Creating an Arithmetic and Logic Unit (ALU) with ModelSim

Catherine Kuntoro
San Jose State University
catherine.kuntoro@sjsu.edu

***Abstract*—To showcase the creation of a 32 bit Arithmetic and Logic Unit (ALU) through the usage of ModelSim simulation tool**

I. INTRODUCTION

Through the use of a ModelSim simulation tool, create a fully functional ALU that is able to calculate the required arithmetic and logic functions for the 'CS147DV' instruction set. The objectives of this project are as listed below:
1) To download and install a digital simulation tool named ModelSim before setting it up
2) To implement a ALU module using Verilog HDL and its requirement
3) To rigorously verify the procedures by using test bench code to examine the accuracy of the ALU by using HDL
4) To simulate and observe firsthand the output signal waveforms of ALU using ALU test bench code

This report includes an explanation of how to properly download and set up the ModelSim simulator, in addition to illustrating and detailing the implementation of the ALU.

II. INSTALLATION AND SETUP

1. Installation of the ModelSim SimulatorThe ModelSim simulator can be downloaded from this link: http://www.mentor.com/company/higher_ed/modelsim-student-edition . Then, click on the "Download Student Edition" button
2. Start the installation of the files into your computer drive by opening the installation file, running it, and completing the installation steps required
3. A form will appear in the browser after a completed installation of the simulator. The form requires for the student's name, address, phone number, email, university name, and other information to be filled in.
4. After form is submitted, an email will be sent from Modelsim. A licence with the name 'student_license.dat' will be attached to the email
5. Save 'student_license.dat' to the top level installation directory for ModelSim PE Student (for example, c:/modeltech_pe_edu). This directory must contain the sub-directory 'win32pe_edu'. Do not edit the license file 'student_license.dat' in order to ensure that the license will work properly
6. ModelSim PE Student Edition would be successfully and smoothly runned now

III. STEPS TO SIMULATION PROJECT CREATION

This section entails the procedures of simulating the Verilog code for ALU and its testing with the test bench code using ModelSim tool.

These are the steps to simulate the ALU's Verilog code

A. Download the given starter files in Canvas

In this link: https://sjsu.instructure.com/courses/1374451/assignments/5365121?module_item_id=10741710, download the prj_01.zip and extract the contents of the .zip file, which includes:

1. prj_definition.v
   This file has the specific bits and definitions that would be utilized to create the ALU module
2. prj_01_tb.v
   This is the test bench file that would be utilized for testing purposes to test the functions of ALU with specific input values. Additionally, to implement the ALU, complete the source code in this file
3. alu.v
   This file has the module code for the ALU. To implement the ALU, also complete the source code in this file

B. Creating a New Project and File Setups

Although using the folder that has all of the extracted files would be acceptable, it is possible to create a new project file. Go to File, New, and select Project. A pop up shown in Figure 1 should appear.

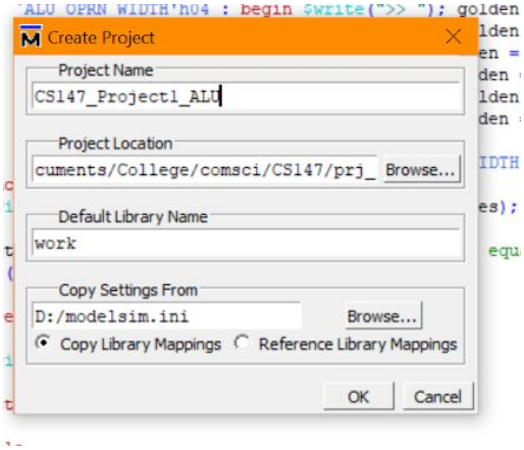

Figure 1: Creating a Project Window

Name the project with an appropriate name such as CS147_Project1_ALU, then click OK. Afterwards, click the option that says “Add Existing Files” in the top right hand corner as Figure 2 is showing below.

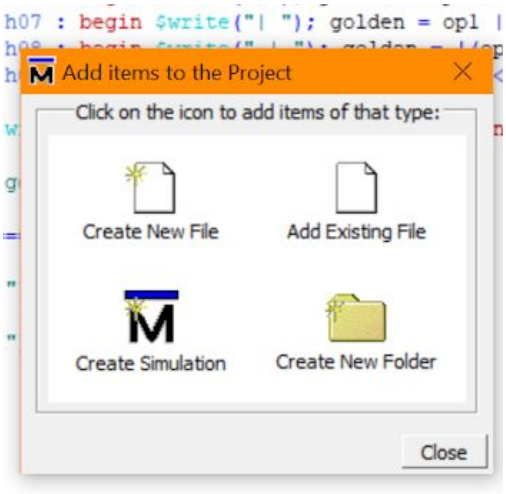

Figure 2: Adding Existing Files

Select the prj_definition.v, prj_01_tb.v, and alu.v files in the folder that contains those files, as Figure 3 is showing below:

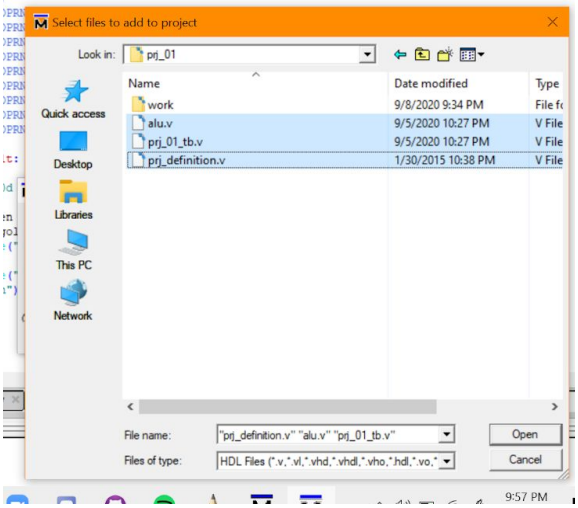

Figure 3: Selecting Existing Files

Once all the necessary files have been added into the project, under the ‘Project’ tab on the left hand side, select the prj_definition.v, prj_01_tb.v, and alu.v files. Then, click on the “Compile” tab, and click on “Compile All”, as shown below in Figure 4.

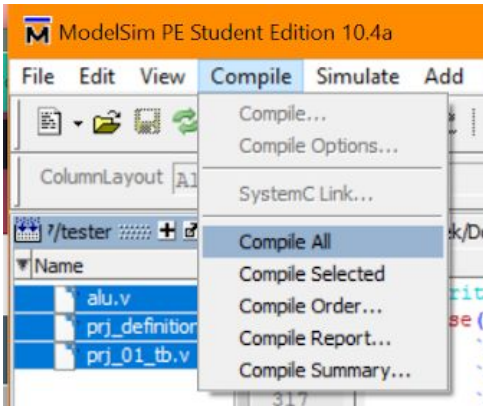

Figure 4: Compiling the Projects

A successful compilation will have a green checkmark under the “Status” column, as shown below in Figure 5.

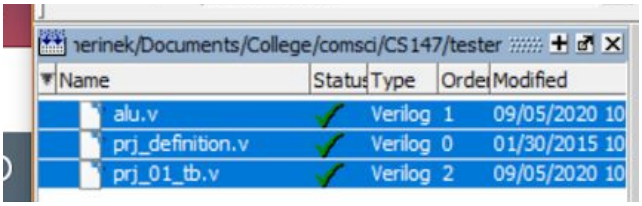

Figure 5: Checking Success of Compilation

C. Checking for Output Waveforms

On the bottom left, there is a tab called “Library”. Click on that tab, and then under work, click on prj_01_tb.v. The location of the prj_01_tb is shown in Figure 6 below.

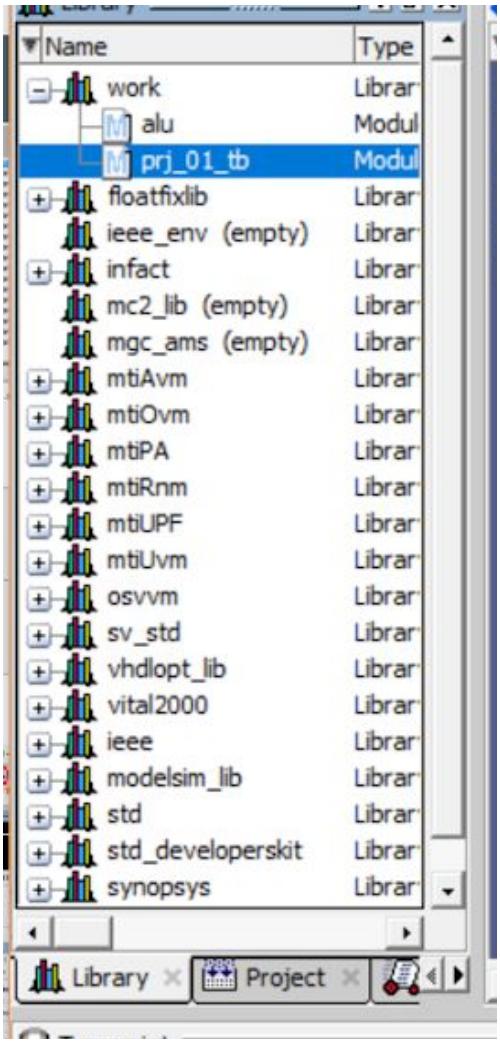

Figure 6: Clicking prj_01_tb in the Library Tab

Afterwards, ModelSim should change to Simulation layout as shown below in Figure 7. Then, select all of the options that appear on the object window (the window on top of the purple-colored background), and click ‘Add Wave’.

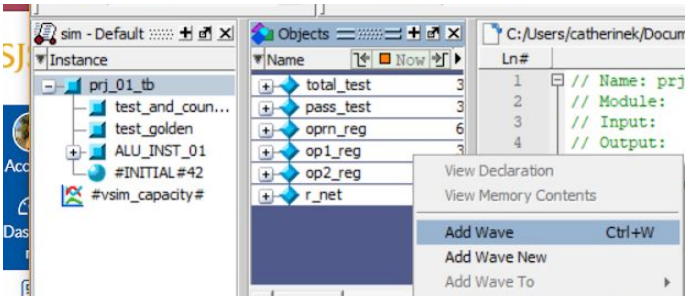

Figure 7: Adding a Wave to the Objects

Then, a new tab with a black background will appear. This is the tab called “Wave” that allows for the output waveforms to be seen. The values in the wave will usually be displayed in the hex number system, though it is possible to change it by selecting all of the objects and clicking on “Radix”.

To execute the program and see all of the output waveforms, click on the ‘Run All’ button as shown in Figure 8 below.

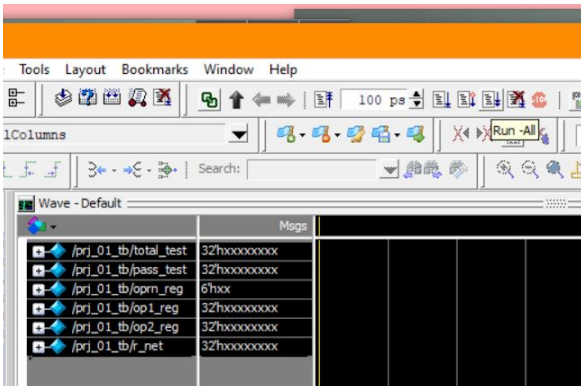

Figure 7: Window for Waveform and Run All Button Above

IV. REQUIREMENTS OF AN ALU

ALU, or Arithmetic and Logic Unit, is a fundamental digital circuit for the central processing unit (CPU) that uses boolean logic operations such as OR, AND, NOR, NOT, XOR in order to compute arithmetic operations such as addition, multiplication, subtraction, and division, and other operations such as register shifting left or right and other bitwise operations.

In order for the ALU to decode and understand what operations need to be done, the Control Unit register (CU) will transfer the opcodes necessary to the ALU. The opcodes act as a description for how should the ALU use and

manipulate the data, and the opcodes are given into the ALU through input registers.

All arithmetic and logical operations will be broken down by the ALU into multiple two operand operations, since the ALU usually takes in only two operands, one operator, and results in one output. As an example, the mathematical expression:
R = (A - B*C) is broken down into:

P1 = B*C
R = A - P1

Figure 8 shows a diagram of the ALU:

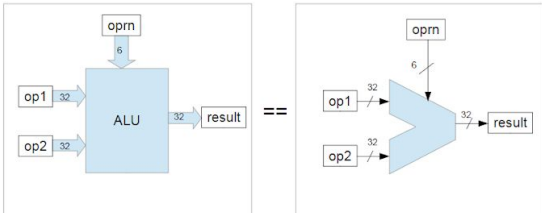

Figure 8: Interface Diagram for ALU

The parts of an ALU include:

a) op1 and op2, which are the operands that accept the inputs to the ALU. The operands has 32 bits each in this ALU diagram
b) The result will give the output. The result are encoded in 32 bits in this ALU diagram
c) oprn is the operation code, or opcode, which is an instruction given from the Control Unit that tells the ALU what to do with the two operands. 6 bits encode the operation code for this ALU diagram

Therefore, the ALU receives input data to process from the two operands, and also an operation code that determines what operation should be done on the operands, and finally the ALU will give the result as an output data.

IV. DESIGNING AND IMPLEMENTING THE ALU

A Hardware Descripting Language, such as Verilog, allows the userts to describe the operation and data flows without creating logic gates and other circuit implementations. This allows the user to write a hardware program, like an ALU, with a language that appears to be like a high level language in first glance. Verilog also needs a simulator such as Modelism to perform the circuit implementations that an ALU needs.

A. Design Strategy

Think of the ALU like it is a module, and take note of which inputs and outputs are used for the ALU. Below is the code for a basic module for the ALU, where “op1”, “ op2”, “oprn” are the input ports, and the output port is represented as “result”. The input ports opr1 and op2 both have 32 bits, the input port oprn has 6 bits, while the output port result has 32 bits.

```
module alu(result, op1, op2, oprn)
        input[31:0] op1,op2;
        input[5:0] oprn;
        Output[31:0] result;
                //set of operations appear
endmodule
```

B. Required Operations to be Done by the ALU

After declaring the types of ports, declare the set of operations. After the declarations of each operation of the ALU, the module will end and leave the output wire “result”, exit the ALU module, and then the output waveforms will be seen.

The required operations that the ALU needs to perform in this project are as listed below:

1. Addition
2. Subtraction
3. Multiplication
4. Shift Right
5. Shift Left
6. Bitwise AND
7. Bitwise OR
8. Bitwise NOR
9. Set less than

To use a specific operation, an opcode will be given as an input to the ALU. Below is an implementation of the ALU in Verilog code.

```
// Whenever op1, op2 or oprn changes do something
always @ (op1 or op2 or oprn)
begin
    case (oprn)

        `ALU_OPRN_WIDTH'h01 : result = op1 + op2; // addition
        `ALU_OPRN_WIDTH'h02 : result = op1 - op2; // subtraction
        `ALU_OPRN_WIDTH'h03 : result = op1 * op2; // multiplication
        `ALU_OPRN_WIDTH'h04 : result = op1 >> op2; // shift right
        `ALU_OPRN_WIDTH'h05 : result = op1 << op2; // shift left
        `ALU_OPRN_WIDTH'h06 : result = op1 & op2; // Bitwise and
        `ALU_OPRN_WIDTH'h07 : result = op1 | op2; // Bitwise or
        `ALU_OPRN_WIDTH'h08 : result = !(op1 | op2); // Bitwise nor
        `ALU_OPRN_WIDTH'h09 : result = op1 < op2; // set less than

        default: result = `DATA_WIDTH'hxxxxxxxx;

    endcase
end
```

Figure 9: Implementation of the ALUin Verilog Language

`ALU_OPRN_WIDTH'h01 indicates that the ALU opcode is 6 bits. h01 means that the hex code one will encode a specific arithmetic operation, in this case the addition operation. Therefore, the ALU will be performing a specific operation that is based on what value the oprn register holds. If the oprn's value or the op1 and op2's values are unknown, the ALU will give an 'X' as its output and the operation will be a failure.

V. STRATEGIES AND IMPLEMENTATION FOR TEST CASES

After the Verilog implementation code of the ALU is completed, it is necessary to check the module in order to verify the validity and accuracy of the arithmetic operation done, and whether or not the ALU meets the user's requirements.

The ALU operation could be verified and tested through the use of the 'Test Bench' code. Through the Test Bench code, also written in the Verilog language, carefully selected and chosen numbers are defined as the inputs in order to test the validity of the ALU's operations by comparing what output the ALU actually gave out, versus the expected output. Therefore, by checking the expected versus actual outputs, the user could validate the design and implementation of the ALU.

First, the test bench code will provide to the ALU the two operands, op1 and op2, and also one opcode, oprn. Afterwards, the ALU will give an output, which is the result, back into the test bench code. It is possible then to see the input and output waveforms in order to check the validity of the ALU's operations. However, if any of the operands or opcode is undefined or unknown, then the register with the results will give 'x' as an output, or specifically 32'dx, which is represented as a red line. Figure 10 below shows an output of 'x', or 32'dx.

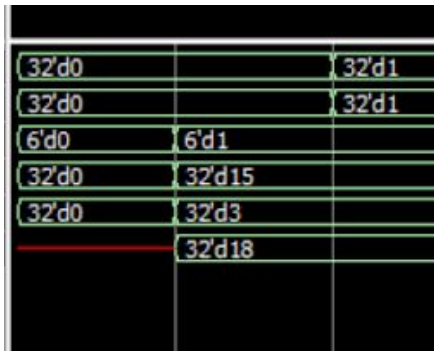

Figure 10: Unknown/Undefined Value

Going from the topmost wave to the lowest wave, each wave represents the total test, the number of tests passed, the operand register, the first operand, the second operand, and the result register respectively.
If a wave does not seem to have a value onto it, it seems that the value of the current wave is the same as the value of the last wave

A.Addition

op1 = 15
op2 = 3
oprn = 1
result = 18

Figure 11: Wave Output

op1 = 15
op2 = 5
oprn = 1
result = 20

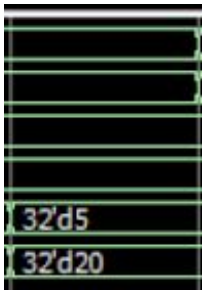

Figure 12: Wave Output

op1 = -15
op2 = 5
oprn = 1
result = -10

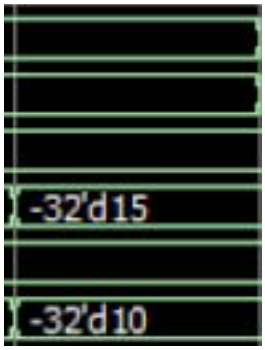

Figure 13: Wave Output

op1 = 15
op2 = -5
oprn = 1
result = 10

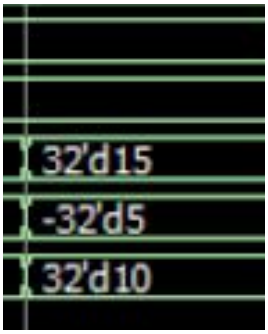

Figure 14: Wave Output

op1 = -15
op2 = -5
oprn = 1
result = -20

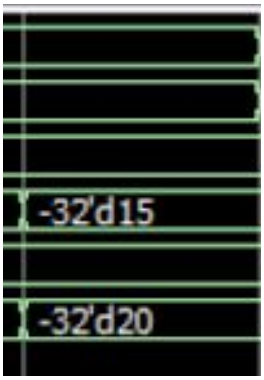

Figure 15: Wave Output

B. Subtraction

op1 = 15
op2 = 5
oprn = 2
result = 10

Figure 16: Wave Output

op1 = -15
op2 = 5
oprn = 2
result = -20

Figure 17: Wave Output

op1 = 15
op2 = -5
oprn = 2
result = 20

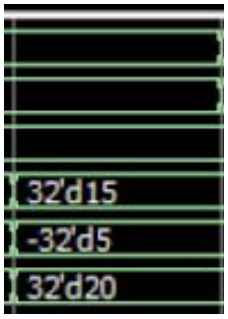

Figure 18: Wave Output

op1 =-15
op2 = -5
oprn = 2
result = -10

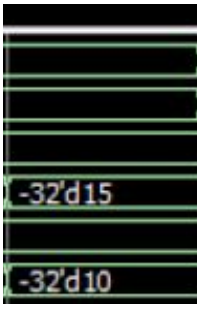

Figure 19: Wave Output

C. Multiplication

op1 = 15
op2 = 5
oprn = 3
result = 75

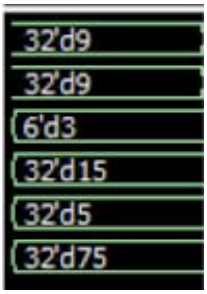

Figure 20: Wave Output

op1 = -15
op2 = 5
oprn = 3
result = -75

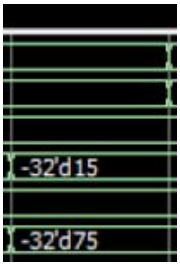

Figure 21: Wave Output

op1 = 15
op2 = -5
oprn = 3
result = -75

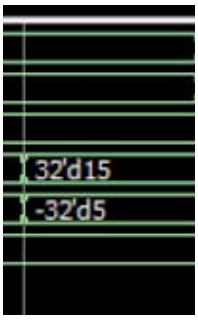

Figure 22: Wave Output

op1 = -15
op2 = -5
oprn = 3
result = 75

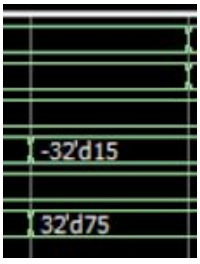

Figure 23: Wave Output

D. Shift Right

op1 = 15
op2 = 0
oprn = 4
result = 15

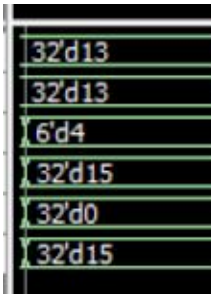

Figure 24: Wave Output

op1 = 15
op2 = 1
oprn = 4
result = 7

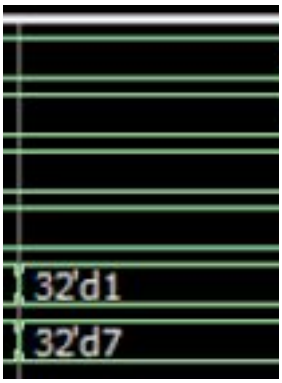

Figure 25: Wave Output

op1 = 15
op2 = 2
oprn = 4
result = 3

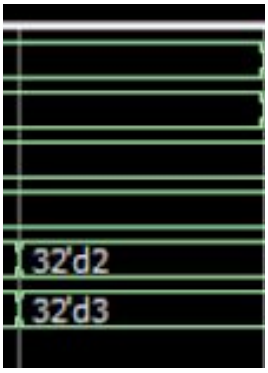

Figure 26: Wave Output

op1 = 15
op2 = 3
oprn = 4
result = 1

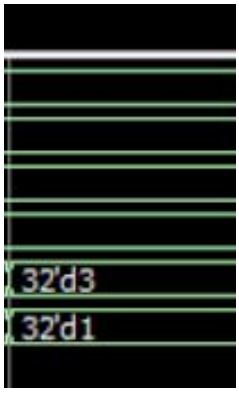

Figure 27: Wave Output

op1 = 15
op2 = 4
oprn = 4
result = 0

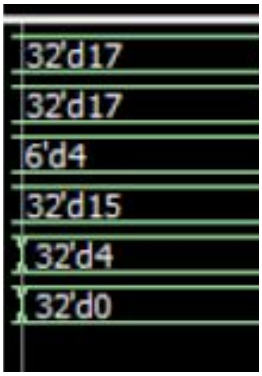

Figure 28: Wave Output

E. Shift Left

op1 = 15
op2 = 0
oprn = 5
result = 15

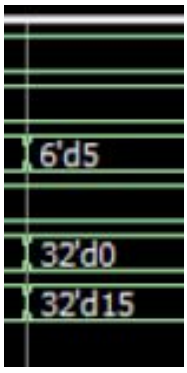

Figure 29: Wave Output

op1 = 15

op2 = 1
oprn = 5
result = 30

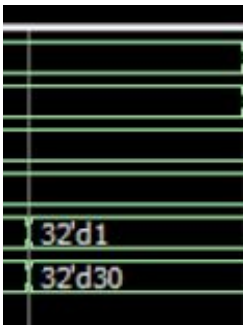

Figure 30: Wave Output

op1 = 15
op2 = 2
oprn = 5
result = 60

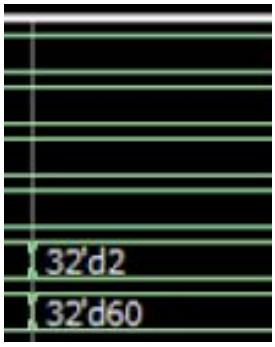

Figure 31: Wave Output

op1 = 15
op2 = 3
oprn = 5
result = 120

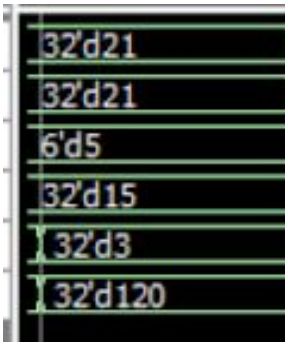

Figure 32: Wave Output

op1 = 15
op2 = 4
oprn = 5
result = 240

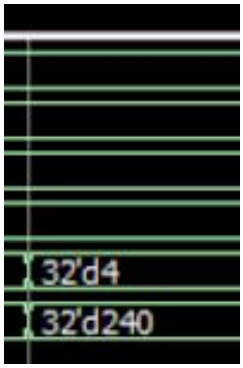

Figure 33: Wave Output

F. Bitwise and

op1 = 0
op2 = 1
oprn = 6
result = 0

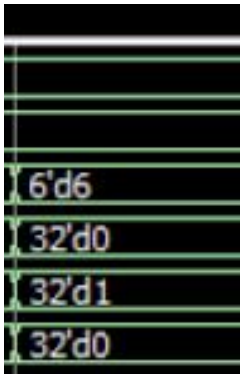

Figure 34: Wave Output

op1 = 0
op2 = 0
oprn = 6
result = 0

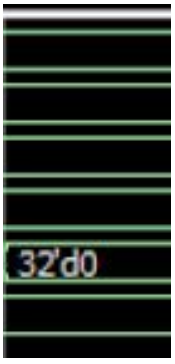

Figure 35: Wave Output

op1 = 1
op2 = 1
oprn = 6
result = 1

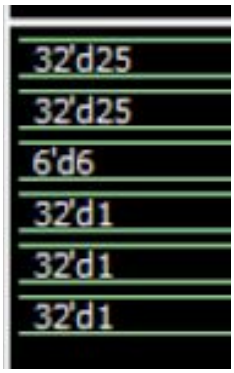

Figure 36: Wave Output

op1 = 15
op2 = 8
oprn = 6
result = 8

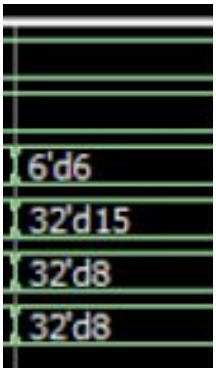

Figure 37: Wave Output

G. Bitwise or

op1 = 0
op2 = 1
oprn = 7
result = 1

Figure 38: Wave Output

op1 = 0
op2 = 0
oprn = 7
result = 0

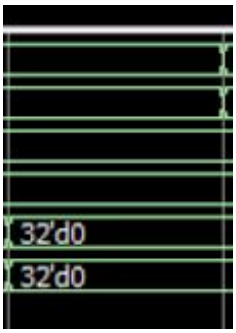

Figure 39: Wave Output

op1 = 1
op2 = 1
oprn = 7
result = 1

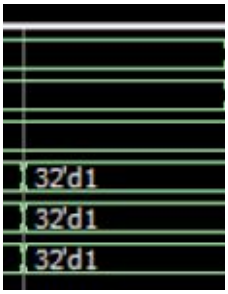

Figure 40: Wave Output

op1 = 15
op2 = 8
oprn = 7
result =15

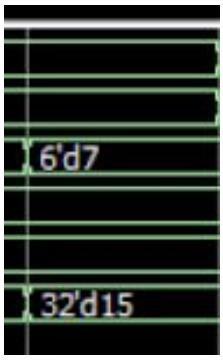

Figure 41: Wave Output

H. Bitwise nor

op1 = 0

op2 = 1
oprn = 8
result = 0

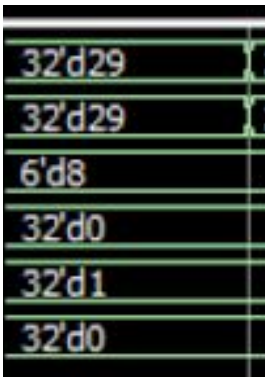

Figure 42: Wave Output

op1 = 0
op2 = 0
oprn = 8
result = 1

Figure 43: Wave Output

op1 = 1
op2 = 1
oprn = 8
result = 0

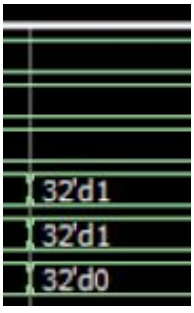

Figure 44: Wave Output

op1 = 15
op2 = 8
oprn = 8
result = 0

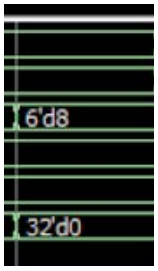

Figure 45: Wave Output

I. Set less than

op1 = 0
op2 = 1
oprn = 9
result = 1

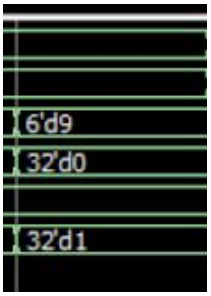

Figure 46: Wave Output

op1 = 0
op2 = 0
oprn = 9
result = 0

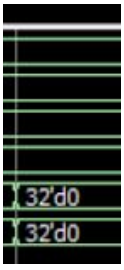

Figure 47: Wave Output

op1 = 1
op2 = 1
oprn = 9
result = 0

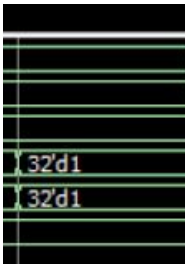

Figure 48: Wave Output

op1 = 15
op2 = 8
oprn = 6
result = 9

Figure 49: Wave Output

Transcript Window

After the simulation is completed, ModelSim will show the outputs in the Transcript window. The user can see how many tests are completed, and whether or not any test failed. The transcript window also provides the user with the lines that have a compiler error, if any. A Transcript window is shown below.

There are a total 39 of test operations, therefore to determine if the ALU is valid, all 35 test operations must be shown as a success, which can be seen in the figure below.

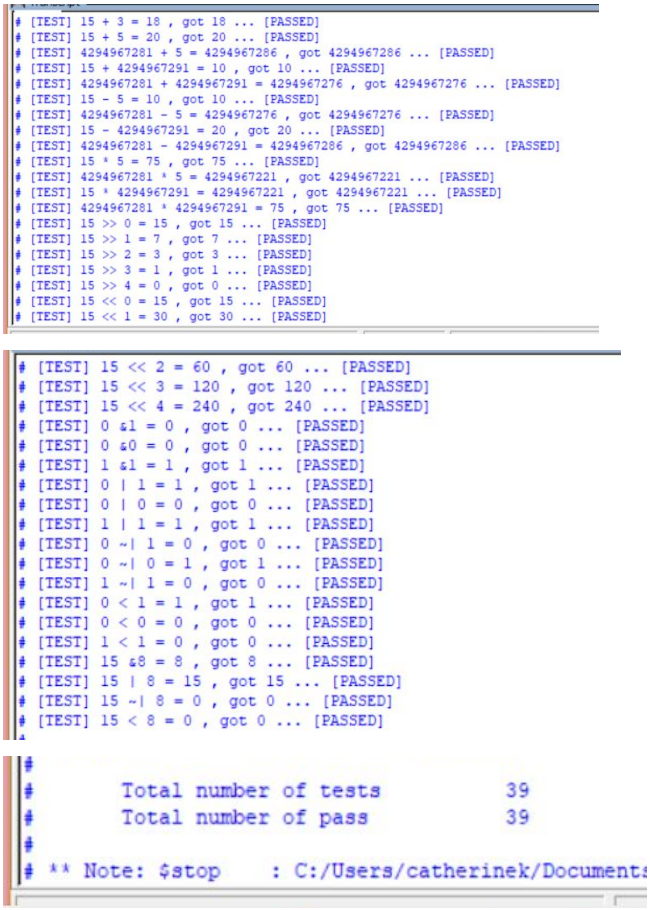

Figure 50: Transcript Window

VI. CONCLUSION

Creating and completing the ALU with Verilog language through the utilization of the ModelSim simulator allows me to gain a deeper understanding of how to use ModelSim Simulator, and how to use the Verilog language as well. This project allows me to make modules and test cases, which would surely be helpful in future projects, in addition to letting me know how to create waves so I could examine the output waves to check for the validity of the inputs and outputs.

VII. REFERENCES

[1] K. Patra. CS 147. Class Lecture, Topic: “Lect 01: Computer System, Instruction Set, ALU.” San Jose State University, San Jose, CA, September 10, 2020

[2] K. Patra. CS 147. Class Lecture, Topic: “Lab 01: Setup for CS147 Lab.” San Jose State University, San Jose, CA, September 10, 2020

[3] K. Patra. CS 147. Class Lecture, Topic: “Lab 02: Hierarchical Modeling.” San Jose State University, San Jose, CA, September 10, 2020

[4] K. Patra. CS 147. Class Lecture, Topic: “Lab 03: Data Flow Modeling I.” San Jose State University, San Jose, CA, September 10, 2020

[5]] K. Patra. CS 147. Class Lecture, Topic: “Lab 04: Data Flow Modeling II.” San Jose State University, San Jose, CA, September 10, 2020