Behavioral Modeling of ALU

YUNLIN XIE

San Jose State University [yunlin.xie@sjsu.edu](mailto:yunlin.xie@sjsu.edu)

***Abstract*—In this report, the steps of installing and setting up a digital simulation tool are introduced. This report also shows how to implement and test Arithmetic and Logic Unit (ALU), as well as how to simulate and observe waveforms using ALU test bench.**

***Keywords—digital simulation tool, ALU, simulate, waveforms, ALU test bench*.**

1. **INTRODUCTION**

We can use digital simulation tools to design, implement, and simulate a digital circuit, this behavior involves the use of a Hardware Design Language (HDL). We will use Verilog as our HDL in this project, and the simulator we use is ModelSim. The goal of this project is to fulfill the following objectives:

1. Successfully install and setup our digital simulation tool-- ModelSim.
2. Implement Arithmetic and Logic Unit (ALU) module using HDL.
3. Test our implemented ALU by implementing test bench code.
4. Use our implemented ALU test bench code to simulate and observe the output waveforms.

# TOOL INSTALLATION AND SETUP

We use ModelSim as our digital simulation tool, because it is free and it is also the one recommended by our professor. We follow the steps below to install and set up our simulation tool in Windows machines.

1. Open the following link and click the “Download Student Edition” button found in the page: [https://www.mentor.com/company/higher\_ed/model sim-student-edition](https://www.mentor.com/company/higher_ed/modelsim-student-edition)
2. Fill out the required information and click “Submit”, then you will receive an email sent by [“sales\_in](mailto:sales_info@mentor.com)f[o@mentor.com](mailto:sales_info@mentor.com)” with the download link for installation (.exe) file.
3. Click the link in the email and download the .exe file in your preferred location.
4. Double the .exe file you download in your computer, run it and complete all the steps to finish the installation.
5. When the installation is completed, a browser window will open with the License Request form. Fill out the required information and submit the form.
6. Then you will receive an email with an attached file named “student\_license.dat”. Save the file to the top level installation directory for ModelSim PE Student

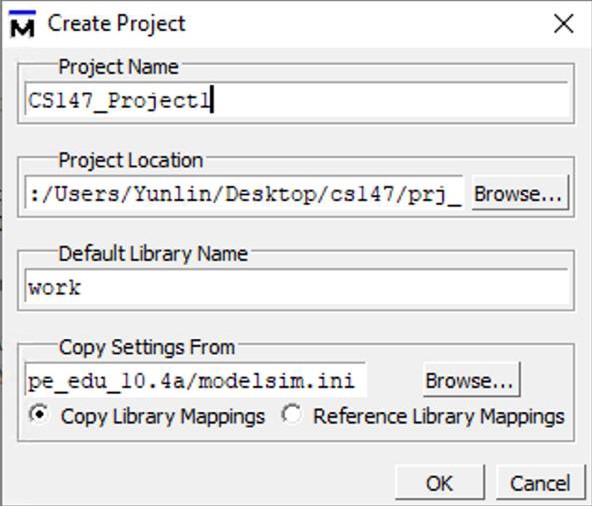
Edition. This is the directory that contains that sub- directory “win32pe\_edu”. Do not change the file name.

1. Now the ModelSim is successfully installed and setup in your computer.

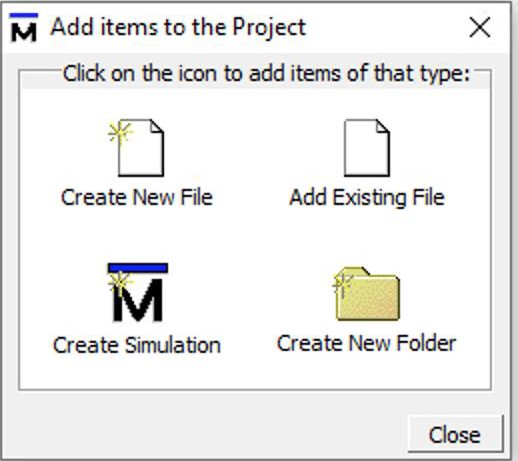
# PROJECT CREATION

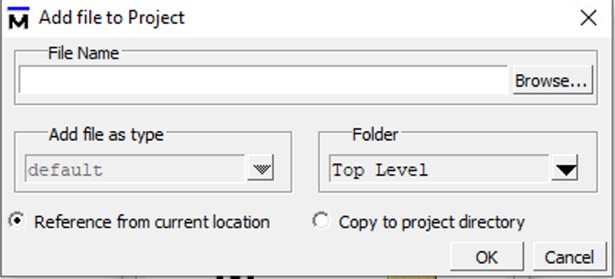
With the ModelSim being pre-installed, we are ready to simulate ALU using Verilog code. Following steps can guide you to create our project:

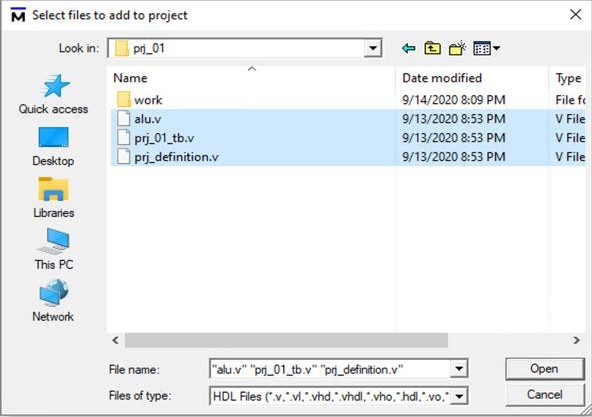
1. Download the “prj\_01.zip” file from project page in course website, and extract it to get three “.v” files.
   * The “alu.v” file which needs to be implemented, includes the module code for ALU.
   * The “prj\_01\_tb.v” file which needs to be implemented, is used for testing ALU with specific instances.
   * The “prj\_definition.v” contains the definitions and specific bits used in creating the ALU module, which should not be modified.
2. Open ModelSim, click “file” button on the top left. Navigate to “File->New->Project”. Choose a preferred project name and location, then click “OK”. (see image below for reference)



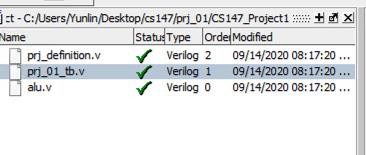
1. After done step 2, we can see an image as the first one showing below. We need to click “Add Existing File”. Then another window will pop out, which looks like the second image. Click “Browse” in the second image and another window will pop out, which looks like the third image showing below. Find the three files you download in step 1. Select all three files and click “Open” in image 3. In mage 2 click “OK”. Finally, “Close” image 3. (see image below for reference)



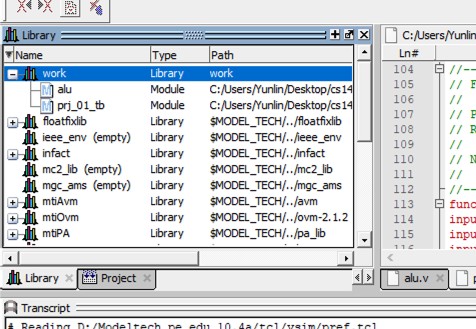




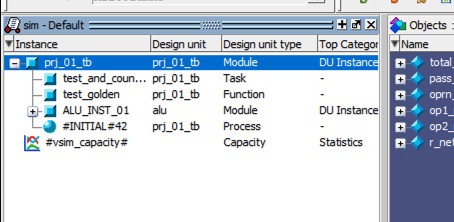
1. After adding three files and editing them correctly, click the “Compile” on the top line, and choose “Compile All” in the drop-down menu. A green hook mark will show near each filename if the compilation is successful. (see image below for reference)



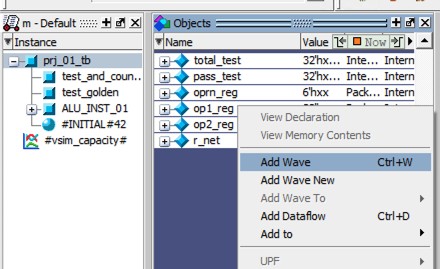
1. Click “Library” tab on the left bottom (where you can switch between Library and Project). Click “work” folder and then double-click the “pro\_01tb.v” file in the drop-down list. (see image below for reference)



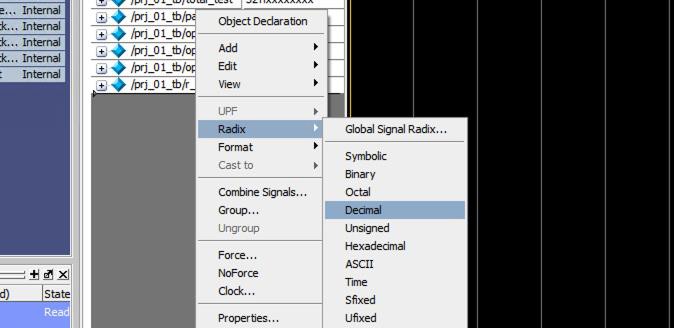
1. After doing step 5, a window named “Sim” appears. Double click “prj\_01\_tb”. (see image below for reference)



1. An “Object” window will appear. Select all items showing in the window (press shift button and then select), and right click them. Select “Add Wave” in the menu after right clicking. (see image below for reference)



1. A new window will pop out. The left side has all inputs, outputs, and operations, while the right side is a black background. We use this window to see signal waveforms.
   * To change value type shown in maps, select all (press shift button and then select) values and right click them. In the pop-out menu, select “Radix” and then select the type you want to display.
   * Click “Run All” button shown in the second image below, the new are able to see the waveforms.



# REQUIREMENTS OF ALU

We need to implement a fully functional Arithmetic & Logic Unit (ALU) module using HDL. The ALU should be able to support arithmetic and logic function for “CS147DV”

instruction set. The “CS147DV” instruction set is specifically designed for course CS147. The bit format of instructions in the “CS147DV” contains 6-bit “OpCode” and 6-bit “funct” to help identify a specific operation (e.g. OpCode 0x00 / funct 0x025 is “or” operation).

For our ALU in this project, we are supposed to fulfill the following operations:

|  |  |  |
| --- | --- | --- |
| **Name (Mnemonic)** | **Operation** | **Operatio n Code**  **(h means in hex)** |
| Addition (add) | R[rd]=R[rs]+R[rt] | h01 |
| Subtraction (sub) | R[rd]=R[rs]-R[rt] | h02 |
| Multiplication  (mul) | R[rd]=R[rs]\*R[rt] | h03 |
| Shift right logical (srl) | R[rd]=R[rs]>>shamt | h04 |
| Shift left logical (sll) | R[rd]=R[rs]<<shamt | h05 |
| Bitwise AND (and) | R[rd]=R[rs] & R[rt] | h06 |
| Bitwise OR (or) | R[rd]=R[rs] | R[rt] | h07 |
| Bitwise NOR (nor) | R[rd]=~(R[rs] | R[rt]) | h08 |
| Set less than (slt) | R[rd]=(R[rs]<R[rt]) ?  1 : 0 | h09 |

# DESIGN AND IMPLEMENTATION OF ALU

## Design Strategy:

Basic ALU modules contain three input ports: A, B, and

F. Basic ALU modules also contain one output port: R. A and B are 32-bit input ports for inputting numbers. F is a 6-bit input port for inputting operator. R is a 32-bit output for outputting result.

In the Verilog code files for this project: A is replaced by op1, B is replaced by op2, R is replaced by result, and F is replaced by oprn.

In this project, the input parameters for op1 and op2 replace the source register in CS147DV instruction set; while the output parameter for result will replace the destination registers. More detailed code and explanation will be shown below, and you can also reference to the section IV.

## Implementation:

1. Addition

In the addition operation, we add op1 and op2 to get the outcome and store it in result. The condition for addition is when oprn equals h01. See the image at the bottom of this section.

1. Subtraction

In the subtraction operation, we subtract op2 from op1 to get the outcome and store it in result. The condition for subtraction is when oprn equals h02. See the image at the bottom of this section.

1. Multiplication

In the multiplication operation, we multiply op1 by op2 to get the outcome and store it in result. The condition for multiplication is when oprn equals h03. See the image at the bottom of this section.

1. Shift Right Logical

In the shift right logical operation, we shift op1 to the right using “>>” operator, the number of bits we shift is the input of op2. We get and store the outcome in result. The condition for shift right logical is when oprn equals h04. See the image at the bottom of this section.

1. Shift Left Logical

In the Shift Left Logical operation, we shift op1 to the left using “<<” operator, the number of bits we shift is the input of op2. We get and store the outcome in result. The condition for shift left logical is when oprn equals h05. See the image at the bottom of this section.

1. Bitwise AND

AND operation uses input op1, input op2, and “&” operator. We get and store the outcome in result. The condition for AND is when oprn equals h06. See the image at the bottom of this section.

1. Bitwise OR

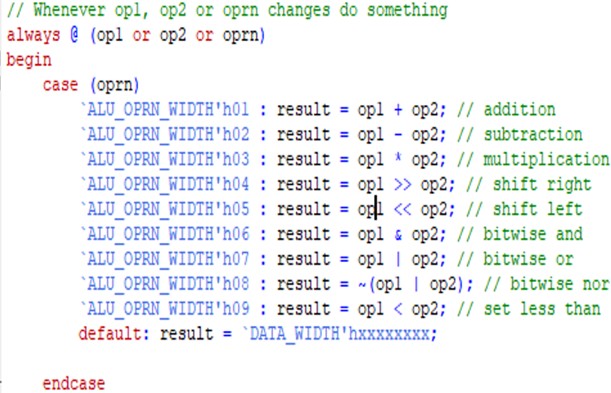
OR operation uses input op1, input op2, and “|” operator. We get and store the outcome in result. The condition for OR is when oprn equals h07. See the image at the bottom of this section.

1. Bitwise NOR

NOR operation uses input op1, input op2, and “~” operator. We take bitwise OR of op1 and op2, and store the outcome in result, then inverse it. The condition for NOR is when oprn equals h08. See the image at the bottom of this section.

1. Set Less Than

Set Less Than operation uses input op1, input op2, and “<” operator. If op1 is less than op2, result will be 1, otherwise it will be 0. We get and store the outcome in result. The condition for addition is when oprn equals h09. See the image at the bottom of this section.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 22 | nor | 1 | 0 | 1111111111111111  1111111111111110 |
| 23 | nor | 0 | 0 | max |
| 24 | nor | 6 | 9 | 1111111111111111  1111111111110000 |
| 25 | less than | 4 | 5 | 1 |
| 26 | less than | 1 | 1 | 0 |
| 27 | less than | 2 | 1 | 0 |

# TEST STRATEGY AND TEST IMPLEMENTATION

After successfully implementing all the code, we can test the module using our designed cases and observe waveforms on test bench. You can refer to section III on how to use test bench to observe waveforms and how to change digital types.

I create three cases for each operation, which is total 27 cases. Firstly, I will show all the tests I designed in part A. Secondly, I will explain nine among them, one for each operation. Lastly, I will show the test results.

## All Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case #** | **Operation** | **op1** | **op2** | **Expected Result** |
| 1 | addition | 15 | 3 | 18 |
| 2 | addition | 0 | 10 | 10 |
| 3 | addition | 17 | 0 | 17 |
| 4 | subtraction | 15 | 5 | 10 |
| 5 | subtraction | 0 | 0 | 0 |
| 6 | subtraction | 5 | 5 | 0 |
| 7 | multiplication | 2 | 7 | 14 |
| 8 | multiplication | 0 | 3 | 0 |
| 9 | multiplication | 5 | 0 | 0 |
| 10 | right shift | 4 | 1 | 2 |
| 11 | right shift | 4 | 2 | 1 |
| 12 | right shift | 4 | 3 | 0 |
| 13 | left shift | 3 | 1 | 6 |
| 14 | left shift | 3 | 2 | 12 |
| 15 | left shift | 3 | 3 | 24 |
| 16 | and | 0 | 3 | 0 |
| 17 | and | 1 | 2 | 0 |
| 18 | and | 2 | 3 | 2 |
| 19 | or | 0 | 0 | 0 |
| 20 | or | 0 | 7 | 7 |
| 21 | or | 6 | 9 | 15 |

**Test Case Examples**

1. **Test case #1 for “addition” operation**

op1 = 15;

op2 = 3;

oprn = 1;

We will do the following operation to get the result: result = op1 + op2, which is 15 + 3 = 18. So, result equals 18, same as expected result.

## Test case #4 for “subtraction” operation

op1 = 15;

op2 = 5;

oprn = 2;

We will do the following operation to get the result: result = op1 - op2, which is 15 - 5 = 10. So, result equals 10, same as expected result.

## Test case #7 for “multiplication” operation

op1 = 2;

op2 = 7;

oprn = 3;

We will do the following operation to get the result: result = op1 \* op2, which is 2 \* 7 = 14. So, result equals 14, same as expected result.

## Test case #12 for “right shift” operation

op1 = 4 (0000 0100);

op2 = 3;

oprn = 4;

We will do the following operation to get the result: result = op1 >> op2. We right shift each bit in op1 for 3 bits, we will get 0. So, result equals 0, same as expected result.

## Test case #15 for “left shift” operation

op1 = 3 (0000 0011);

op2 = 3;

oprn = 5;

We will do the following operation to get the result: result = op1 << op2. We left shift each bit in op1 for

3 bits, we will get 0001 1000, which is 24. So, result equals 24, same as expected result.

## Test case #18 for “and” operation

op1 = 2 (0000 0010);

op2 = 3 (0000 0011);

oprn = 6;

We will do the following operation to get the result: result = op1 & op2. Only when current bits in both numbers are “1”, we get “1” in result. We will get 0000 0010, which is 2. So, result equals 2, same as expected result.

## Test case #21 for “or” operation

op1 = 6 (0110);

op2 = 9 (1001);

oprn = 7;

We will do the following operation to get the result: result = op1 | op2. We get “1” for a resulting bit if there is a “1” in current bit of either of these two numbers. For this case, we will get 1111, which is 15 in decimal format. So, result equals 15, same as our expected result.

## Test case #24 for “nor” operation

op1 = 6(0000 0000 0000 0000 0000 0000 0000 0110);

op2 = 9(0000 0000 0000 0000 0000 0000 0000 1001);

oprn = 8;

We will do the following operation to get the result: result = ~ (op1 | op2). Op1 is 6, its binary form is shown above. Op2 is 9, its binary form is also shown above. We take “or” of them, which is 15, in binary is 0000 0000 0000 0000 0000 0000 0000 1111. Then we

need to inverse it, we can get the binary number as 1111 1111 1111 1111 1111 1111 1111 0000, which

equals our expected result.

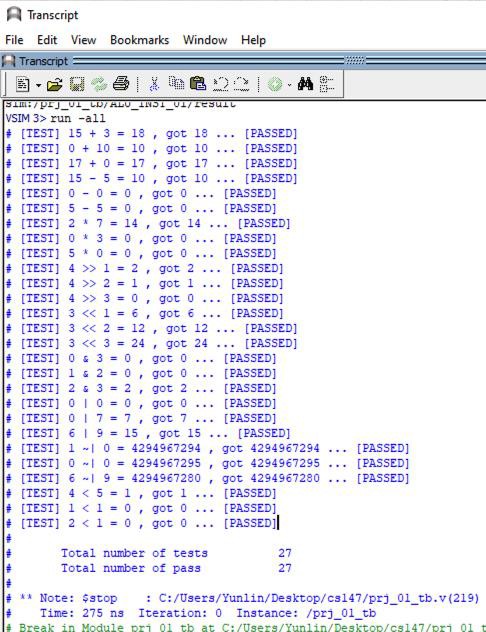
## Test case #25 for “less than” operation

op1 = 4;

op2 = 5;

oprn = 9;

We will do the following operation to get the result: result = op1 < op2 ? 1 : 0. In our case, 4 is less than 5, so the result is 1, matches our expected value.



# CONCLUSION

ModelSim was installed and set up successfully. Project creation process is also very smooth, where we use the starting code given by the Professor. All files are implemented based on the specified requirements. The test results are all the same as expected results, which means all tests are passed. Therefore, it is reasonable to conclude that the project is overall successful.

After doing project, I learned how to install, setup and use ModelSim. In addition, I am able to understand simple VerilogHDL, as well as use it to design, implement and test a simple ALU module.

# REFERRENCES

1. M. Morris Mano, and M. D. Ciletti, Digital Design, 4th ed., December 15, 2006.
2. S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, 2nd ed., Prentice Hall PTR, February 21, 2003..
3. K. Patra, Assignment -- Project 1 (Behavioral Modeling of ALU), CS- 147 Sec 01, Canvas, unpublish

## Test Result

We can see the test results in Transcript window. All the test results for my test is shown below in the image, which is the same as my test design. The image tells us all tests are passed.