Arithmetic Logic Unit(ALU) Behavioural Modeling

Zachary Ian May
San Jose State University
Sunnyvale, CA
Zachary.May@sjsu.edu

***Abstract*—This Project deals primarily with the creation of a model of an Arithmetic Logic Unit, through this project we are meant to set up our work environment and become familiar with Verilog.**

***Keywords*—ALU, Behavioral Modeling, Verilog**

I. Introduction

This projects goal is to model the behavior of an Arithmetic Logic Unit using the ModelSim Verilog Simulation Program. Through this project, we should be able to implement basic functions such as addition, left shifts, bitwise operations, and comparative operations. Additionally, it should respond to changes in input by updating the operation and result, which is to be outputted such that we can monitor and test the outcome.

II. Instalation of the Simulator

The Simulation software being used for CS147, and by extent this project is ModelSim by MentorGraphics. This software allows us to model dataflow and behavior of logic circuits. Through use of this simulator we intend to model the behavior of an ALU.

The steps to install this simulator are as follows:

a) First we go to the link: Download Page. We are using the Student Edition of the software, this software only works for Windows, should this be a problem other software exists, additionally virtualization is an option. These options will not be covered in this report.

b) Next, select the download button.

  i) Upon selecting the download button, the site will give a prompt to create an account, proceed through this step.

c) After the installer download is finished, run the installer and go through normal installation steps.

  i) Be sure to note down which location the software is installed in as it will be needed later for installing the license.

d) Once the installer has finished, a license will need to be generated, the installer will give a prompt to do this, follow its steps and fill out the provided form.

e) Once the form has been finished, an email with the license attached will be sent to the email provided in the form, download the file and place it in the top directory of the installation location.

f) Once these steps have been completed the simulator is ready for this project.

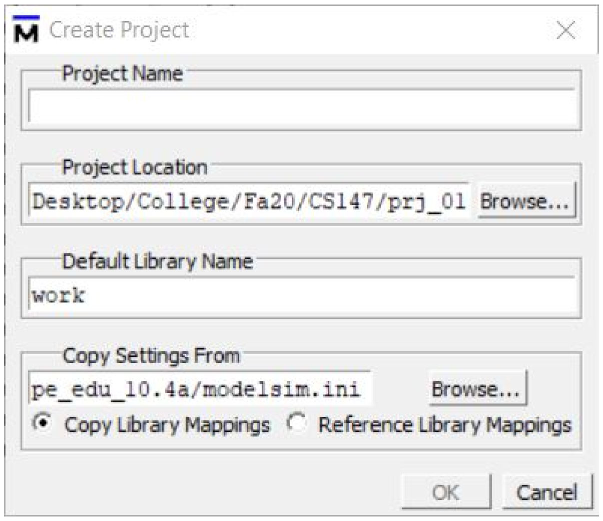
III. Project creation

In order to begin work on a project, within the simulator we must create a project.

a) Once the simulator is open, select from the top bar File>New>Project…

b) This will bring up the dialog window in Figure 2.1. In this dialog window enter the name of the project and select a folder for the project location.

  i) It is not advised to leave the location as default.



Project Name:
Project Location: Desktop/College/Fa20/CS147/prj_01
Default Library Name: work
Copy Settings From: pe_edu_10.4a/modelsim.ini
[✓ Copy Library Mappings] Reference Library Mappings

Fig 2.1

c) Selecting the "ok" button will create the project.

d) From here, another dialog window will appear (fig. 2.2), through this window you are able to add any files that are required for the project (as in fig. 2.3). The files used in this project are provided in a zip, the files being: alu.v, prj_01_tb.v, and prj_definition.v

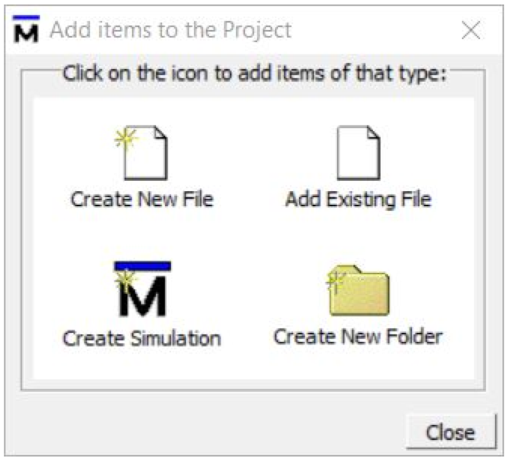


Fig 2.2

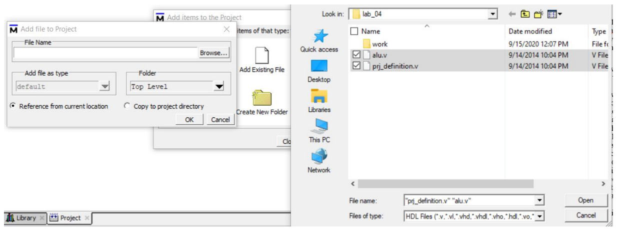
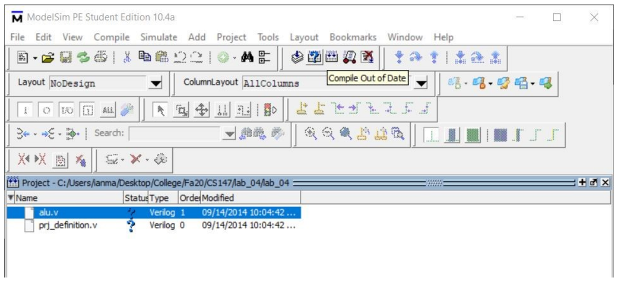


Fig 2.3

e) Once all files have been added they need to be compiled.

   i) First select one of the files in the main window

   ii) Then in the toolbar, select the compile out of date button (fig 2.4).


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Project Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Compile Out of Date
Search:

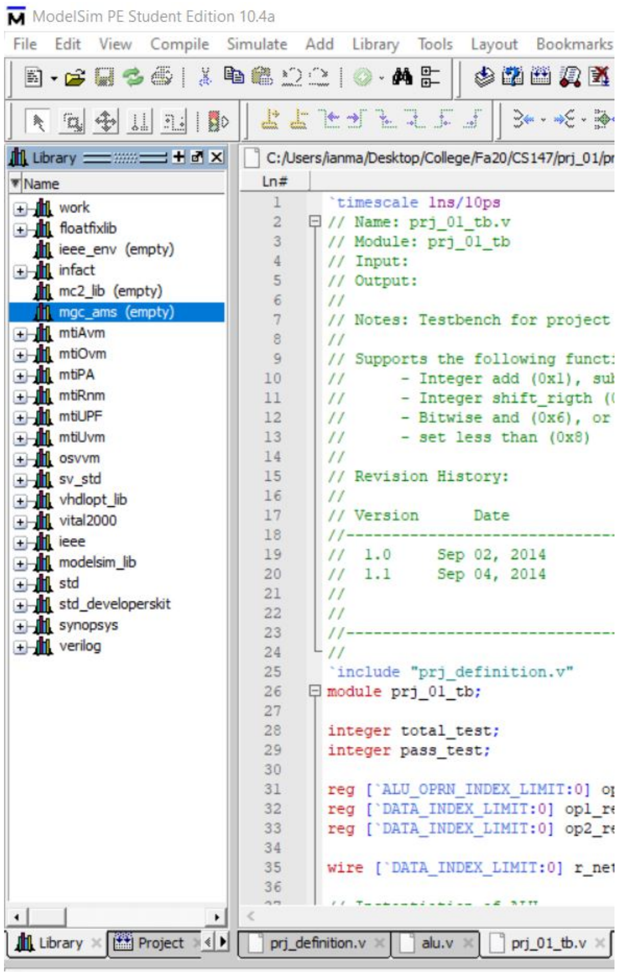
Project - C:/Users/ianma/Desktop/College/Fa20/CS147/lab_04/lab_04

```
Name              Status Type    Order Modified
alu.v             ?      Verilog 1     09/14/2014 10:04:42 ...
prj_definition.v  ?      Verilog 0     09/14/2014 10:04:42 ...
```

Fig 2.4

   iii) This will compile any files in the project that have been updated.

f) Once all files in the project have been compiled, the project is ready to be run. To do this, change the tab from project to library



C:/Users/ianma/Desktop/College/Fa20/CS147/prj_01/pr

```
`timescale 1ns/10ps
// Name: prj_01_tb.v
// Module: prj_01_tb
// Input:
// Output:
//
// Notes: Testbench for project
//
// Supports the following funct
//      - Integer add (0x1), sub
//      - Integer shift_rigth ((
//      - Bitwise and (0x6), or
//      - set less than (0x8)
//
// Revision History:
//
// Version      Date
//---------------------------------
//  1.0      Sep 02, 2014
//  1.1      Sep 04, 2014
//
//
//---------------------------------
//
`include "prj_definition.v"
module prj_01_tb;

integer total_test;
integer pass_test;

reg [`ALU_OPRN_INDEX_LIMIT:0] op
reg [`DATA_INDEX_LIMIT:0] op1_re
reg [`DATA_INDEX_LIMIT:0] op2_re

wire [`DATA_INDEX_LIMIT:0] r_net
```

Fig 2.5

g) Expand the work folder and select the testbench file you would like to run, in this case prj_01_tb, right click and select simulate.

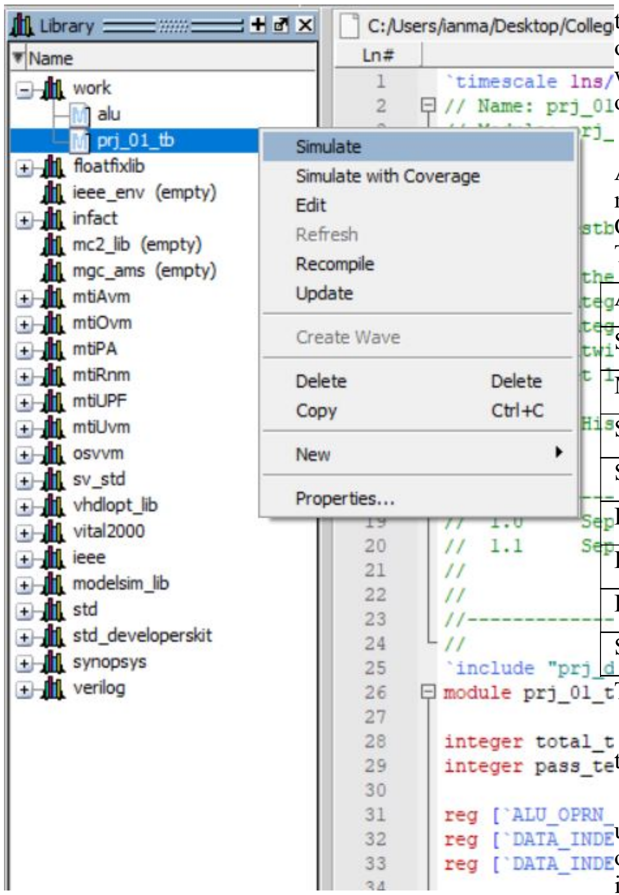


Fig 2.6

h) Once in the simulation view, find the objects window, and select the following objects: oprn_reg, op1_reg, op2_reg, and r_net. Right click these and select add wave. (fig 2.7)

i) Once all this is done, select run all from the toolbar

   i) This will allow you to see what is happening at different points in time, via the wave. The view can be adjusted through the different zoom options available on the tool bar.

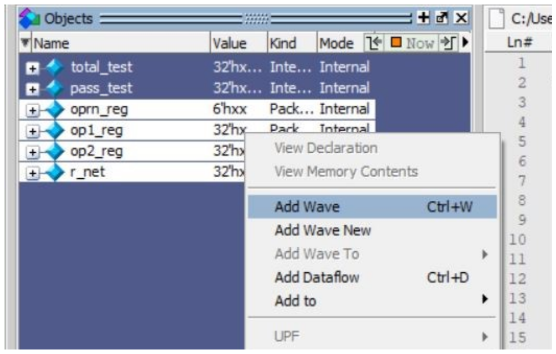


Fig 2.7

IV. ALU Requirements, Design, and Implementation

The ALU is a processor component which does arithmetic using logic gates. The ALU makes use of two three input lines, and one output line. The inputs are the two operands, and the operator. The output line is the result of whichever operation is specified by the code given on the operator line.

The primary requirements for this project are that the ALU have functionalities for addition, subtraction, multiplication, left and right shifting, bitwise operations of OR, AND, and NOR and finally be able to do set less than. These operations are to have opcodes as follows

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

Table 3.1

Most of these operations are simple to implement through the use of basic Verilog commands.

The main design of the ALU behavioral model was to, upon receiving any change to either of the operands or to the operator, execute the new calculation as indicated by the input lines.

The design and a fair amount of the implementation of the ALU was mostly provided in our starter files. The main aspect of the ALU that was to be implemented was the calculations to be done by the ALU. The determination of which calculation was to be done is implemented through a case statement. This statement determines which of the operators is to be used, then makes the calculation. Each case uses its built in Verilog instruction, with the exception of set less than. During testing, it was discovered that negative numbers would be treated as larger than positive numbers, due to the way Verilog treats negative numbers. The twos compliment would result in the negative number being treated as larger. As we are dealing with signed numbers, the solution to this was to check the MSB of the first operand and if that bit is a 1, to use a different statement to determine if the operand is less than operand 2. As a result, the implementation of the ALU's behavioral model took a relatively short amount of time relative the time it took for testing.

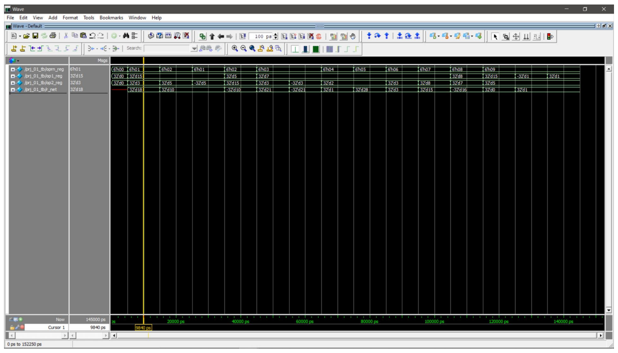
V. Test Strategies and Implementation

In order to test this simulation, a problem that exemplifies each possible problem with each calculation was used. The implementation of testing was a set of problems that is run through both the ALU behavior model, and a testing model, and finally reviewed to ensure that any problems that may have been present in both models would be recognized. Through this implementation, a problem with the set less than function was discovered, where negatives were treated as if they were larger than a positive number.

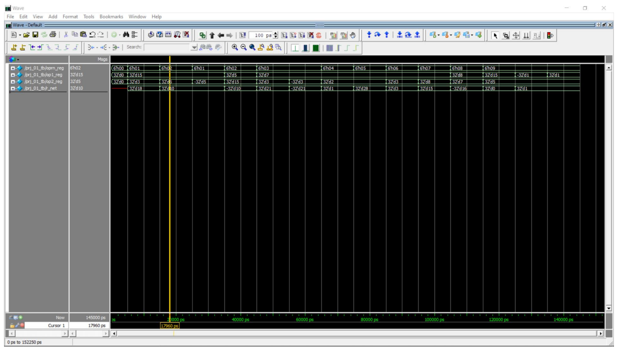
The problem set used to test the behavior model was as follows:

15 + 3, 15-5, 15+(-5), 5-15, 7*3, 7*-3, 7>>2, 7<<2, 7&3, 7|8, 8~|7, 15<5, -1<5, 1<5

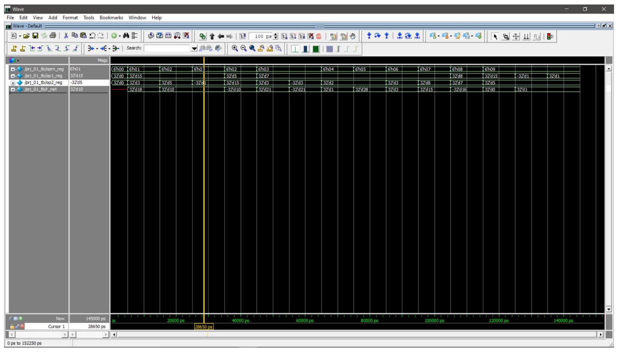
Each of these problem sets returned their correct value when using the final behavioral model. The entries in the wave for each test are given as images in order below.



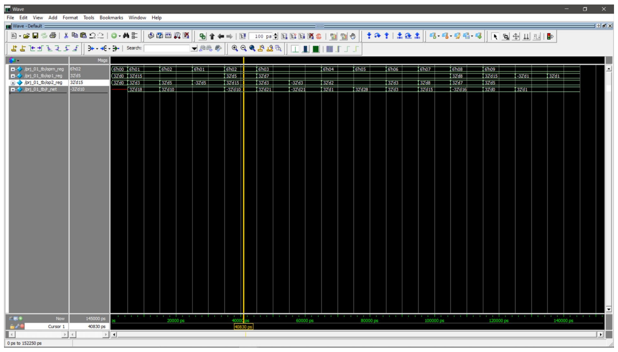
15 + 3



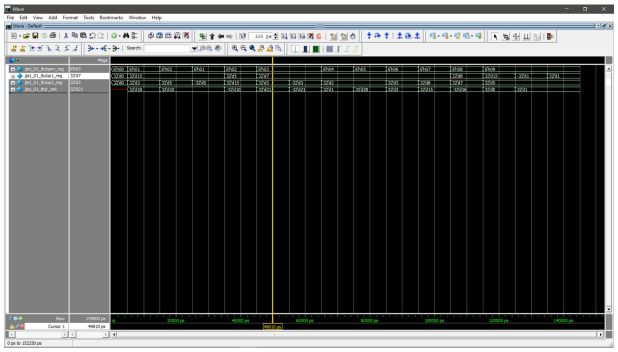
15-5



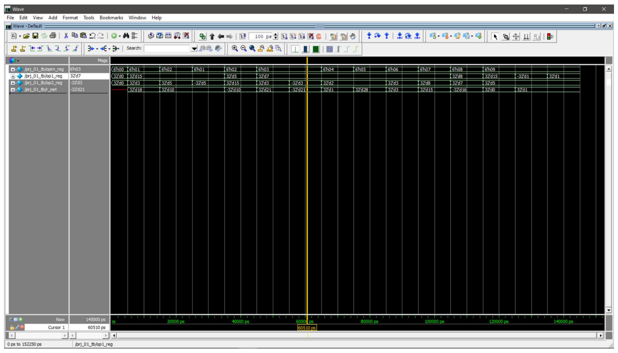
15+(-5)



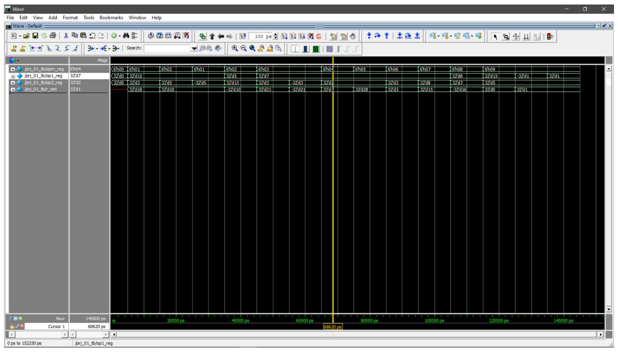
5-15



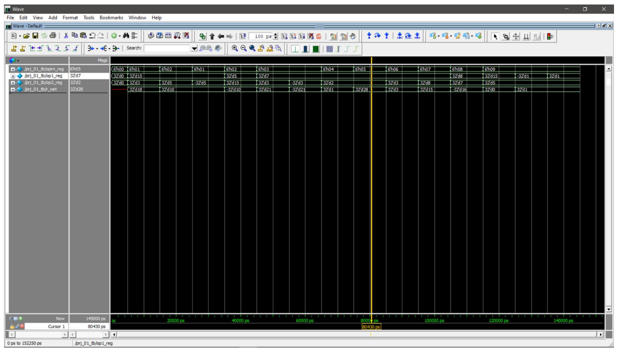
7*3



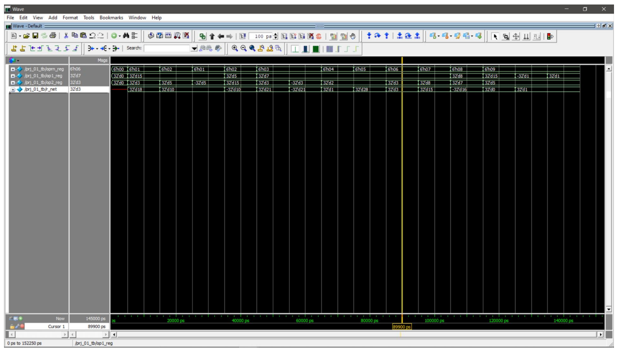
7*-3



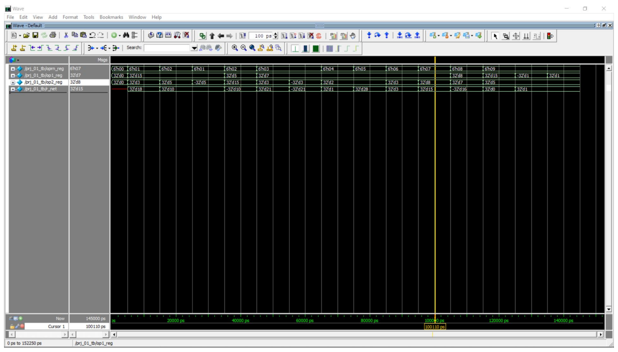
7>>2



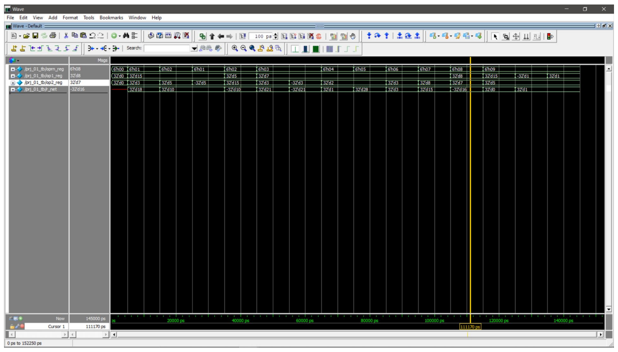
7<<2



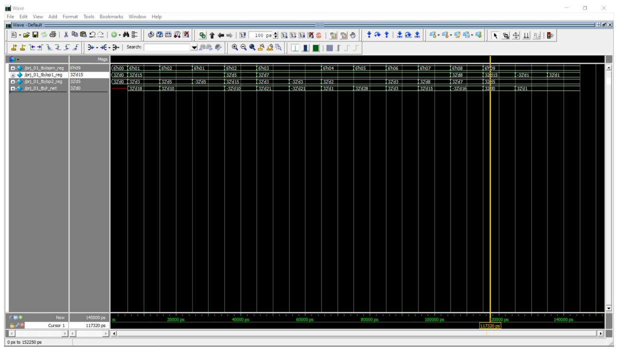
7&3



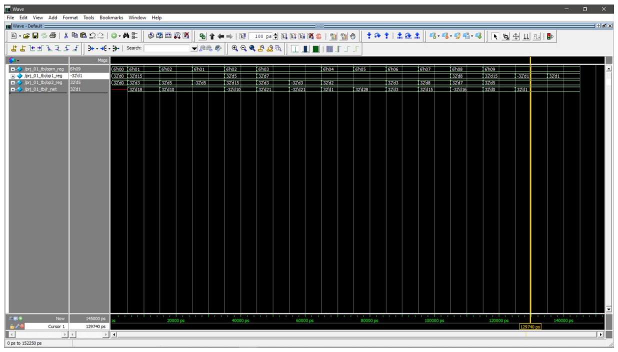
7|8



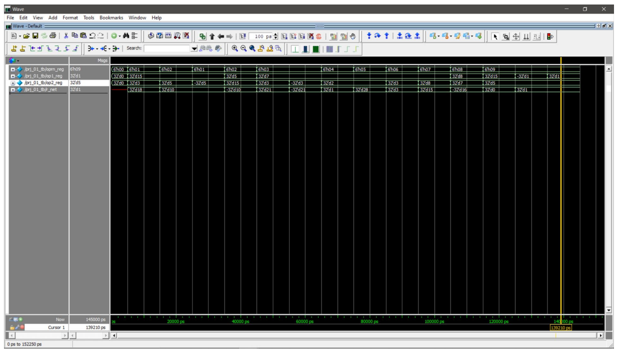
8~|7



15<5



-1<5



1<5

Conclusion

This project has been a good exercise to set up our work environment and to get familiar with some of the operations available in Verilog. Through this we have been able to additionally refresh our familiarity with the operations of an ALU and some of the problems that can occur with negative numbers interaction with positive numbers. Going forward, I believe that most problematic points with modeling the behavior of an ALU have been dealt with, however if there was an aspect which I forgot the continuation of the project would likely be in order to fix such problems.