Digital Circuit Simulator Project at Undergraduate Level

Maddumage Karunaratne
Electrical Engineering Department
University of Pittsburgh at Johnstown
Johnstown, PA, USA
maddu@pitt.edu

***Abstract*—This paper discusses various aspects of student learning based on a project to develop a primitive digital electronic circuit simulator at the undergraduate level within the Computer Engineering program at the University of Pittsburgh at Johnstown, PA. Beyond software programming, the project exposes students to a language compilation step, circuit data base schemes, and simulation techniques in their final semester. Students are to understand a simple text based grammar used to describe the logic circuit, how it is compiled and stored in memory, and the simulation kernel itself before completing the project. The paper also includes a discussion of survey results from students about their experience.**

Keywords— Digital electronic, logic circuit, simulation, project based learning

I. INTRODUCTION

With ever increasing computing power to run software applications, more and more daily tasks and entertainments are being performed by computers. Video games and virtual online games have become a major entertainment for children, young adults, and even for elderly. Those software applications, sometimes combined with hardware modules, such as in the case of car races on Xbox or PS4, can still be classified as forms of simulators developed by the gaming industry. In certain situations, such as air plane piloting, more and more training is being conducted by software simulators embedded into specialized hardware modules to avoid safety and cost issues. In some cases, such as in reconnaissance or combat missions, simulation in a physical or virtual setting is the only way to train before the actual task. Various forms of Simulations are being utilized even more in technology fields. For the last several decades, many tasks during electronic circuit design flow have been performed by computers with dedicated software programs. Today, no one would commercially manufacture any type of electronic device without having many simulations done to verify its correctness and validity.

While most of our electrical and computer baccalaureate graduates may not right away enter careers to design large complex systems, they might still become part of a large team supporting such developments or individual developers of point tools (Apps). These graduates, being computer engineering majors, would benefit more by exposure to underline technology and algorithmic developments [1, 2] required to create such tools so that they would be better qualified to take up challenging positions and pursue dynamic careers. With emerging virtual reality tools, it is beneficial for software and hardware engineers to have some level of exposure to underlying principles in simulation techniques. Additionally, they would still need to understand the trends, adapt, and adopt the technologies to be successful in their careers. Computer Engineering majors take four mathematics, two physics and two chemistry courses in early years. They take several electrical, electronics, software programming, computer architecture, and signal processing courses in sophomore and junior years. Students, before working on this project within the Advanced Digital Systems course, must have used PSpice^{TM} [3] simulator in an electronic course; implemented algorithms in a C programming [4] and an Embedded Systems course; verified many designs in hardware circuit description language [5] using a commercial simulator [6]. Therefore, by the time students face this project, they have used several different commercial simulators in relevant course assignments. Within this Advanced Digital Systems course, they would be exposed to underlying simulation techniques by developing a primitive simulator as a class project.

This paper elaborates the development of a basic logic circuit simulator using C language so that the reader has enough details to adopt it to similar programs if so desired. The project also helped the Computer Engineering program meet the program educational objective of "adapt to technological change," and the student outcome of "having an appropriate mastery of the knowledge, techniques, skills, and modern tools of their disciplines." Part II of the paper is focused on learning objectives with a discussion in circuit model building, while part III outlines the implementation of the simulator project and assessing students. Part IV ponders upon the results of a survey completed by students after they have received the grades for the project. Part V states concluding remarks with possible future plans.

II. LEARNING OBJECTIVES

Simulation technology is vast and complex, and typical electrical or computer engineering undergraduates are not exposed to such theory. However, as an application of project based learning, a primitive simulator was developed where students explored and learned different techniques described below before deciding on a particular version to be implemented. To achieve the final objective of making a simulator, students as a whole group, considered their current skills and knowledge in digital electronics and programming, the effort required, complexity, and time commitments. Since this is only one project out of several projects and lab experiments for the course, students were given about 2-3 weeks for completion. However, it was assigned as individual work to achieve maximum learning.

In a prior Digital Electronics course, students verify their paper design assignments by describing the circuits in Verilog [5] Hardware Description Langue (Verilog HDL). That language was selected instead of VHDL [7] due to its simplicity and minimal verbiage. As stimulus, students develop testbenches in high level Verilog constructs to drive input signals to exercise their designs. It is noted here that the term (circuit) *testing* may be used interchangeably with the term *verification* by many, but the term *verification* means to check the accuracy of the design while (circuit) *testing* is used to identify defective devices after manufacture.

As the entry point into this project, students propagated signal values for many logic circuits on paper while thinking how a program can do the same tasks. One way to develop a high performance simulator is to describe each circuit component or module as a function in C language. A logic element may be described as a C code function, and an instance of that logic element would be an individual invocation of that function with appropriate signal values as data going into the function. A circuit module containing multiple logic elements is modelled like a function itself, which invokes multiples of other functions. The connectivity among circuit elements or modules is imposed by the hierarchy of the function calls. For sequential circuits, data retaining memory elements may be modelled by *static* type local variables.

On a different modelling approach, logic elements in a circuit module may be described by C language statements without the hierarchy of function calls. Then the entire circuit becomes a single C code module (or function). For sequential circuits, states of logic elements or modules may be kept in local *static* variables of the functions. One can surely extend the C language based modelling techniques to use an object oriented language such as Java or C++. In that case, a logic element type becomes a *class* and an instance of logic element becomes an *object* of that class type. However, the concurrent behavior of hardware cannot be modelled by typical C or C++ since such languages lack multithreading support within the language.

One drawback of the above two modelling approaches is that each new circuit needs to be modelled manually while only a limited reuse of prior models may be possible. Another disadvantage is that time delays of circuit components cannot be incorporated into C code modules, and hence the model is only good as a zero delay simulation. Such higher level simulations are suitable at early stage of designs where circuit timing is not considered. However, the main advantage is the speed of obtaining simulation results even for massive circuits such as processor designs. In many commercial chip design settings, functional accuracy of designs is validated by such zero delay software simulation techniques.

Upon an explanation, students were directed to consider event driven circuit evaluation vs. the cycle based evaluation. In event driven techniques, only the change in values are propagated through the circuit. In the cycle based method, the entire circuit is evaluated every time an input is changed. Students, as understandably, chose the simpler cycle based technique which is easy to understand and implement in software. Another key decision made was to how to represent the circuit (logic elements, and their connectivity) in the program memory after reading the circuit using a compilation stage. Connectivity can be maintained using linked lists in C language, but the complexity would increase heavily. Students as a group decided upon a table representation in order to keep the data structures simple. Evaluation of each logic element based on its input values had the C language choices of cascading if-else statements, switch statements, or the faster table lookup method. The choice was left for each student. After initial thinking of 1's and 0's, students realized, when three-state gates are encountered, they would need to enhance the algebra to include X and Z symbols making it more complex.

The industry level simulators are capable of processing circuit descriptions given in an appropriate format and language [**5**, **7**] into a form that it can simulate using an underlying simulation engine. Most of them use event driven simulation techniques allowing timing delays in circuit elements for accurate results. The data to drive the circuit, typically termed as *stimulus data* or *test vectors*, may be given as a data file (as the case with this project), or as a description of signal behaviors using the same language used to model the circuit itself (Verilog or VHDL testbench), but at a much higher abstraction level. Being familiar with truth table formats, students decided on a data file of 1's and 0's as to give stimulus data.

Stimulus data vector file contains 1's and 0's in a matrix form. Each line (row) in that file represents a set of values for all the inputs of the circuit to be applied at the same time. The next set of input values is in the next row, and so forth. Since this is zero delay timing, each row of input values has to be propagated through the circuit until the circuit values become stable (not changing anymore). If the evaluation of circuit elements has not ceased after many iterations, those elements are considered to have formed internal feedback loops leading to oscillations. Further evaluations of those unsettled elements are to be stopped to suppress fruitless processing. Final values of oscillating elements cannot be determined since they are not stable. For such oscillatory circuit nodes, unknown logic value X is placed, which would eventually cease feedback path evaluations allowing the simulator to apply the next row (vector) of values from the stimulus file, and to make progress. After circuit values have become stable for each input value row (vector), output signal values are printed out along with input values.

III. Project Implimentation and Learning

To keep the project at undergraduate level, the circuits were limited to contain only the typical combinational elements (*and, nand, or, nor, not, buf, xor, xnor, bufif*). Also a very simple grammar was devised for ease of syntax parsing, and error checking. While the grammar resembles Verlog HDL to some extent, it is far simpler and constrained. Fig. 1 shows an example circuit with 3 inputs and 3 outputs connected to 8 logic gates. For the grammar, sstudents picked the line oriented format with every line ending with ';' letter, and the first word in each row is a keyword (*module, endmodule, input, output, and, nand, or, nor, not, buf, xor, xnor, bufif, logic_0, logic_1, logic_x, logic_z*). Circuits were restricted to single bit Boolean variables without multi bit vectors. Fig. 2 illustrates the circuit in Fig. 1 using this simple grammar. First line is the module name; next line is *input list* followed by *output* declarations. Between the third line and *endmodule* keyword, logic elements may appear in any order. For such rows, the second word is the instance name of the element. The third word is the output signal from the element. The rest of entries of an element declarations are the input signals to that element, limited to 3 inputs which is easy to extend if students wish to. Connecting wires are implied when they appear, such as N1, N2, N3, N4, and NP in the circuit description shown in Fig. 2.

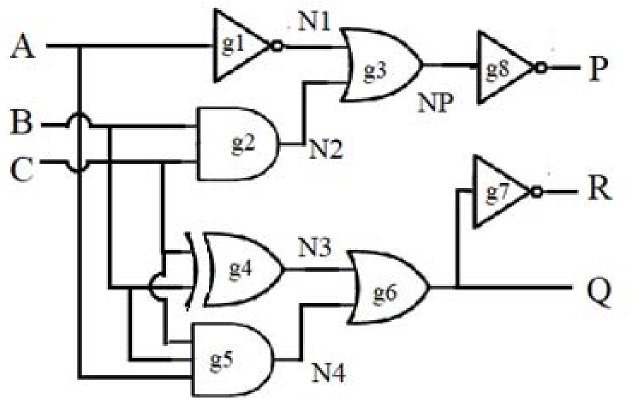

Fig. 1. Circuit with 3 inputs and 3 outputs having 8 gates

Due to lack of compiler expertise among students, the author provided a primitive compiler to store the internal in-memory representation of the logic circuit in a two-dimensional array (table) of integers as shown in Fig. 3, which represents the circuit from Fig. 1 in the program memory. The circuit element names were stored in a separate array of character strings. The column *Row Number* in Fig. 3 is only for easy reference in here, and not stored in the table. Each row of the table corresponds to an input, an output, a logic element, or an internal wire. The ELE_TYPE column holds the type of the element a table row represents. Table 1 shows the relationship between the logic elements supported and how each element type is represented in the circuit data base. Index number of a row of the table represents the ID number of the logic element corresponds to that row. For example, the gate g1 which drives wire N1 (at row 8) occupies row number 7. First set of the table rows are occupied by inputs followed by outputs, except for very 1st row (contains all 0's) which is not supposed to be accessed at all since the row with index 0 is not holding a valid element. That row is used as a trap for potential software errors in the simulation engine. Since the table needs a fixed number of columns, each logic element is allowed to have up to 3 inputs (IN1_ID, IN2_ID, and IN3_ID). This limit can easily be increased to allow more input signals for logic elements.

```
module circuit1 ;
    input A B C ;
    output P Q  R ;
    NOT g1  N1 A ;
    AND  g2  N2  B  C  ;
    OR g3 NP N1  N2  ;
    XOR g4 N3 B  C ;
    AND  g5 N4 C B A ;
    OR g6 Q N4 N3  ;
    NOT g7 R Q ;
    NOT g8 P NP ;
endmodule
```

Fig. 2. Simple grammar to describe the circuit in Fig. 1

Row Number	*ELE_TYPE*	*OUT_ID*	*IN1_ID*	*IN2_ID*	*IN3_ID*	*ELE_OUT_VAL*	*ELEMENT NAME*
0	0	0	0	0	0	0	Null
1	5	1	1	0	0	3	A
2	5	2	2	0	0	3	B
3	5	3	3	0	0	3	C
4	6	4	19	0	0	3	P
5	6	5	17	0	0	3	Q
6	6	6	18	0	0	3	R
7	3	8	1	0	0	3	g1
8	11	8	7	0	0	3	N1
9	1	10	2	3	0	3	g2
10	11	10	9	0	0	3	N2
11	2	12	8	10	0	3	g3
12	11	12	11	0	0	3	NP
13	4	14	2	3	0	3	g4
14	11	14	13	0	0	3	N3
15	1	16	3	2	1	3	g5
16	11	16	15	0	0	3	N4
17	2	5	16	14	0	3	g6
18	3	6	5	0	0	3	g7
19	3	4	12	0	0	3	g8

Fig. 3. In memory representation of the circuit in Fig. 1.

When an element does not require or use all 3 inputs, a 0 is placed for its unused input columns (to indicate nonexistent input wires). E.g. Row 7 (g1) has only 1 input (A), so both IN2_ID and IN3_ID are 0, whereas g2 (row 9) has 2 inputs and hence only IN3_ID is 0 at row 9. Column 2 holds ID of the wire that an element drives (output of that element). E.g. Gate g2 (row 9) drives N2 (row 10), and hence OUT_ID of g2 has 10 (N2). When an element is evaluated (or applied directly in case of primary inputs), its output value is updated on

ELE_OUT_VAL (6th) column, and at the same time OUT_ID entry is used to update the wire (circuit node) driven by that element. E.g. To evaluate g6, the simulator takes 6th column values from rows 16 and 14, and updates 6th column entries at rows 17 and 5. Nonexistence input to an element is indicated by 0 for that input ID (e.g. gate g3 at IN3_ID column has 0) so that the simulator would not try to find its value. This mechanism eliminated the need to keep an input count for each element. Another example: Evaluation of g4 takes 6th column values from rows 2 and 3, and updates 6th column at rows 13 and 14.

The overall algorithm is described next. Read an input vector, and deposit its bit values to ELE_OUT_VAL column entries of the inputs in their order of appearance; Now from 1st primary output element to the last logic element in the table (Fig. 3), evaluate each element (row), and update its ELE_OUT_VAL entry only if its current value is different from the new value. If there are no more updates (i.e. new value is same as old value) for an entire pass of 1st primary output element to the last element (Fig. 3), then the simulation has stabilized. If a large number of passes (iterations) has been performed compared to the circuit size, then oscillations are present and the next updated value becomes X which would lead to the stabilization of value propagation. After circuit values have become stable, input and output signal values are written as results. These steps are repeated until the stimulus vector file is completely utilized, and at that point the simulation successfully ends.

TABLE 1. NUMERICAL CODE VALUES FOR ELEMENT TYPES

Code value	Logic element
1	AND
2	OR
3	NOT
4	XOR
5	input
6	output
7	Logic 0
8	Logic 1
9	Logic X
10	Logic Z
11	wire
12	NOR
13	NAND
14	XNOR

Restricting the simulation algebra to logic 0 and logic 1 would have made the simulator run faster, but it would not allow handing of feedback paths, and three state gates which require a floating (high impedance) logic value. Therefore, all four (0,1,X,Z) logic values (levels) were utilized for logic evaluations. Most students used the table lookup method for evaluating logic elements. Some used if-else conditional statements, but no performance comparisons were made between the two implementations. As an example, Table 2 illustrates a lookup table for *and* operation represented as a two-dimensional array. Two input values to *and* gate are used to lookup their *and* results. That partial result is then used with another input value for multiple (3 or more) input *and* gates. First row and first column, which map to 0 indices in a C-code data table, are reserved as a way of checking for coding errors. As shown in column 1 of Table 2, the logic values 0, 1, X, Z are encoded (i.e. represented in the C code) as 0, 1, 2, 3 integer values, respectively.

TABLE 2. LOGICAL *AND* OPERATION LOOKUP TABLE

Unused	X	0	X	X	X
0 (=0)	0	0	0	0	0
1 (=1)	X	0	1	X	X
X (=2)	X	0	X	X	X
Z (=3)	X	0	X	X	X

Fig. 4 shows the simulator output on the screen for the example circuit in Fig. 1. The columns show values of inputs (A, B, and C in Fig. 1) in their order of *input* declaration in the circuit description (Fig. 1) followed by outputs in the order of *output* declarations. Each row of displayed values corresponds to a single input vector (from file *vec1.txt*). The input *vec1.txt* stimulus file contained all 8 possible vectors (0's and 1's) for 3 bit inputs as evident from Fig. 4. Application of X as a don't care (instead of unknown) on input was not applied which would otherwise have given all possible 27 vectors. One can verify that the in memory data base is correct by carrying out the value propagation on (Fig. 1) gates and also on the compiled circuit in Fig. 3 itself for each stimulus vector individually. Students were given several circuits and test vector files along with results from the author's solution (simulator) so they would know the expected results. They submitted their program for grading by the author on other test cases. The simulation results for each test circuit were verified for accuracy using the commercial HDL simulator by both students and the author.

```
C:\Logicsim\Lsim  circuit1.v   vec1.txt
0 0 0 0 0 1
0 0 1 0 1 0
0 1 0 0 1 0
0 1 1 0 0 1
1 0 1 1 1 0
1 1 0 1 1 0
1 1 1 0 1 0
```

Fig. 4. Simulation output for circuit in Fig. 1.

IV. Student Feedback

To achieve maximum learning outcomes, this assignment was formulated as an individual student project. Students were surveyed after they received the project grades allowing them sufficient time for self-reflection on the project and the effort they put in. The survey results gathered from 13 Computer Engineering majors are summarized qualitatively in Table 3. The responses were collected based on the typical rubrics of 1 to 5 (1-strongly disagree, 2-disagree, 3-neutral, 4-agree, 5-strongly agree). Column 2 lists what each question was trying to assess based on the perception and knowledge students acquired. The 3rd column shows the triplet of the minimum, average, and maximum (as min/ave/max) rubric scores of the student responses. The last column provides the standard deviation (SD) of the scores for each question.

TABLE 3. STUDENT SURVEY RESULTS

	Survey Question	Min/Ave/ Max	SD
1	I completed the Logic Simulator project.	4.0 / 4.2 / 5.0	0.4
2	I think my simulator worked based on my testing.	3.0 / 3.9 / 5.0	0.5
3	Logic simulator assignment helped me to understand and learn (more) about modular programming	3.0 / 3.8 / 5.0	0.7
4	I learned (more) about software simulators from the project	3.0 / 4.2 / 5.0	0.7
5	This project made me understand how the HDL simulator works on Verilog circuits	2.0 / 3.7 / 5.0	0.9
6	I understood the data base structure of a given circuit in the project	4.0 / 4.2 / 5.0	0.4
7	I think I can enhance it to use linked lists (not a table) to hold the circuit data base.	3.0 / 4.2 / 5.0	0.8
8	Now I can appreciate the efforts gone into making the HDL simulator (although it is free to students)	4.0 / 4.5 / 5.0	0.5

Question 1 gauges student's perception of own achievement level, which is high as a group having a low SD value. However, the minimum score of the 2nd question shows that some of them did not complete the work correctly to pass all the test cases available to them. Its low 0.5 SD value also indicates a clustering closer to the average (3.9), and only a few had the highest confidence on their work. It should be noted here that their work was assessed on a different set of test cases not available to students. Question 3 is on learning software engineering from this project since it incorporates several software modules (including source code and compiled object code files) rather than working on a single source code file. Its score shows a wider variation again with a higher SD value.

Questions 4 and 5 are about the basic simulation techniques used in the project. The survey shows that they understood the simulation techniques of this project while not being comfortable on how a commercial tool works, which is understandable. Question 5 shows the highest variation of answers and the highest SD. The narrow triplet score range and the positive response to question 6 indicate that students achieved the main thrust of the assignment since that is essential to complete the project. Question 7 reveals that students have a high confidence level in complex programming on average, but with a few at the low end leading to a high SD value. Question 8 is an explicit attempt to get them to appreciate the commercially available circuit simulator they have used in this and a prior course, and to think about the effort that must have gone into developing it over many years by many developers.

For comparison of students' performance, Table 4 provides the statistics on the project grade and the final course grade. Both ere normalized by the highest rubric value of 5. Table 4 shows a wide variation for the project grade and even larger gap for the course grade. Both show the typical undergraduate academic performance of having low and high achieving students in any discipline. The standard deviation for the project grade is in the vicinity of the highest standard deviation for the student survey which may indicate somewhat reliable survey results.

TABLE 4. STUDENT GRADES (NORMALIZED)

	Category	Min/Ave/Max	SD
1	Project grade	2.6 / 4.3 / 5.0	0.9
2	Course grade	1.6 / 3.8 / 5.0	1.3

V. Conclusion

The decision to introduce this complex project was done with ample caution and care so as not to overwhelm students who are also working on senior capstone projects. Student learning objectives were to show how a circuit description is processed (compiled) to create an in-memory data base, and then to show how to navigate the circuit for logic value propagation through circuit elements. Based on the survey responses and the grades students earned for the project, it can be concluded that the way the project was formulated and assignment of student work were a success. This project would certainly benefit the students by increasing their self-confidence on facing new engineering problems in future. The author as the instructor has to put in a considerable effort to formulate and support this project which is much more than writing a simple C program. However, with the level of benefit it brings to students, the author plans to continue similar projects in future offerings of the course. By the end of the course, a few students were suggesting another class project to write yet another software program, using the same circuit database, which would generate input stimulus to apply for simulation of simple logic circuits.

REFERENCES

[1] S. Fincher, "What are We Doing When We Teach Programming?" Proceedings of the 29th Annual Conference on Frontiers in Education ‘99, San Juan, Puerto Rico, November 10-13, 1999, pp 12a4-1- 12a4-5.

[2] S. Ludi, J. Collofello, "An analysis of The Gap Between the Knowledge and Skills Learned in Academic Software Engineering Course Projects and Those Required in Real Projects," Proceedings of the Conference on Frontiers in Education, Reno, NV, USA, October 10-13, 2001, pp T2D08-T2D11.

[3] PSpice™ documentation, 2017. [Online] Available at: http://www.orcad.com/products/orcad-pspice-designer/review

[4] Dev-C++ software development environment, 2017. [Online] Available at: https://sourceforge.net/projects/orwelldevcpp.

[5] Verilog Hardware Description Language, IEEE Standard 1364-2005, 2005.

[6] Modelsim™ documentation, 2017. [Online] Available at: https://www.mentor.com/company/higher_ed/modelsim-student-edition.

[7] VHDL Hardware Description Language, IEEE Standard 1076-2008, 2008.

[8] E. Frontoni, A. Mancini, F. Caponetti, and P. Zingaretti, "A framework for simulations and tests of mobile robotics tasks," Proceedings of the 14th Mediterranean Conference on Control and Automation, MED '06, Ancona, Italy, June 28-30, 200