure to sophisticated computing hardware and software. With the lower-level details of computer architecture often masked by that sophistication, the challenge is ensuring that students acquire a good grasp of the concepts and their application in a relatively short period of time to serve the subsequent course on interfacing and embedded systems, ELEC 371. The approach taken for addressing this challenge reflects hardware continuity across courses, as discussed in Section 2.

Fig. 5: On-screen appearance of Nios II & DEX simulator

Figure 5 shows the on-screen appearance of the custom *sim-nios2* simulator developed for ELEC 274 to model the execution behavior of the Altera Nios II processor and the interfacing features of the laboratory hardware where the Nios II processor is implemented within the FPGA. (The same simulator is used in the

subsequent course, ELEC 371.) The graphical representation of the board shows register and memory contents, switch and pushbutton states, LED outputs, and simulated character output. Typing on the keyboard simulates character input. Clicking the mouse on pushbuttons and switches causes input events for simulation. To control the simulator, students click on the three buttons centered below the board: green for continuous simulated execution, yellow for single-stepping one instruction at a time, and red to pause or stop continuous execution.

The graphical view in Fig. 5 does not accurately depict the DE0 board in Fig. 1 for three reasons. First, students in ELEC 274 have already become familiar with the hardware in their preceding coursework. Second, the display requirements for computer architecture simulation are different than for simpler logic simulation. Third, the simulator reflects the fact that Altera has designed compatible DE0, DE1, and DE2 boards for educational use, with the primary differences being that the DE1/DE2 boards have more switches, pushbuttons, and LEDs. Software prepared for the DE0 board does not typically require modification for the DE1/DE2 boards. Hence, the more generic *DEx* designation is used for the simulator.

The software for the *sim-nios2* simulator in Fig. 5 consists of three components. The first component is the back-end instruction set simulator which is an adaptation of the *SimpleScalar* simulation software [2]. This software has been selected because of prior experience and familiarity with its code [2], and also because of the relative ease of adapting it to support the Nios II architecture. The second component consists of entirely original code to model the state of the simulated input/output features, and relevant modifications to the *SimpleScalar* code to detect memory accesses to input/output addresses and invoke the custom software for input/output simulation. The third component consists of entirely original code that uses the X Windows graphics library to draw the on-screen display and update it in response to user interaction and simulation state changes.

For students to generate Nios II code for simulated execution in the *sim-nios2* simulator on their personal computers, it is possible to rely on a full installation of Altera software that combines logic-design tools with open-source code-generation tools adapted by Altera for the Nios II. But a simpler approach is used instead. Altera provides the full source of the Nios II code-generation software. The minimal subset of necessary tools from that software has been separately compiled under Cygwin for Microsoft Windows and under Mac OS. The software package provided to students in the course includes only the *sim-nios2* simulator, the Nios II assembler, the linker, and a few additional utility programs. Students need only type a single ‘make’ command to invoke the assembler and linker automatically for their code. Hence, there is no reliance on the full Altera installation.

The learning benefits of this simulator are provided through features such as the display of register contents and most recently executed instruction as shown in Fig. 6. Furthermore, the contents of memory can be viewed, as shown in Fig. 7. Changes in the register and memory contents can be viewed slowly to aid in detailed learning by using single-stepped execution (clicking on the yellow button in Fig. 5). Even with continuous simulated execution (clicking on the green button in Fig. 5), students can still observe, at human-perceptible speeds, how programs write data in consecutive memory locations.

Students can also write interactive programs with character input/output because the simulator models those features that are found in the actual hardware of Fig. 1. Similarly, interactive programs can be written to use the other simulated features such as the switches and LEDs. Because the simulator closely reflects the actual hardware, the same programs can also be executed in the laboratory without modification. Hence, students can prepare code before a laboratory session, and subsequently repeat or extend their work after a laboratory session.

r14	00000000	r30	00000000
r15	00000000	r31	00000000
PC	0000002c	br	0x2c

Fig. 6: Closer view of register and instruction display

JTAG UART output			
This output is printed by a small program written in Nios II assembly language.			
Memory contents			
1000:	73696854	74756f20	20747570 70207369
1010:	746e6972	620a6465	20612079 6c616d73
1020:	7270206c	6172676f	7277206d 65747469
1030:	6e69206e	6f694e0a	49492073 73736120
1040:	6c626d65	616c2079	6175676e 002e6567
1050:	00000000	00000000	00000000 00000000
1060:	00000000	00000000	00000000 00000000
1070:	00000000	00000000	00000000 00000000

Fig. 7: Closer view of UART output and memory inspector

5. STUDENT FEEDBACK

In Fall 2011 and Winter 2012, students in the courses on digital logic and computer architecture were surveyed near the end of each term about their use of the simulators. For the computer architecture course, ELEC 274, the response rate was 48% for a class of 102 students. Approximately 37% of respondents made weekly use of the

computer architecture simulator, and 65% of respondents used the simulator to prepare for the dedicated laboratory sessions for ELEC 274. As a learning aid, 52% of respondents found the simulator to be quite useful for their learning, and another 33% found it somewhat useful.

For the digital logic course, ELEC 271, the response rate was 62% for a class of 123 students. Less than 10% of respondents indicated that they made weekly use of the digital logic simulator. One factor for this result is that ELEC 271 does not have a dedicated laboratory; practical application of logic concepts is pursued in a separate laboratory course. Although there is a close connection between the lecture course and the laboratory course (including laboratory exercises designed by the lecture course instructor), the reliance on the commercial CAD software in the laboratory may affect student attitudes about utilizing the custom simulator as an additional learning aid. Nonetheless, 44% of respondents in the logic course indicated that the logic simulator was somewhat useful for their learning and another 22% indicated that the simulator was quite useful.

These results from the logic course, in comparison to the results for the other course, indicate opportunities for earlier enhancement of student appreciation for custom simulators as a learning aid, and for encouraging increased utilization of the logic simulator for independent learning, possibly through more in-class demonstrations and practice problems that involve use of the simulator.

6. CONCLUSION

This paper has presented the *miniVHDL* logic simulator and the *sim-nios2* computer simulator that have been developed as custom graphical and interactive learning aids for two courses at Queen's University. The logic simulator closely reflects the laboratory hardware, yet it permits more rapid circuit specification and functional verification to enable learning of important concepts and their practical application. The computer simulator enables detailed learning of instruction behavior through single-stepped execution, and it also enables complete programs with input/output to be tested with faster continuous simulation. Feedback from students indicates that learning benefits are being obtained, and that there is further opportunity to enhance the utilization and benefits for the logic simulator, in particular.

The development of these simulators has been driven by significant revisions in the relevant courses, which will now be expected to remain in a relatively stable form for several years. It is therefore possible to consider dedicating some future efforts toward enhancements. For example, waveform diagrams could be introduced into both simulators, and the computer simulator could be extended with modeling of simple physical systems for virtual hardware/software co-simulations of embedded systems.

Acknowledgements

The development of the digital logic simulator described in this paper was aided by a 2011 Teaching and Learning Enhancement Grant from the Queen's University Centre for Teaching and Learning that supported the contributions of Valerie Sugarman, who was a student in the Department of Electrical and Computer Engineering.

References

- [1] Stephen Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with VHDL Design*. New York, NY: McGraw-Hill, 2009 (3rd ed.), 939 pp. {ISBN: 978-0-07-722143-0}
- [2] Doug Burger and Todd M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH Computer Architecture News*, vol. 25, no. 3, pp. 13–25, June 1997.
- [3] John L. Donaldson, Richard M. Salter, and Rebecca E. Punch, "DLSys: A Toolkit for Design and Simulation of Computer System Architecture," *Proc. Workshop on Computer Architecture Education*, (San Antonio, TX; 13 February 2011), 7 pp., 2011.
- [4] M. Isabel García, Santiago Rodríguez, Antonio Pérez, and Antonio García, "p88110: A Graphical Simulator for Computer Architecture and Organization Courses," *IEEE Trans. On Education*, vol. 52, no. 2, pp. 248–256, May 2009.
- [5] Carl Hamacher, Zvonko Vranesic, Safwat Zaky, and Naraig Manjikian, *Computer Organization and Embedded Systems*. New York, NY: McGraw-Hill, 2012 (6th ed.), 736 pp. {ISBN: 978-0-07-338065-0}
- [6] Naraig Manjikian, "Enhancements and Applications of the SimpleScalar Simulator for Undergraduate and Graduate Computer Architecture Education," in *Proc. Workshop on Computer Architecture Education*, (Vancouver, BC; 10 June 2000), 8 pp., 2000.
- [7] Naraig Manjikian, "Multiprocessor Enhancements of the SimpleScalar Tool Set," *ACM SIGARCH Computer Architecture News*, vol. 29, no.1, pp. 8–15, March 2001.
- [8] Naraig Manjikian and Stan Simmons, "Evolution and Enhancements of a Microprocessor Systems Course," *IEEE Trans. On Education*, vol. 42, no. 4, November 1999. Included in special CD-ROM supplement that is available at <http://www.ewh.ieee.org/soc/es/Nov1999/Begin.htm>.
- [9] <http://pages.cs.wisc.edu/~larus/spim.html>.
- [10] http://en.wikipedia.org/wiki/Shunting-yard_algorithm.
- [11] <http://www.altera.com/devices/processor/nios2/ni2-index.html>.