Arithmatic Logic Unit (ALU)

CS147 Project 1: Setting up the ALU

Ryan Tran
College of Science: Computer Science
San José State University
San José, United States
tranryanp@gmail.com

***Abstract*—This report explains the installation, usage, and setup of ModelSim, a hardware simulation tool. Through Verilog code, an arithmetic logic unit (ALU) will be created to handle operations between operands. This ALU setup will be vital to the next CS147 project.**

I. Introduction

This project is the first assignment of CS147. The objective is to create an ALU using Verilog to calculate results for nine operations: addition, subtraction, multiplication, shift logical right, shift logical left, bitwise and, bitwise or, bitwise nor, and less than. ModelSim is a simulation tool for multiple hardware description languages, but this project will only focus on Verilog. Setting up this ALU will be necessary for the next CS147 project in which its usage will be required. This report contains the requirements, designs, and test results of such tasks.

II. Installing ModelSim

This section covers the installation process of ModelSim. Mentor Graphics, the company that created ModelSim, provides a free edition of ModelSim for students. Below are the steps to install it:

1) Go to this download link on your browser: https://www.mentor.com/company/higher_ed/modelsim-student-edition.

2) Click on "Download Student Edition" to begin downloading the installer.

3) Run the downloaded installer file and go through the installer's steps.

4) After the installation is complete, a browser will open up with a form to complete. The form has details such as name, address, email, university name, and more. Fill out this form in order to receive a license key in the provided email.

5) After the form is complete, an email will arrive with a license key attached. It is named 'student_license.dat'. Download this .dat file and place it in the top-level directory of ModelSim (defaultly located at C:/Modeltech_pe_edu_10.4a). This directory should contain the folder 'winpe_edu'.

6) Installation complete. ModelSim should be ready to use.

III. Creating a Simulation Project

This section covers how to create a simulation project for the ALU. The simulation project will allow for the Verilog code for the ALU to be tested. Following are the steps to create the simulation project:

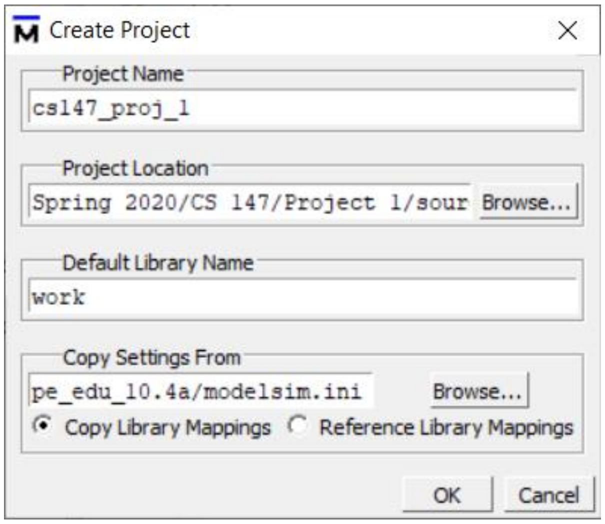
1) Download the HDL source code for Project 1 from Canvas. The file to download is 'prj_01.zip' and contains three files: 'alu.v', 'prj_01_tb.v', and 'prj_definition.v'.

- *a)* 'alu.v' is the file that will be completed with code for the ALU.
- *b)* 'prj_01_tb.v' is the testbench file that will be used to check if the ALU operations are functioning.
- *c)* 'prj_definition.v' contains specific details regarding inputted bits and definitions that will make the ALU perform as intended.

2) Extract the files into a new folder. This folder will be the root of the project.

3) Launch ModelSim.

4) Go to File -> New -> Project… The following box will appear:



Project Name: cs147_proj_1

Project Location: Spring 2020/CS 147/Project 1/sour

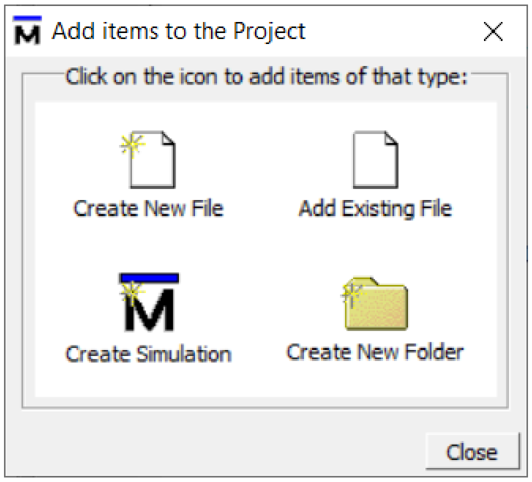
Default Library Name: work

Copy Settings From: pe_edu_10.4a/modelsim.ini

(● Copy Library Mappings) ○ Reference Library Mappings

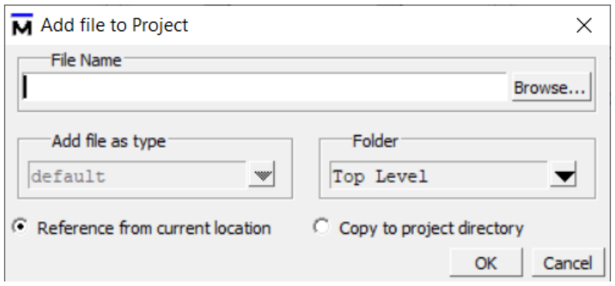
You can name it whatever you'd like. In this case, it will be 'cs147_proj_1'. Changing the project location is recommended as the original directory is difficult to access if you need to access your project files. Choosing the new folder that olds the project files is best.

5) After pressing OK, this window will appear:



Choose the "Add Existing File" option.

6) Upon choosing that open, another window will pop up:



File Name:

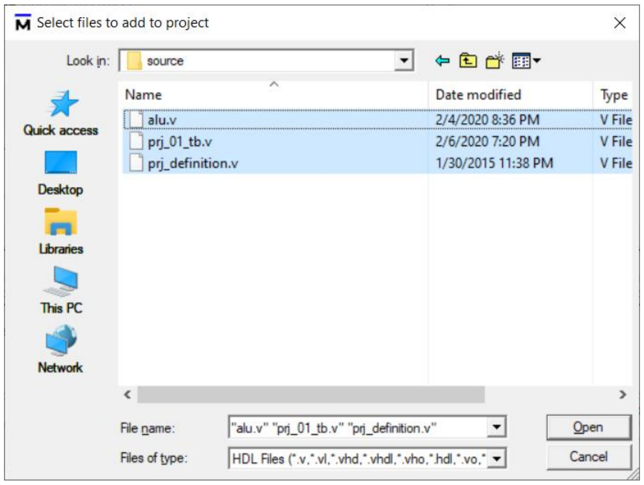
Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

Choose "Browse…".

7) A window will appear allowing you to change directories. Go to the folder with the project files and select all of them. This can be done either by selecting them all by dragging the mouse, clicking each file while holding ctrl, or clicking the top file and holding shift while pushing the down arrow.



Look in: source

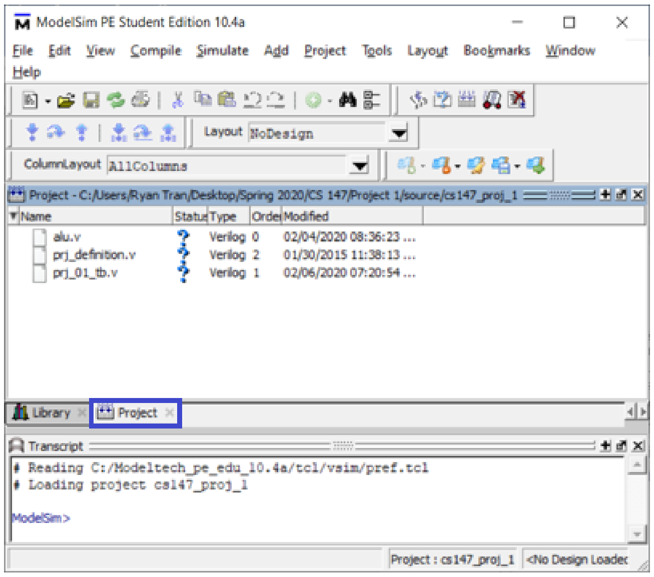
Name	Date modified	Type
alu.v	2/4/2020 8:36 PM	V File
prj_01_tb.v	2/6/2020 7:20 PM	V File
prj_definition.v	1/30/2015 11:38 PM	V File

File name: "alu.v" "prj_01_tb.v" "prj_definition.v"

Files of type: HDL Files (*.v,*.vl,*.vhd,*.vhdl,*.vho,*.hdl,*.vo,*

Select "Open" to add the files to the project.

8) The files will be added and appear as shown in the "Project" tab boxed in blue:


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Library
Project
Transcript

Layout NoDesign

ColumnLayout AllColumns

Project - C:/Users/Ryan Tran/Desktop/Spring 2020/CS 147/Project 1/source/cs147_proj_1

Name	Status	Type	Order	Modified
alu.v	?	Verilog	0	02/04/2020 08:36:23 ...
prj_definition.v	?	Verilog	2	01/30/2015 11:38:13 ...
prj_01_tb.v	?	Verilog	1	02/06/2020 07:20:54 ...

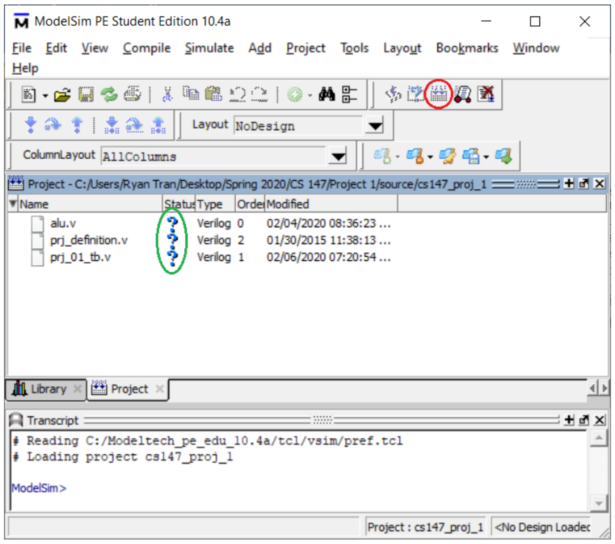
```
# Reading C:/Modeltech_pe_edu_10.4a/tcl/vsim/pref.tcl
# Loading project cs147_proj_1

ModelSim>
```

Project : cs147_proj_1 <No Design Loaded>

IV. Running the Simulation

Referring to the image above, the files have been added to the project. Now they will be simulated to test if they work. Provided are the steps to run the simulation:

1) The Status column indicates whether the compile is successful or not. When new files are added, its status is set as a question mark, as shown by the green circle, since ModelSim does not know if the file compiles or not yet.


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Library
Project
Transcript

Layout NoDesign

ColumnLayout AllColumns

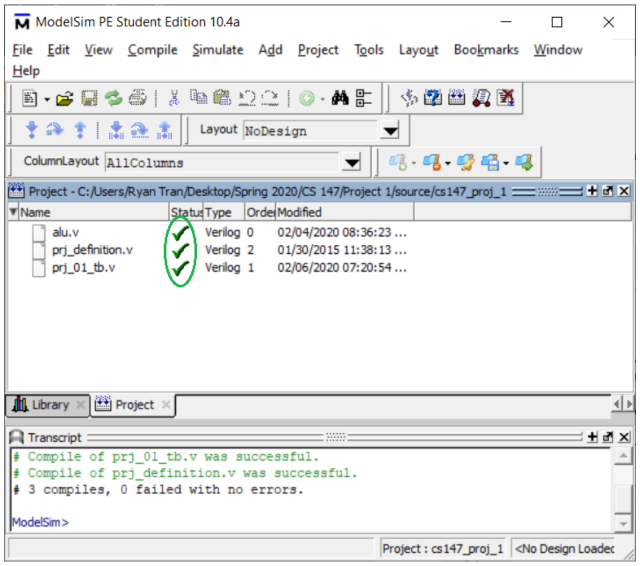
Project - C:/Users/Ryan Tran/Desktop/Spring 2020/CS 147/Project 1/source/cs147_proj_1

Name	Status	Type	Order	Modified
alu.v	?	Verilog	0	02/04/2020 08:36:23 ...
prj_definition.v	?	Verilog	2	01/30/2015 11:38:13 ...
prj_01_tb.v	?	Verilog	1	02/06/2020 07:20:54 ...

```
# Reading C:/Modeltech_pe_edu_10.4a/tcl/vsim/pref.tcl
# Loading project cs147_proj_1

ModelSim>
```

Project : cs147_proj_1 <No Design Loaded>

The icon circled in red is the "Compile All" button that needs to be selected. Alternatively, you can compile each file one by one with the icon two to the left of the red circle that is called "Compile". After clicking "Compile All", ModelSim should appear as follows:


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Project
ModelSim>

Project - C:/Users/Ryan Tran/Desktop/Spring 2020/CS 147/Project 1/source/cs147_proj_1

Name	Status	Type	Order	Modified
alu.v	✓	Verilog	0	02/04/2020 08:36:23 …
prj_definition.v	✓	Verilog	2	01/30/2015 11:38:13 …
prj_01_tb.v	✓	Verilog	1	02/06/2020 07:20:54 …

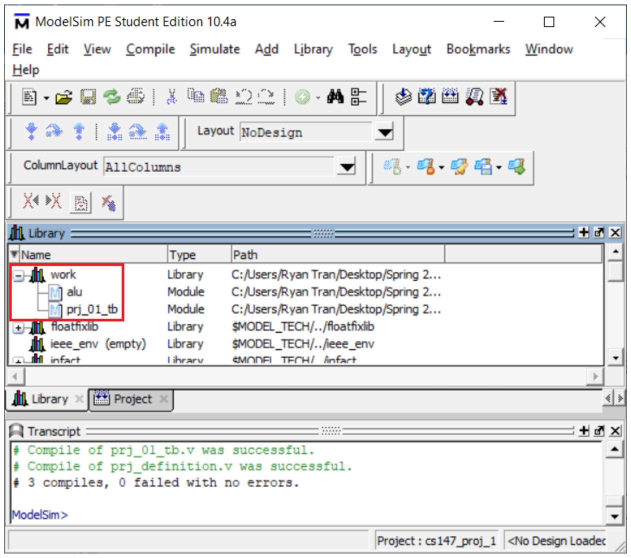
Transcript

```
# Compile of prj_01_tb.v was successful.
# Compile of prj_definition.v was successful.
# 3 compiles, 0 failed with no errors.
```

Project : cs147_proj_1 <No Design Loaded

As shown by the green circle, the Status has changed to a ✓, indicating that the file has compiled properly. Alternatively, if you see a ✗, this indicates that there was an error in that file and it was not able to compile properly. Upon correcting the error, the mark will change to a check mark upon the next compile.

2) Go to the “Library” tab boxed in blue and then expand the ‘work’ library. You will see ‘alu’ and ‘prj_01_tb’ listed within ‘work’. It is shown boxed in red:


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Project
ModelSim>

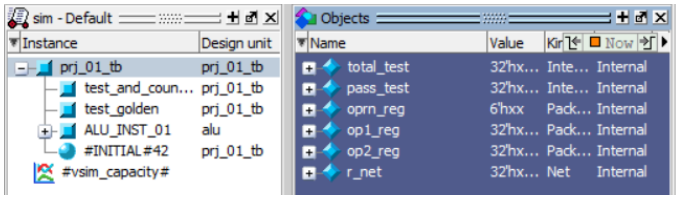
Name	Type	Path
work	Library	C:/Users/Ryan Tran/Desktop/Spring 2…
alu	Module	C:/Users/Ryan Tran/Desktop/Spring 2…
prj_01_tb	Module	C:/Users/Ryan Tran/Desktop/Spring 2…
floatfixlib	Library	$MODEL_TECH/../floatfixlib
ieee_env (empty)	Library	$MODEL_TECH/../ieee_env
infact	Library	$MODEL_TECH/../infact

Transcript

```
# Compile of prj_01_tb.v was successful.
# Compile of prj_definition.v was successful.
# 3 compiles, 0 failed with no errors.
```

Project : cs147_proj_1 <No Design Loaded

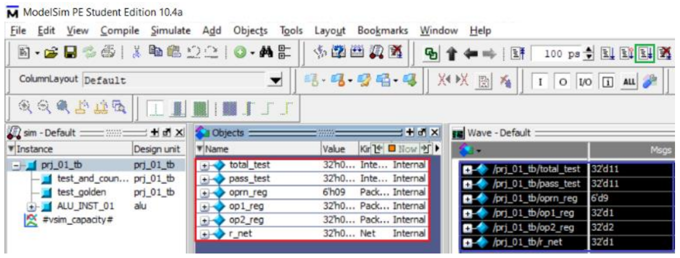
3) Double-click on ‘prj_01_tb,’ the test bench file. This will open up a ‘sim’ tab along with an ‘Objects’ tab.



Instance	Design unit
prj_01_tb	prj_01_tb
test_and_coun…	prj_01_tb
test_golden	prj_01_tb
ALU_INST_01	alu
#INITIAL#42	prj_01_tb
#vsim_capacity#	

Name	Value	Kind	Mode
total_test	32'hx…	Inte…	Internal
pass_test	32'hx…	Inte…	Internal
oprn_reg	6'hxx	Pack…	Internal
op1_reg	32'hx…	Pack…	Internal
op2_reg	32'hx…	Pack…	Internal
r_net	32'hx…	Net	Internal

4) Select all of the items in the ‘Objects’ tab as shown in the red box. Right-click on it and choose ‘Add Wave’ to add all the Objects to the ‘Wave’ tab as shown in the blue box. To now run the simulation, click the ‘Run -All’ button boxed in green in the top-right of the image.


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Objects Tools Layout Bookmarks Window Help
100 ps
ColumnLayout Default
sim - Default
Objects
Wave - Default
Msgs

Instance	Design unit
prj_01_tb	prj_01_tb
test_and_coun…	prj_01_tb
test_golden	prj_01_tb
ALU_INST_01	alu
#vsim_capacity#	

Name	Value	Kind	Mode
total_test	32'h0…	Inte…	Internal
pass_test	32'h0…	Inte…	Internal
oprn_reg	6'h09	Pack…	Internal
op1_reg	32'h0…	Pack…	Internal
op2_reg	32'h0…	Pack…	Internal
r_net	32'h0…	Net	Internal

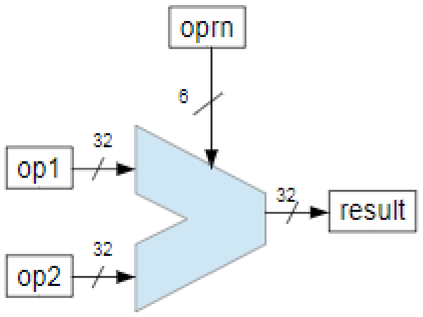
	Msgs
/prj_01_tb/total_test	32'd11
/prj_01_tb/pass_test	32'd11
/prj_01_tb/oprn_reg	6'd9
/prj_01_tb/op1_reg	32'd1
/prj_01_tb/op2_reg	32'd2
/prj_01_tb/r_net	32'd1

5) The results will appear in the ‘Wave’ window and the ‘Transcript’ window at the bottom of the interface.

V. Setting up the ALU

An arithmetic logic unit is, as the same says, responsible for carrying out arithmetic and logical operations. Aside from the obvious, it is a digital electronic circuit that performs these operations on binary values. The ALU implements arithmetic logic operations such as Addition, Subtraction, Multiplication, etc. and logical operations such as AND, OR, NOR, etc.

The ALU that will be set up is 32 bit. It takes in two operands, both of which are 32-bit, and one operation, which is 6-bit. The output result is 32-bit. Below is a diagram of said ALU for visual understanding.



As denoted by the direction of the arrows, the three inputs are the two operands and the operation. There is only one output which is the result.

A. *Requirements*

Most of the code is already as provided. Only the operations must be implemented and tested. As shown in the previous diagram, each operation has its own 6-bit respective value. The operations and their operation code are:

1) Addition (0x1)
2) Subtraction (0x2)
3) Multiplication (0x3)
4) Shift right (0x4)
5) Shift left (0x5)
6) Bitwise AND (0x6)
7) Bitwise OR (0x7)
8) Bitwise NOR (0x8)

9) Set less than (0x9)

B. ALU Implementation

Provided for us is the code for addition to get us started:

```
`ALU_OPRN_WIDTH'h01 : result = op1 + op2; // addition
```

Using this code snippet as reference, we can tell that the first segment dictates the operation code. Addition is the hexadecimal value 0x1, so it is denoted as ‘h01’ in the code, with the ‘h’ representing it as hexadecimal instead of ‘0x…’ Therefore, for the next eight operations, we will simply be incrementing the operation code up to 9. The second half is simply the operating portion of the code. All that needs to be done is swapping the ‘+’ in the addition for whatever operation needs to be performed. Below are the completed ALU operations.

```
`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;
```

C. ALU Testing

In the ‘prj_01_tb.v’ test bench file, there is work to still be completed. Firstly, test cases must be created. Three are provided as reference as shown below:

```
// test 15 + 3 = 18
#5  op1_reg=15;
    op2_reg=3;
    oprn_reg=`ALU_OPRN_WIDTH'h01;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h02;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h01;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
```

As shown, there are three given test cases. The ‘#5’ means that the program waits 5ns before performing that line. This information is found at the top of the file:

```
`timescale 1ns/10ps
// Name: prj_01_tb.v
// Module: prj_01_tb
```

In the ‘oprn_reg,’ it is seen that the operation code is changed to test different operations. The two fields above it are ‘op1_reg’ and ‘op2_reg,’ which are the operands to be used with the operation. The last line calls a function found within the same file that will test the operations with the fields we determine. Addition and subtraction are already tested, so below are all the other test cases that have been added:

```
// test 15 + 3 = 18
#5  op1_reg=15;
    op2_reg=3;
    oprn_reg=`ALU_OPRN_WIDTH'h01;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h02;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h01;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=2;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h03;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=4;
    op2_reg=0;
    oprn_reg=`ALU_OPRN_WIDTH'h03;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=1;
    oprn_reg=`ALU_OPRN_WIDTH'h04;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=2;
    oprn_reg=`ALU_OPRN_WIDTH'h04;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=3;
    oprn_reg=`ALU_OPRN_WIDTH'h04;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=1;
    oprn_reg=`ALU_OPRN_WIDTH'h05;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=2;
    oprn_reg=`ALU_OPRN_WIDTH'h05;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=7;
    op2_reg=3;
    oprn_reg=`ALU_OPRN_WIDTH'h05;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h06;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h07;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=15;
    op2_reg=5;
    oprn_reg=`ALU_OPRN_WIDTH'h08;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=2;
    op2_reg=1;
    oprn_reg=`ALU_OPRN_WIDTH'h09;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
#5  op1_reg=1;
    op2_reg=2;
    oprn_reg=`ALU_OPRN_WIDTH'h09;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
```

It is important to have many test cases to ensure that all operations work as intended. As seen above, there are multiple test cases for some of the operations to double-check accuracy.

Now that the test cases are done, we must finish the final part of this file, which is located near the bottom. We need to complete the printing of the test cases. Again, addition is provided for us:

```
`ALU_OPRN_WIDTH'h01 : begin $write("+ "); golden = op1 + op2; end
```

As shown, the first half is once again the same as implementing the ALU. It is simply the operation code. Similarly, the second half is also near the same. Instead of 'result,' it is now 'golden,' as well as the operation in quotation marks. All that is necessary now is to repeat this for all operations. Below is the completed code:

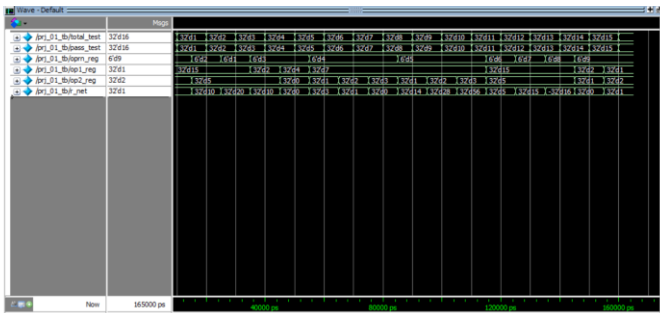
```
`ALU_OPRN_WIDTH'h01 : begin $write("+ "); golden = op1 + op2; end
`ALU_OPRN_WIDTH'h02 : begin $write("- "); golden = op1 - op2; end
`ALU_OPRN_WIDTH'h03 : begin $write("* "); golden = op1 * op2; end
`ALU_OPRN_WIDTH'h04 : begin $write(">> "); golden = op1 >> op2; end
`ALU_OPRN_WIDTH'h05 : begin $write("<< "); golden = op1 << op2; end
`ALU_OPRN_WIDTH'h06 : begin $write("& "); golden = op1 & op2; end
`ALU_OPRN_WIDTH'h07 : begin $write("| "); golden = op1 | op2; end
`ALU_OPRN_WIDTH'h08 : begin $write("~| "); golden = ~(op1 | op2); en
`ALU_OPRN_WIDTH'h09 : begin $write("< "); golden = op1 < op2; end
default: begin $write("? "); golden = `DATA_WIDTH'hx; end
```

VI. Testing

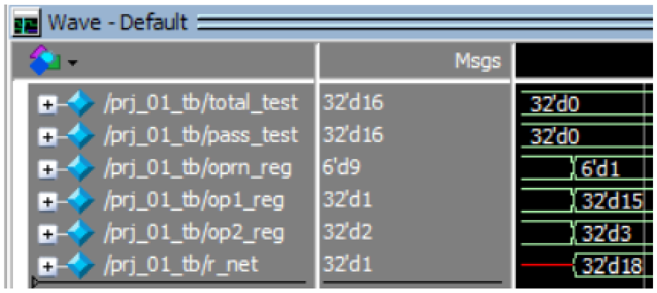
Referring back to Step VI, we will be simulating the now completed code. Following the steps, results are printed to the 'Transcript' tab at the bottom. Pasted are the results of the test:

```
# [TEST] 15 + 3 = 18 , got 18 ... [PASSED]
# [TEST] 15 - 5 = 10 , got 10 ... [PASSED]
# [TEST] 15 + 5 = 20 , got 20 ... [PASSED]
# [TEST] 2 * 5 = 10 , got 10 ... [PASSED]
# [TEST] 4 * 0 = 0 , got 0 ... [PASSED]
# [TEST] 7 >> 1 = 3 , got 3 ... [PASSED]
# [TEST] 7 >> 2 = 1 , got 1 ... [PASSED]
# [TEST] 7 >> 3 = 0 , got 0 ... [PASSED]
# [TEST] 7 << 1 = 14 , got 14 ... [PASSED]
# [TEST] 7 << 2 = 28 , got 28 ... [PASSED]
# [TEST] 7 << 3 = 56 , got 56 ... [PASSED]
# [TEST] 15 & 5 = 5 , got 5 ... [PASSED]
# [TEST] 15 | 5 = 15 , got 15 ... [PASSED]
# [TEST] 15 ~| 5 = 4294967280 , got 4294967280 ... [PASSED]
# [TEST] 2 < 1 = 0 , got 0 ... [PASSED]
# [TEST] 1 < 2 = 1 , got 1 ... [PASSED]
#
#       Total number of tests          16
#       Total number of pass           16
```

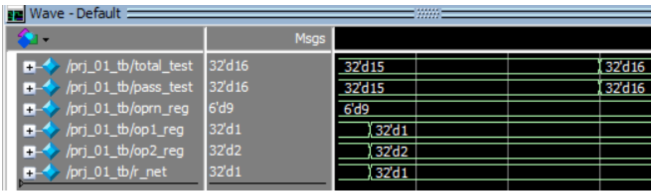
As shown, the 16 tests that span all operations have passed, meaning that both 'alu.v' and 'prj_01_tb.v' were implemented properly. Now we can analyze the clock portion of the test:



The left side shows the registers and their values at the end of runtime while the right side shows what value the register holds at a given time. Let's take a closer, in-depth look at an example test case.


Wave - Default
Msgs
/prj_01_tb/total_test 32'd16 32'd0
/prj_01_tb/pass_test 32'd16 32'd0
/prj_01_tb/oprn_reg 6'd9 6'd1
/prj_01_tb/op1_reg 32'd1 32'd15
/prj_01_tb/op2_reg 32'd2 32'd3
/prj_01_tb/r_net 32'd1 32'd18

For the values displayed in the waves, the value that comes after 'd' is the decimal value that is important to us. As we can see, the current 'total_test' is 0 because this our first operation being performed. Same with 'pass_test' since no test has yet to be run yet. The 'oprn_reg' holds our operation code. In this case, it is 1, meaning addition. Our operands, 'op1_reg' and 'op2_reg' hold 15 and 3 respectively. Therefore, the 'r_net,' or our result, contains 18 because 15 + 3 = 18. This behavior can be seen throughout all operations. Just to drive the point home, let's look at the final operation.


Wave - Default
Msgs
/prj_01_tb/total_test 32'd16 32'd15 32'd16
/prj_01_tb/pass_test 32'd16 32'd15 32'd16
/prj_01_tb/oprn_reg 6'd9 6'd9
/prj_01_tb/op1_reg 32'd1 32'd1
/prj_01_tb/op2_reg 32'd2 32'd2
/prj_01_tb/r_net 32'd1 32'd1

At the beginning of this segment, 15 tests have been completed with 15 passed tests. Operation code 9 is loaded in, which is "set less than," and the operands are 1 and 2. Since 1 < 2, which is true, the result is sent as 1. At the very end, the total tests and passed tests are both set to 16. These final values are what dictates the 'Msgs' tab on the left half. With this, we have analyzed how the 'Transcript' and 'Wave' tab can output the test results, proving that all test cases have passed.

VII. Conclusion

At the end of this project and report, I have learned how to install, navigate, and operate ModelSim. Additionally, I have learned the Verilog syntax and how waveforms are output during simulation. With a 100% success rate (at least on my generated test cases), I am ready for future Verilog projects. With that, I have completed the first CS147 project in which I implemented an ALU through Verilog in ModelSim.

References

[1] K. Patra, Class Lecture, Topic: "Computer System, Instruction Set, ALU," San José State University, San José, Jan., 28, 2020.