Arithmetic & Logic Unit (ALU)

Azael Zamora

San Jose State University

azael.zamora12@gmail.com

Abstract: **This report will focus on the design and implementation of a 32-bit Arithmetic and Logic Unit (ALU) using ModelSim simulator. The report will contain the following:**

1) Installation of ModelSim.
2) Implementing the arithmetic & logic unit (ALU) using Verilog HDL.
3) Running a simulation by implementing the test bench code to test the ALU.
4) Create and observe the single waveforms using the ALU test bench code.

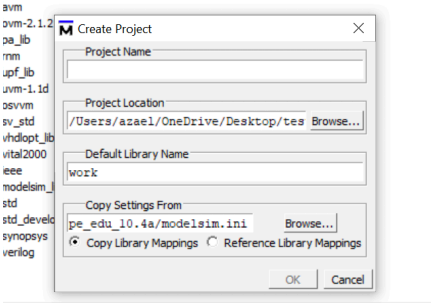
I. Installation of ModelSim

The ModelSim simulator program can be dowloaded for free from https://www.mentor.com/company/higher_ed/modelsim-student-edition from any web browser. Select "Download Student Edition" in order to download the free student version. Open the installation file and complete the installation steps as needed. When the installation is done, a form will appear in the browser, that will require you to provide your name, address, phone number, email, the name of the university. Once the information has been filled out, click "finish". Shortly after, you will receive an email from ModelSim that will contain the license file named "student_license.dat". The email will also tell you where you will need to save the license file, in order to use the simulator. Once the file has been saved in the appropriate directory, you should now be able to run the ModelSim PE Student Edition.

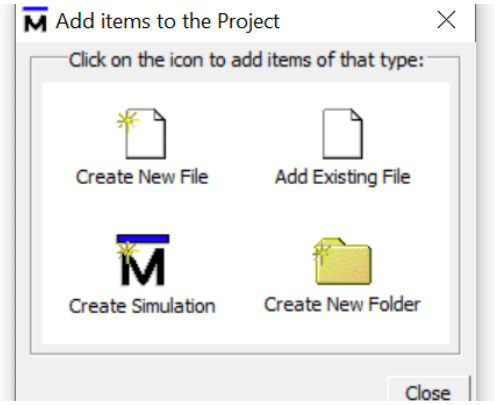
II. Preparation of a project in ModelSim

In order to prepare a project in ModelSim, first we will need to download the .zip file that is provided for this project. Once you unzip the .zip file, the following files should be contained: 'alu.v', 'prj_definition.v', and 'proj_01_tb.v'. For this project you will only need to modify 'alu.v' and 'proj_01_tb', do not modify 'prj_definiton.v'.

Next, open the ModelSim program, and navigate to File > New > Project. The following window should appear:

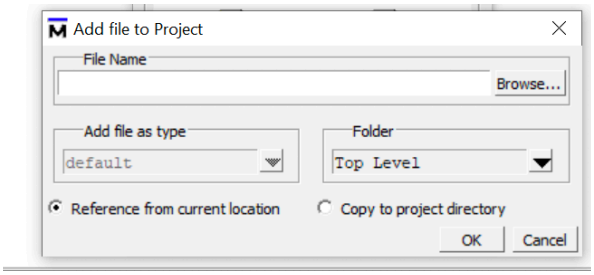


Once the the project folder has been created, you can add the files to the project. Since the necessary files for the project have been downloaded, use 'Add Existing File' to the files into the project.

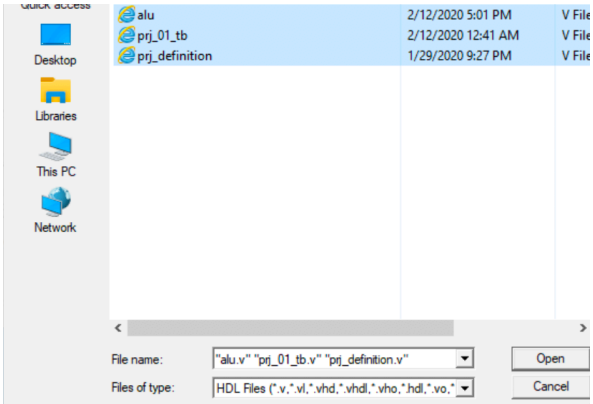


Click on the icon **"Add Existing File"**

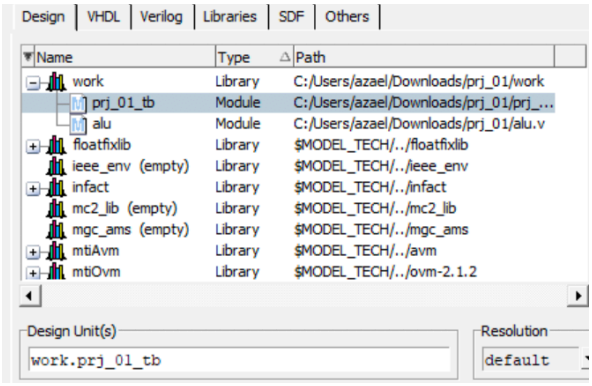
You will need to navigate to the directory of the downloaded .zip file to select three files in order to add them to the project folder you have created.



Click on "**Browse"** to find the three files.

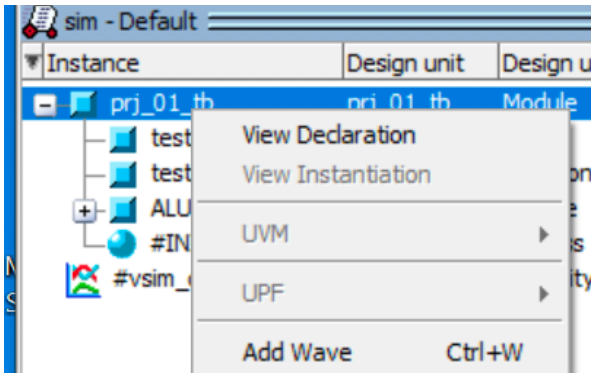


Once the files have been added to the project as shown, the next step is compiling all the files in the project folder by pressing the compile button. The files need to be compiled in order for them to run once the implementation is completed. After compiling the files, the simulator can now be started by pressing the simulation button. A separate window will open, and you will need to click on the 'work' library, and will then click on the file 'proj_01_tb.v' so that the simulation may start on the correct file.



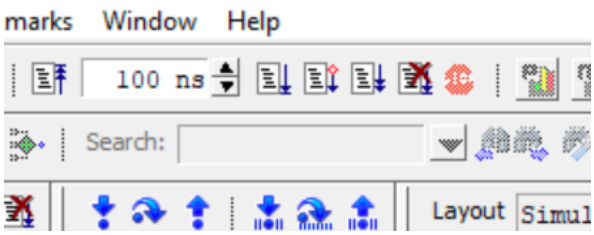
Under 'work' select 'prj_01_tb'

To be able to observe the signal waveforms as the simulator is running, right click on the file 'prj_01_tb' and click the option 'Add Wave'.



Click '**Add Wave'** to add signal waveforms

To run, reset, or specify how long the simulation should last, the following image will display the buttons for the functions.



The tools to run the simulation.

III. The ALU Requirements

The ALU (Arithmetic and Logic Unit) provides the basic function of a computer. The ALU is mainly a circuit that can handle the logical operations of the processor like logical AND, NOR, OR, the basic arithmetic operations like addition, subtraction, multiplication, bitwise AND, bitwise OR, and even the logical right and left shifts.

Mathematical and logical functions are often broken down in terms of operands and operations by the ALU. For the purpose of this project, 'op1_reg' and 'op2_reg' are used to define the operands, while 'oprn_reg' is utilized to define the operation. The ALU of this project will take two 32-bit registers 'op1_reg' and 'op2_reg', and the operation that will be applied in a 6-bit register 'oprn_reg' also know as the op-code, and will save the result in a 32-bit register called 'r_net'.

IV. Designing and Implementing the ALU

The implementation of the ALU is relatively easy to program the necessary functions using ModelSim. The following section will focus on the design of the ALU using Verilog Hardware Language.

A. Design

The ALU design is very simple, as it is designed to take in two operators, and an operation and return the result as the output. For this project, the ALU that is built is a 32-bit processor, and each operand is 32-bit, the operand is 6-bit, and the output is 32-bit. All the nine operations that are implemented in this program are defined with 'ALU_WIDTH_OPRN'h0X where the X is used to define the specific operation. In the project, 'h03' is multiplication, while the operants are called 'op1' and 'op2' respectively, and the resulting output is referred to as 'golden'.

B. The Operations that are handled by the ALU

For the purpose of this project, there are nine operations that are handled by the ALU. The following operations are listed along with their implementations:

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

V. Testing Strategy and Testing Implementation

As the implementation is completed in the file called 'alu.v', the testing of the project is accomplished in the other file called 'prj_01_tb.v', which will provide the test cases and

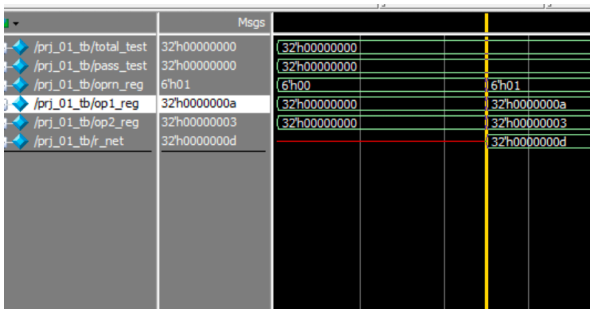
record the results that are necessary to make sure that the design of the ALU is done correctly. By following the instructions from Section II to compile and simulate the program, it is possible to make sure that the ALU was implemented correctly. The following tests are done in which the results named ‘golden’ from the file ‘prj_01_tb.v’ are compared against the results that are collected from ‘alu.v’ methods.

A. Addition

oprn_reg = `ALU_OPRN_WIDTH’h01

op1_reg = 10

op2_reg = 3



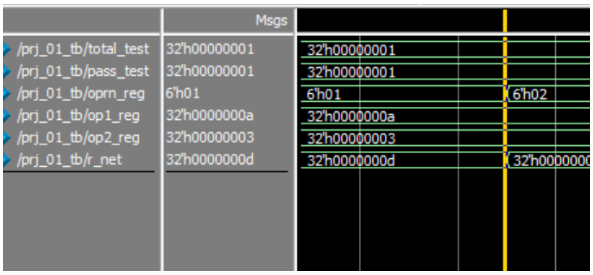
The Waveform for Addition

B. Subtraction

oprn_reg = ‘ALU_OPRN_WIDTH’h02

op1_reg = 10

op2_reg = 3



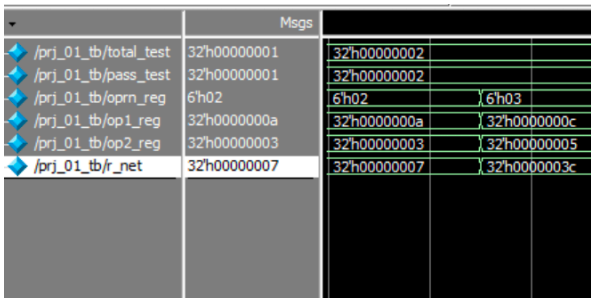
The Waveform for Subtraction

C. Multiplication

oprn_reg = `ALU_OPRN_WIDTH’h03

op1_reg = 12

op2_reg = 5

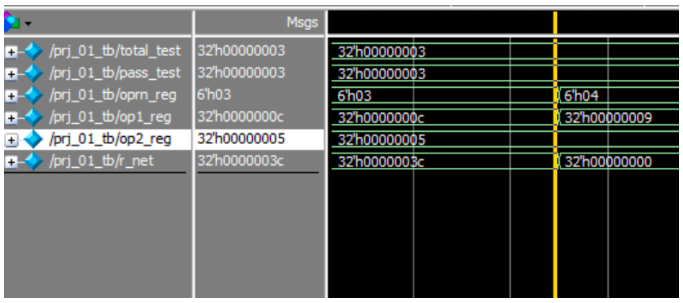


D. Logical Shift Right

oprn_reg = `ALU_OPRN_WIDTH’h04

op1_reg = 9

op2_reg = 5



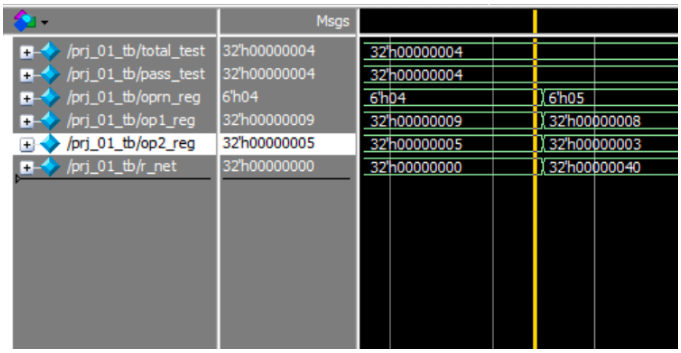
The Waveform for Logical Shift Right

E. Logical Shift Left

oprn_reg = `ALU_OPRN_WIDTH’h05

op1_reg = 8

op2_reg = 3



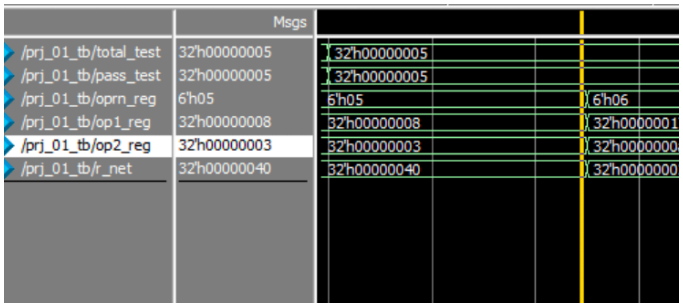
The Waveform for Logical Shift Left

F. Bitwise AND

oprn_reg = `ALU_OPRN_WIDTH’h06

op1_reg = 23

op2_reg = 10



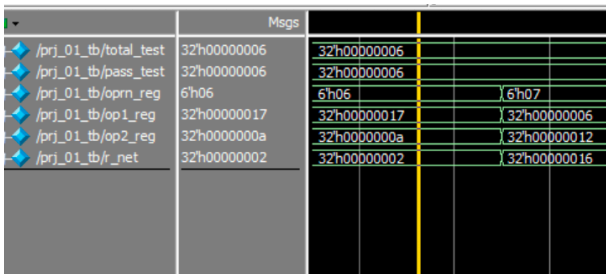
The Waveform for Bitwise AND

G. Bitwise OR

oprn_reg = `ALU_OPRN_WIDTH'h07

op1_reg = 6

op2_reg = 18



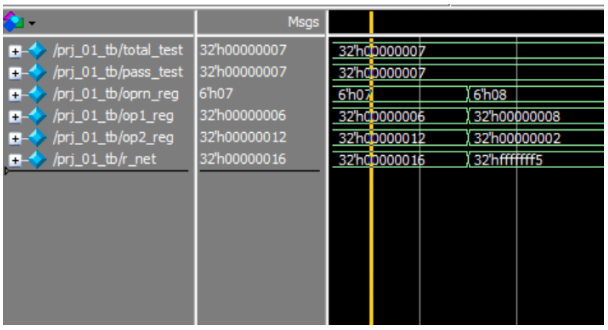
The Waveform for Bitwise OR

H. Bitwise NOR

oprn_reg = `ALU_OPRN_WIDTH'h08

op1_reg = 8

op2_reg = 2



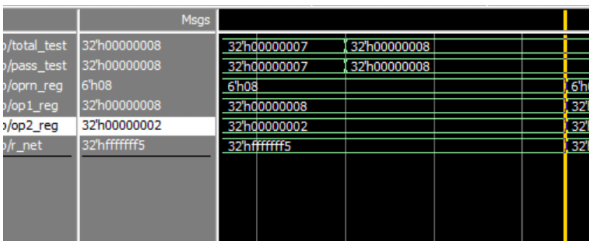
The Waveform for Bitwise NOR

I. Set Less Than

oprn_reg = `ALU_OPRN_WIDTH'h09

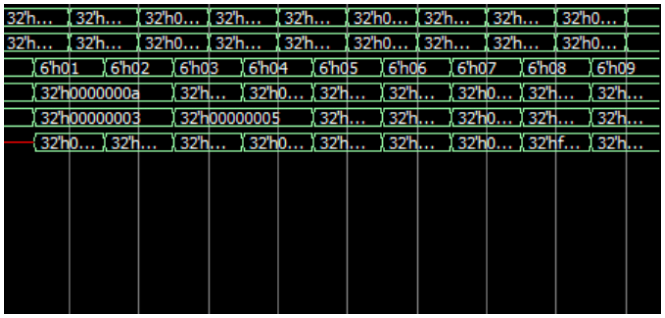
op1_reg = 6

op2_reg = 14



The Waveform for Set Less Than

While the ALU test cases can be displayed by the wavelengths, it can also be represented in text format.



The entire Waveform of the simulation

```
# [TEST] 10 + 3 = 13 , got 13 ... [PASSED]
# [TEST] 10 - 3 = 7 , got 7 ... [PASSED]
# [TEST] 12 * 5 = 60 , got 60 ... [PASSED]
# [TEST] 9 >> 5 = 0 , got 0 ... [PASSED]
# [TEST] 8 << 3 = 64 , got 64 ... [PASSED]
# [TEST] 23 && 10 = 2 , got 2 ... [PASSED]
# [TEST] 6 || 18 = 22 , got 22 ... [PASSED]
# [TEST] 8 ~| 2 = 4294967285 , got 4294967285 ... [PASSED]
# [TEST] 6 < 14 = 1 , got 1 ... [PASSED]
#
#       Total number of tests           9
#       Total number of pass            9
#
# ** Note: $stop    : C:/Users/azael/Downloads/prj_01/prj_01
#    Time: 95 ns  Iteration: 0  Instance: /prj_01_tb
```

The text output of the simulation

VI. CONCLUSION

In short, after completing the project, I learned how to use VirtualBox in order to create a virtual machine that is able to boot up Windows. I was also able to install the ModelSim simulator and was able to create a project using Verilog Hardware language. This has enabled me to create a 32-bit ALU, and was able to implement the nine operations defined in the 'CS147 DV'. To conclude, I was able to learn how to use ModelSim, and run the simulations as well as observe the waveforms that were produced.