Implementation of the Arithmetic & Logic Unit

Craig Huff
CS 147 Section 01
San Jose State University
craig.huff@sjsu.edu

***Abstract*— This report will detail the design and implementation of the Arithmetic and Logic Unit (ALU) using the ModelSim simulator. The report will contain:**

1) **Steps to install the simulator, ModelSim, and the setup process for a project.**
2) **Implementing the 'CS 147 DV' commands into the ALU**
3) **Implementing test methods for the ALU using the Verilog HDL language**
4) **Running a simulation**
5) **Generating waveforms from the ALU using the test methods**

I. Installing ModelSim

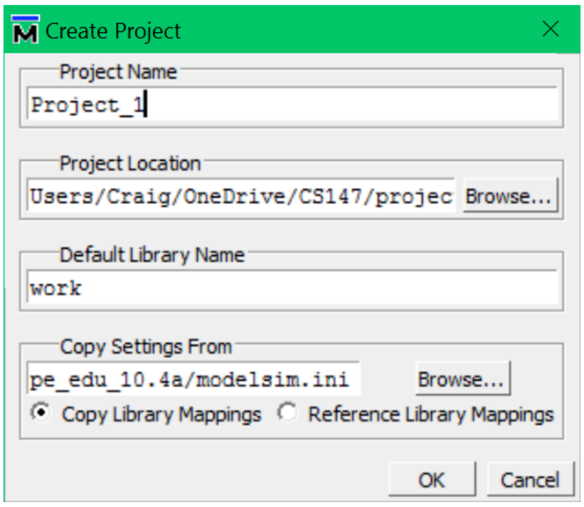
The ModelSim program can be downloaded for free from http://www.mentor.com/company/higher_ed/modelsim-student-edition using any web browser. After navigating to the website, select "Download Student Edition" and direct the computer where to save the installer. To ensure the program will install properly, there are five steps: (1) Run the installer and complete the installation process. (2) Using the same web browser as before, a form will appear which requires the user to fill with their basic information. The necessary fields can include name, phone number, address, email address, university name, etc. (3) Check the email provided in the step above for the ModelSim license attachment. (4) Follow the instructions provided to save the attachment in the top-level directory of the ModelSim PE Student Edition. (5) Ensure that the file remains unchanged throughout this process, as modifications to the file will cause the installation to fail. Once this step is complete, ModelSim is ready to be used.

II. Preparing a Simulator Project

This section demonstrates how to make a new project and how to open files from a folder in the new project.

To prepare a project in ModelSim, we first need to download the .zip file provided for the project. Within in .zip file, there will be three files with the extension '.v'. They are named as follows: 'alu.v' , 'prj_definition.v' , and 'prj_01_tb.v'.

To create a new project within the ModelSim program, navigate to File > New > Project. The window that appears is shown in figure 2.1.


Create Project

Project Name: Project_1

Project Location: Users/Craig/OneDrive/CS147/projec

Default Library Name: work

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

[✓ Copy Library Mappings] Reference Library Mappings

Figure 2.1. Creating a new project

Now that a new project has been made, we can add the files into the project. Since the files necessary for the project have been downloaded, we can use existing files. See figure 2.2

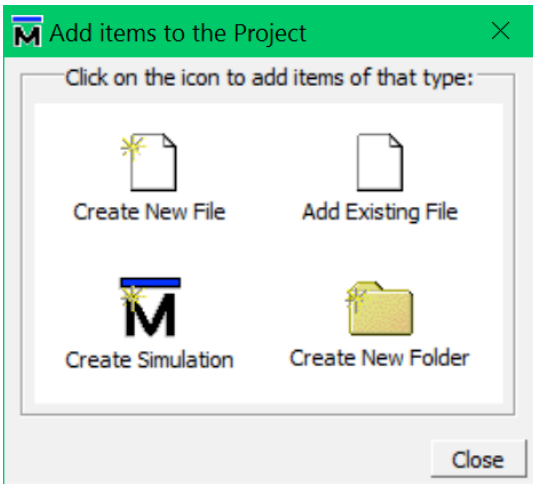
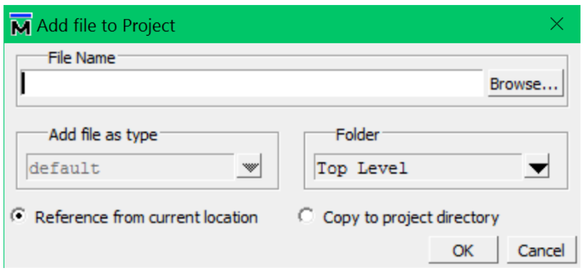


Figure 2.2. Adding existing files

These files will be in the directory of the downloaded .zip file from the step above. Finding and selecting these files is demonstarted in Figures 2.3 and 2.4.


Add file to Project

File Name

Browse...

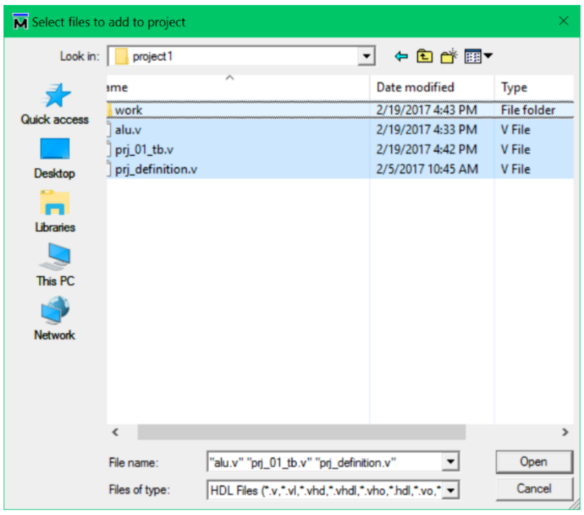
Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

OK Cancel

Figure 2.3. Selecting files to be added


Select files to add to project

Look in: project1

Quick access, Desktop, Libraries, This PC, Network

Name	Date modified	Type
work	2/19/2017 4:43 PM	File folder
alu.v	2/19/2017 4:33 PM	V File
prj_01_tb.v	2/19/2017 4:42 PM	V File
prj_definition.v	2/5/2017 10:45 AM	V File

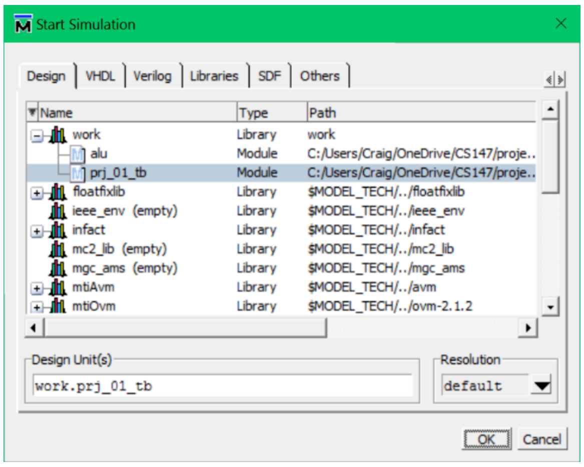
File name: "alu.v" "prj_01_tb.v" "prj_definition.v"

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

Open Cancel

Figure 2.4. Finding the files in the file directory

After all the files have been selected and added into the newly created project, the projects must be compiled by pressing the compile button . Compiling the files allows for them to be run after the implementation is complete. The simulator can be started by pressing the simulate button . In the next window that appears, select the work library and the file 'prj_01_tb' to start the simulation on the correct file. See Figure 2.5


Start Simulation

Design | VHDL | Verilog | Libraries | SDF | Others

Name	Type	Path
work	Library	work
alu	Module	C:/Users/Craig/OneDrive/CS147/proje..
prj_01_tb	Module	C:/Users/Craig/OneDrive/CS147/proje..
floatfixlib	Library	$MODEL_TECH/../floatfixlib
ieee_env (empty)	Library	$MODEL_TECH/../ieee_env
infact	Library	$MODEL_TECH/../infact
mc2_lib (empty)	Library	$MODEL_TECH/../mc2_lib
mgc_ams (empty)	Library	$MODEL_TECH/../mgc_ams
mtiAvm	Library	$MODEL_TECH/../avm
mtiOvm	Library	$MODEL_TECH/../ovm-2.1.2

Design Unit(s): work.prj_01_tb

Resolution: default

OK Cancel

Figure 2.5 Selecting files to simulate

To see the waveforms of the processor while the simulator is running, right click on the file 'prj_01_tb' and click 'Add Wave' or press 'Ctrl + W' on the keyboard as shown in Figure 2.6.

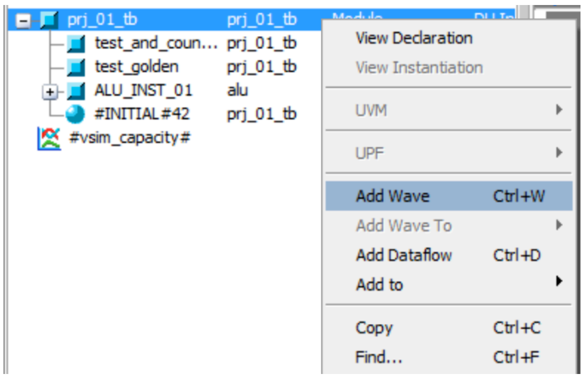

prj_01_tb	prj_01_tb
test_and_coun...	prj_01_tb
test_golden	prj_01_tb
ALU_INST_01	alu
#INITIAL#42	prj_01_tb
#vsim_capacity#	

Figure 2.6. Adding waveforms to be watched

Finally, to reset the simulation, specify how long to run the simulation (in nanoseconds), and run the simulation, the buttons that control these functions are displayed in Figure 2.7

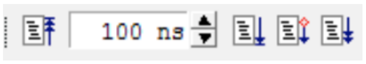


Figure 2.7. Tools to run simulations

III. Requirements of the ALU

The Arithmetic & Logic Unit (ALU) provides the basic functions of a computer. All mathematic and logical programs are handled by the ALU. It is a circuit that handles the logical operations of the processor such as logical AND, OR, NOR, the arithmetic operations of addition, subtraction, multiplication, bitwise AND, bitwise OR, and logical shifts left and right.

All mathematical and logical programs are processed by the ALU and are broken down into the terms of operands and operations. In the program the operands are defined as 'op1_reg' and 'op2_reg', the operation is defined as 'oprn_reg'.

The ALU takes the two registers 'op1_reg' and 'op2_reg' applies the operation specified by 'oprn_reg', which is also known as the op-code, and then saves the output in the result 'r_net'.

IV. Design & Implementation of ALU

Using the ModelSim simulator, it is simple to program the basic functions of the ALU. This section will outline the design of the ALU using the Verilog Hardware Descripting Language (VHDL).

A. Design Strategy

The design of the ALU is simple. It's designed to take two operands and one operation as the input and return the result of the process as the output. The ALU built in this project is a 32-bit processor, which makes each of the operands 32-bit, the operand 6-bit, and the output as 32-bit. Each operation implemented in the program is defined with the name 'ALU_OPRN_WIDTH'h0(#) where the number sign(#)

defines the specific operation. For example, 'h02 is Subtraction. The operands have the names ‘op1’ and ‘op2’. The resulting output is referred to as ‘golden’.

B. Operations Handled by the ALU & Their Implementation

There are nine operations that are handled by the ALU. They are listed as followed with their implementations following.

- ‘'h01’ - Addition: golden = op1 + op2
- ‘'h02’ - Subtraction: golden = op1 − op2
- ‘'h03’ - Multiplication: golden = op1 * op2
- ‘'h04’ - Bitwise AND: golden = op1 & op2
- ‘'h05’ - Bitwise OR: golden = op1 | op2
- ‘'h06’ - Bitwise NOR: golden = ~ (op1 | op2)
- ‘'h07’ - Set Less Than: golden = op1 < op2
- ‘'h08’ - Logical Left Shift: golden = op1 << op2
- ‘'h09’ - Logical Right Shift: golden = op1 >> op2

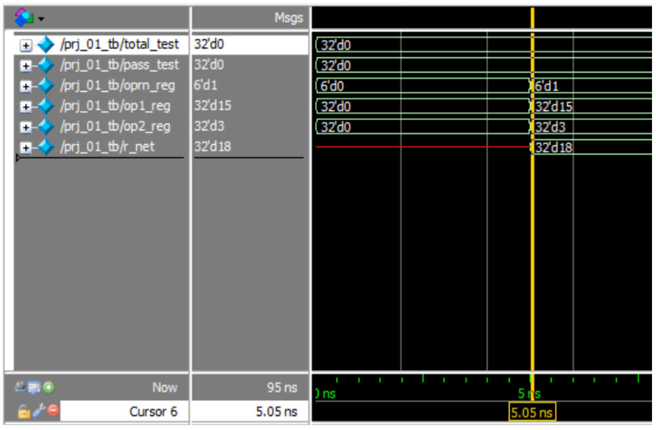
V. Test Strategy & Test Implementation

With the implementation done in the file named ‘alu.v. The testing is done in the file named ‘prj_01_tb.v’, which will provide the operands, operations, and record the result to make sure the ALU’s design is correct. Following the instructions from above in Section II to compile and simulate the program, it’s possible to ensure the ALU is designed correctly. The tests are done by comparing the results of the tests named ‘golden’ within the ‘prj_01_tb.v’ file against the results collected from the ‘alu.v’ methods.

Note: All descriptions of the test will be referring to the variables defined in the ‘test_golden’ function defined within the file ‘prj_01_tb.v’

A. Addition

op1_reg=15;
op2_reg=3;
oprn_reg=`ALU_OPRN_WIDTH'h01;



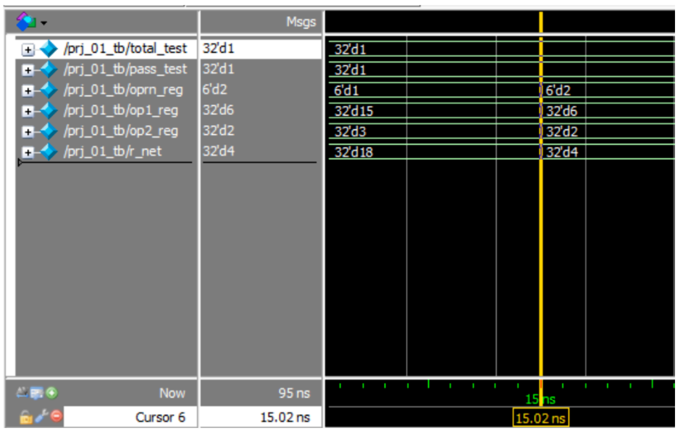
```
                          Msgs
/prj_01_tb/total_test     32'd0     32'd0
/prj_01_tb/pass_test      32'd0     32'd0
/prj_01_tb/oprn_reg       6'd1      6'd0      6'd1
/prj_01_tb/op1_reg        32'd15    32'd0     32'd15
/prj_01_tb/op2_reg        32'd3     32'd0     32'd3
/prj_01_tb/r_net          32'd18              32'd18

Now                       95 ns     0 ns      5 ns
Cursor 6                  5.05 ns             5.05 ns
```

Figure 5.1. Addition Waveform

B. Subtraction

op1_reg=6;
op2_reg=2;
oprn_reg=`ALU_OPRN_WIDTH'h02;



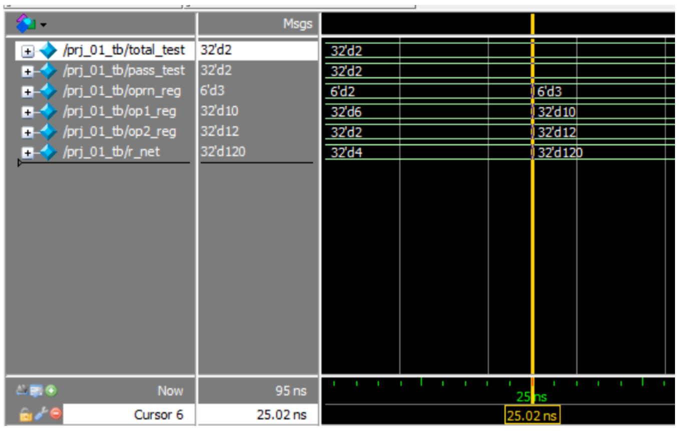
```
                          Msgs
/prj_01_tb/total_test     32'd1     32'd1
/prj_01_tb/pass_test      32'd1     32'd1
/prj_01_tb/oprn_reg       6'd2      6'd1      6'd2
/prj_01_tb/op1_reg        32'd6     32'd15    32'd6
/prj_01_tb/op2_reg        32'd2     32'd3     32'd2
/prj_01_tb/r_net          32'd4     32'd18    32'd4

Now                       95 ns               15 ns
Cursor 6                  15.02 ns            15.02 ns
```

Figure 5.2. Subtraction Waveform

C. Multiplication

op1_reg=10;
op2_reg=12;
oprn_reg=`ALU_OPRN_WIDTH'h03



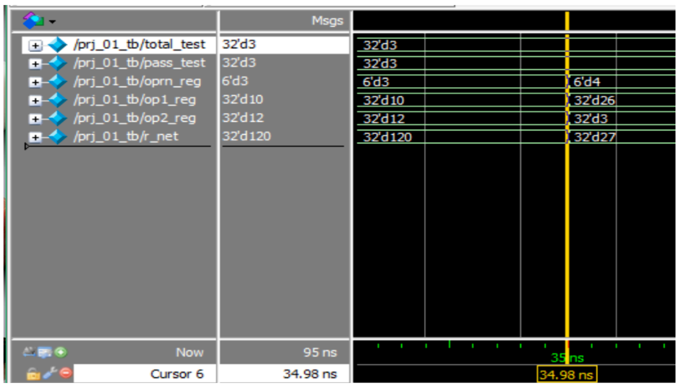
```
                          Msgs
/prj_01_tb/total_test     32'd2     32'd2
/prj_01_tb/pass_test      32'd2     32'd2
/prj_01_tb/oprn_reg       6'd3      6'd2      6'd3
/prj_01_tb/op1_reg        32'd10    32'd6     32'd10
/prj_01_tb/op2_reg        32'd12    32'd2     32'd12
/prj_01_tb/r_net          32'd120   32'd4     32'd120

Now                       95 ns               25 ns
Cursor 6                  25.02 ns            25.02 ns
```

Figure 5.3. Multiplication Waveform

D. Bitwise AND

op1_reg=26;
op2_reg=3;
oprn_reg=`ALU_OPRN_WIDTH'h04;



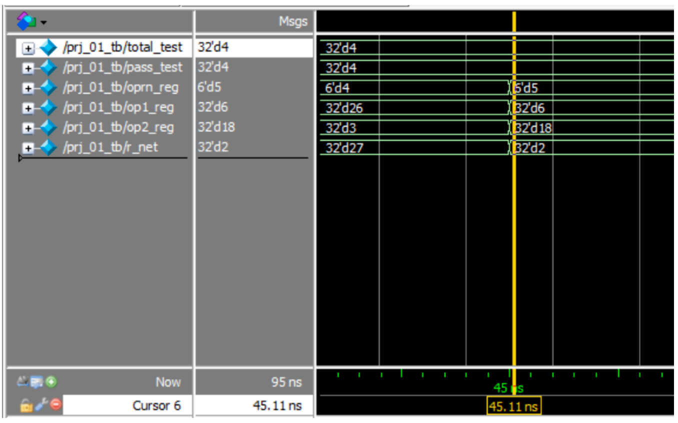
```
                          Msgs
/prj_01_tb/total_test     32'd3     32'd3
/prj_01_tb/pass_test      32'd3     32'd3
/prj_01_tb/oprn_reg       6'd3      6'd3      6'd4
/prj_01_tb/op1_reg        32'd10    32'd10    32'd26
/prj_01_tb/op2_reg        32'd12    32'd12    32'd3
/prj_01_tb/r_net          32'd120   32'd120   32'd27

Now                       95 ns               35 ns
Cursor 6                  34.98 ns            34.98 ns
```

Figure 5.4. Bitwise AND Waveform

E. Bitwise OR

op1_reg=6;
op2_reg=18;
oprn_reg=`ALU_OPRN_WIDTH'h05;



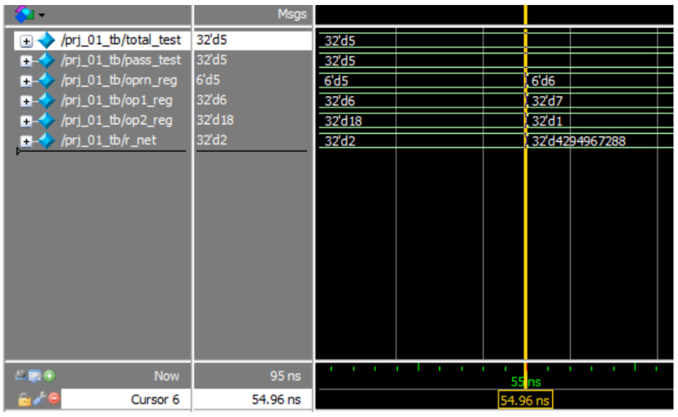
```
                          Msgs
/prj_01_tb/total_test     32'd4    32'd4
/prj_01_tb/pass_test      32'd4    32'd4
/prj_01_tb/oprn_reg       6'd5     6'd4        6'd5
/prj_01_tb/op1_reg        32'd6    32'd26      32'd6
/prj_01_tb/op2_reg        32'd18   32'd3       32'd18
/prj_01_tb/r_net          32'd2    32'd27      32'd2

Now       95 ns                    45 ns
Cursor 6  45.11 ns                 45.11 ns
```

Figure 5.5. Bitwise OR Waveform

F. Bitwise NOR

op1_reg=7;
op2_reg=1;
oprn_reg=`ALU_OPRN_WIDTH'h06;



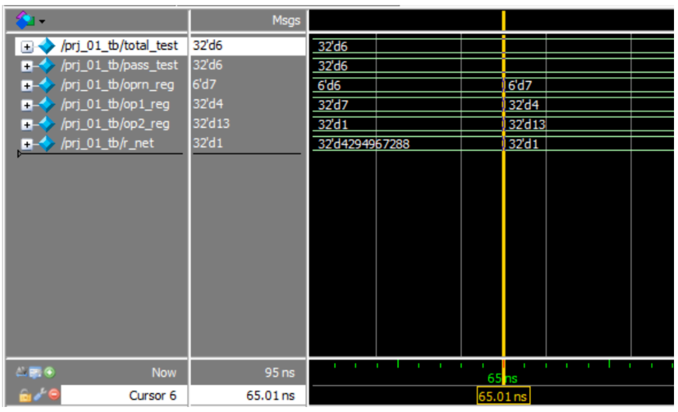
```
                          Msgs
/prj_01_tb/total_test     32'd5    32'd5
/prj_01_tb/pass_test      32'd5    32'd5
/prj_01_tb/oprn_reg       6'd5     6'd5        6'd6
/prj_01_tb/op1_reg        32'd6    32'd6       32'd7
/prj_01_tb/op2_reg        32'd18   32'd18      32'd1
/prj_01_tb/r_net          32'd2    32'd2       32'd4294967288

Now       95 ns                    55 ns
Cursor 6  54.96 ns                 54.96 ns
```

Figure 5.6. Bitwise NOR Waveform

G. Set Less Than

op1_reg=4;
op2_reg=13;
oprn_reg=`ALU_OPRN_WIDTH'h07;



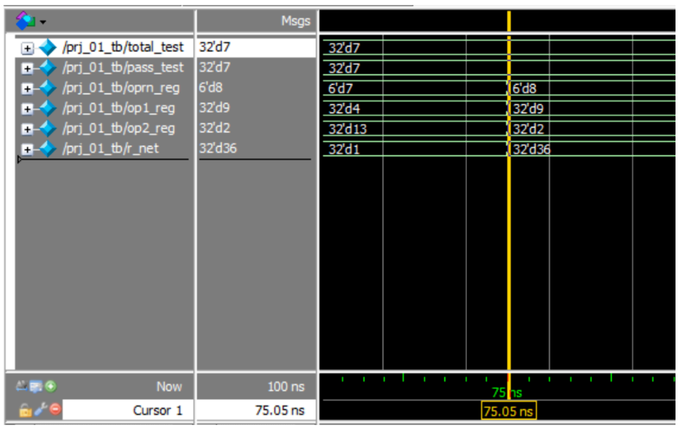
```
                          Msgs
/prj_01_tb/total_test     32'd6    32'd6
/prj_01_tb/pass_test      32'd6    32'd6
/prj_01_tb/oprn_reg       6'd7     6'd6               6'd7
/prj_01_tb/op1_reg        32'd4    32'd7              32'd4
/prj_01_tb/op2_reg        32'd13   32'd1              32'd13
/prj_01_tb/r_net          32'd1    32'd4294967288     32'd1

Now       95 ns                    65 ns
Cursor 6  65.01 ns                 65.01 ns
```

Figure 5.7. Set Less Than Waveform

H. Logical Shift Left

op1_reg=9;
op2_reg=2;
oprn_reg=`ALU_OPRN_WIDTH'h08;



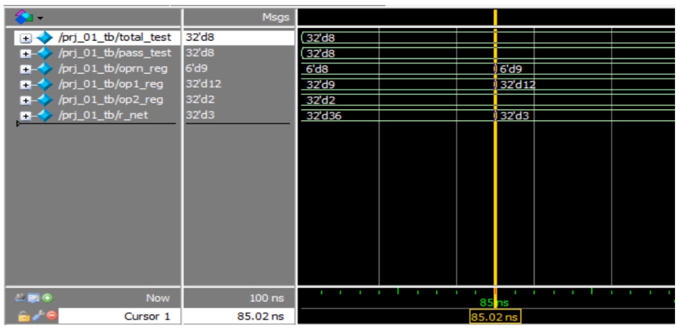
```
                          Msgs
/prj_01_tb/total_test     32'd7    32'd7
/prj_01_tb/pass_test      32'd7    32'd7
/prj_01_tb/oprn_reg       6'd8     6'd7        6'd8
/prj_01_tb/op1_reg        32'd9    32'd4       32'd9
/prj_01_tb/op2_reg        32'd2    32'd13      32'd2
/prj_01_tb/r_net          32'd36   32'd1       32'd36

Now       100 ns                   75 ns
Cursor 1  75.05 ns                 75.05 ns
```

Figure 5.8. Shift Logical Left Waveform

I. *Logical Shift Right*

op1_reg=12;
op2_reg=2;
oprn_reg=`ALU_OPRN_WIDTH'h09;
Note: The op2_reg will not change its value in the waveform since the value remains the same from 'h08 to 'h09. Therefore, it appears to be gone from the waveform in Figure 5.9.

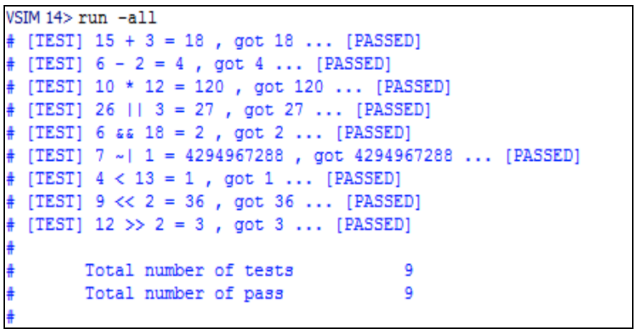


```
                          Msgs
/prj_01_tb/total_test     32'd8    32'd8
/prj_01_tb/pass_test      32'd8    32'd8
/prj_01_tb/oprn_reg       6'd9     6'd8        6'd9
/prj_01_tb/op1_reg        32'd12   32'd9       32'd12
/prj_01_tb/op2_reg        32'd2    32'd2
/prj_01_tb/r_net          32'd3    32'd36      32'd3

Now       100 ns                   85 ns
Cursor 1  85.02 ns                 85.02 ns
```

Figure 5.9. Shift Logical Right

The ALU operations test displayed by wavelengths in Figure 5.10 can also be represented in text as is shown in figure 5.11.



```
VSIM 14> run -all
# [TEST] 15 + 3 = 18 , got 18 ... [PASSED]
# [TEST] 6 - 2 = 4 , got 4 ... [PASSED]
# [TEST] 10 * 12 = 120 , got 120 ... [PASSED]
# [TEST] 26 || 3 = 27 , got 27 ... [PASSED]
# [TEST] 6 && 18 = 2 , got 2 ... [PASSED]
# [TEST] 7 ~| 1 = 4294967288 , got 4294967288 ... [PASSED]
# [TEST] 4 < 13 = 1 , got 1 ... [PASSED]
# [TEST] 9 << 2 = 36 , got 36 ... [PASSED]
# [TEST] 12 >> 2 = 3 , got 3 ... [PASSED]
#
#       Total number of tests           9
#       Total number of pass            9
#
```

Figure 5.10. Results from all ALU tests

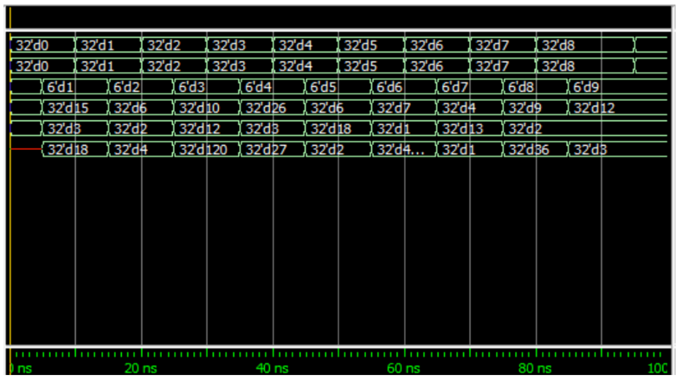

32'd0 32'd1 32'd2 32'd3 32'd4 32'd5 32'd6 32'd7 32'd8
32'd0 32'd1 32'd2 32'd3 32'd4 32'd5 32'd6 32'd7 32'd8
6'd1 6'd2 6'd3 6'd4 6'd5 6'd6 6'd7 6'd8 6'd9
32'd15 32'd6 32'd10 32'd26 32'd6 32'd7 32'd4 32'd9 32'd12
32'd3 32'd2 32'd12 32'd3 32'd18 32'd1 32'd13 32'd2
32'd18 32'd4 32'd120 32'd27 32'd2 32'd4... 32'd1 32'd36 32'd3
0 ns
20 ns
40 ns
60 ns
80 ns
100

Figure 5.11. Entire Simulation Wavelength

VI. Conclusion

From this project, I learned how to dual boot Windows on a Mac computer, how to install the ModelSim simulator and created a project in the Verilog HDL language. This allowed for me to create a 32-bit Arithmetic & Logic Unit and implement the nine operations defined in 'CS147 DV'. Overall, I was able to learn a lot about simulations, the waveforms they produce, and how to manipulate projects using the Verilog language.