

Introduction to Arithmetic Logic Unit (ALU)

Andy Luong
CS 147, San Jose State University
andy.v.luong@sjsu.edu

1. Introduction

The simulation used to test the 32-bit Arithmetic Logic Unit (ALU) is called ModelSim. Model Sim was made by Mentor Graphics, and is free for students from specific universities, which in this case includes San Jose State University. In order, the process of the report and using the ALU is as follows: Installation and setup, requirements for ALU, design and implementation of ALU, test strategy and test implementation, and conclusion. Verilog HDL will also be implemented to use ALU and will result in using various test benches and waveforms.

2. Installation And Setup

A) Installing the Modelsim Simulation Tool

Many simulation tools require payment but some companies offer free tools towards students. Modelsim has a free version for students provided by the Graphic Mentor Company. In this case, a time limit license is not required, and instead is based off of the user's account and their affiliated school and email address.

Steps To Obtaining Modelsim For Students.

1. Click the following link to start:
<https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html>
2. Click "Download ModelSim*-Intel® FPGA edition software"
3. Create an account or sign in to download
4. Select edition "Lite", the latest select release, and your corresponding Operating System
5. Under "Individual Files", click "ModelSim - Intel FPGA Edition (includes Starter Edition)"

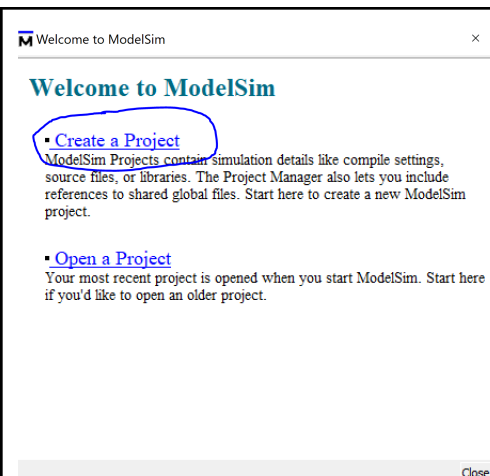
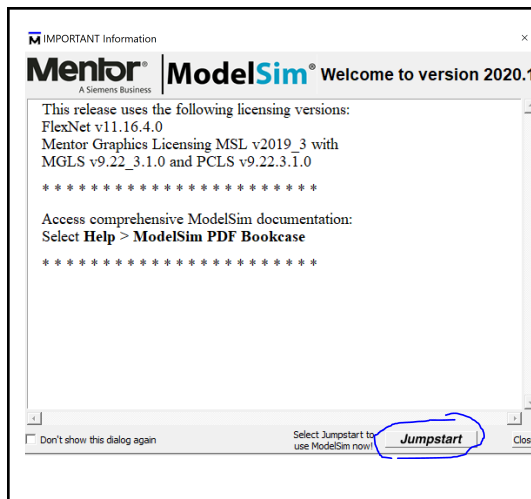
Steps For Installing Modelsim

1. Next and ModelSim - Intel FPGA Starter Edition -License is not required.
2. Next to "Do you accept this license?" click "I accept the agreement"
3. Choose installation directory or stick with default installation location
4. Ready to Install, Next, and Finish

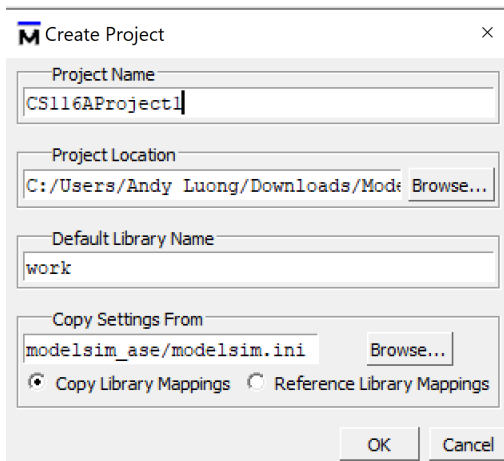
B) Creating Simulation Projects

Steps For Setting Up Modelsim

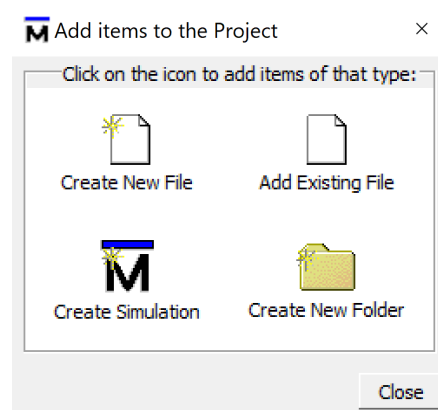
1. You will be introduced to this window when you open up the program. Press Jumpstart to start the project process.	2. Next, click Create a project.
--	----------------------------------



3. Give the project any name and pick the desired project location. Note the default location might not work. An alternative is to use the Downloads Directory and create a folder.

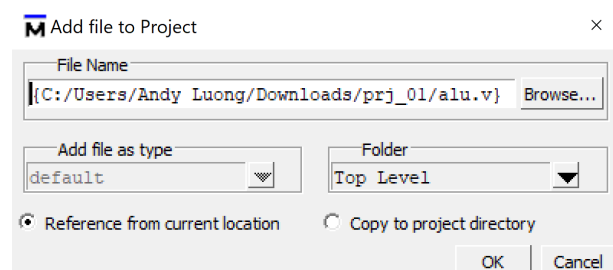


4. Under Add items to the Project, click Add Existing File.

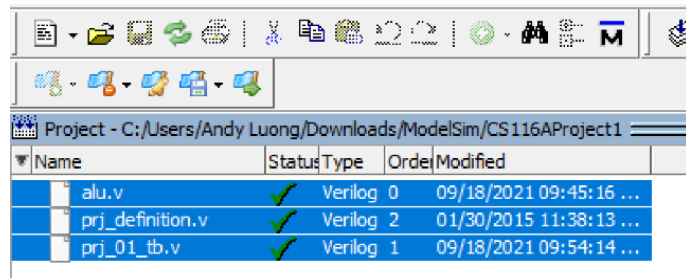


5. Obtain the ALU Verilog code by downloading the zip file that contains "alu.v", "prj_01_tb.v", and "prj_definition".

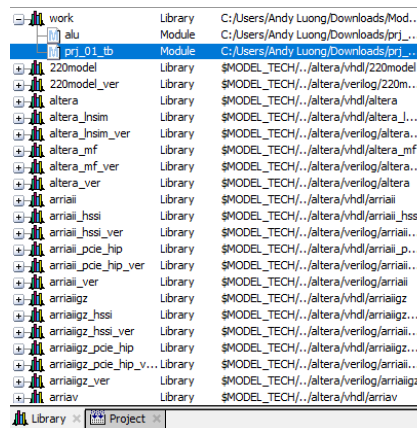
6. Browse for the downloaded ALU Verilog, click OK, and exit the Add items to the Project to successfully use the .v in the project.



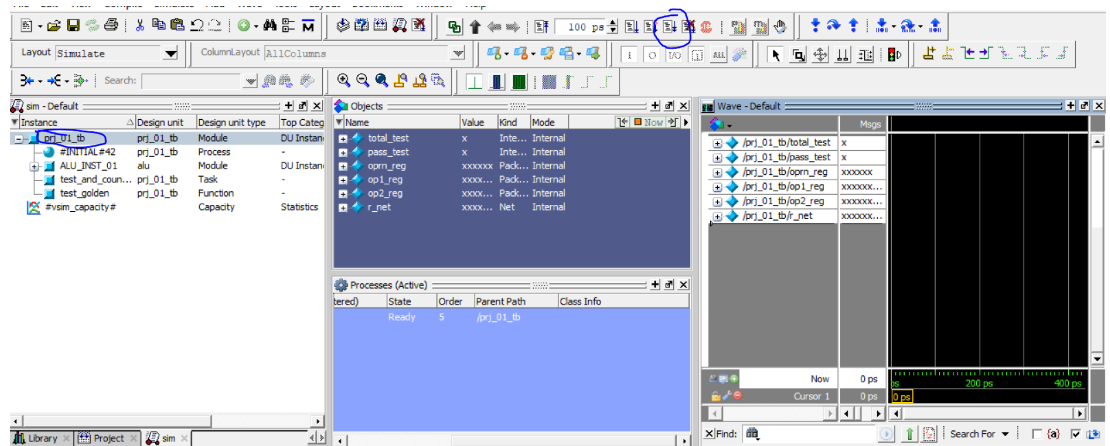
1. Once loaded there are four tabs to unlock in this order: Project, Library, Sim, and Wave
 - a. Project
 - i. Select all of the .v files, right click, and choose Compile All. There should be a green check mark under each once compiling is successful.



- ii.
- b. Library
 - i. Double click on proj_01_tb.v to enable the Object, Wave, & Sim tabs

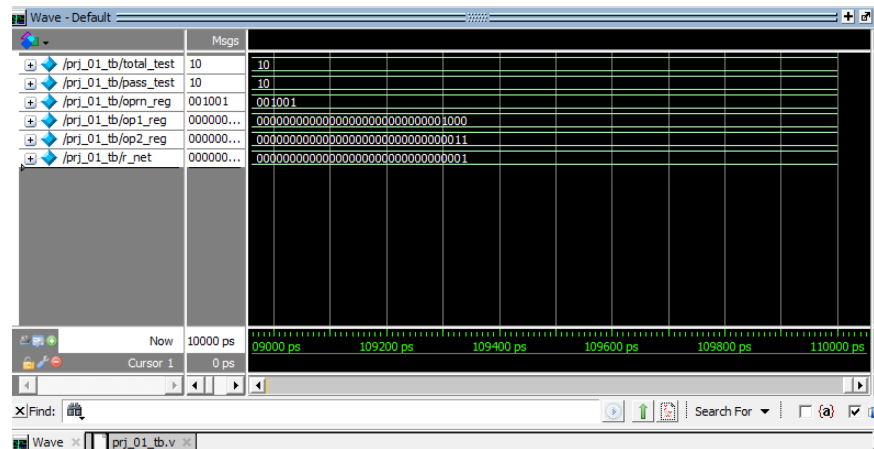


- ii.
- c. Sim
 - i. Right Click proj_01_tb and Add Wave
 - ii. Click Run - All to run the .v files



- iii.
- d. Wave
 - i. The program should output the desired result. The program is successful if the total number of tests matches the number of tests passed.

- ii. In the wave tabe, you can select the objects inside the wave, right click and in Radix, the readings of the numbers Msgs can change from the default Hexadecimal to be Decimal, Binary, Octal, and etc. In section 5, the number format of Decimal will be used for readability.



- iii.

3. Requirements For ALU

The CPU processor needs ALU to operate. The CPU's 5 cycles are Instruction Fetch (IF), Instruction (ID), Execution (EXE), Memory Access (MEM), and Write Back (WB). On-board memory divides does not have separate data lines for input and output and are bidirectional (cause back to back read write inefficiency issue) because there is a limited area to route number of electrical connections on the motherboard.

An ALU will perform arithmetic and logic operations. Addition, subtraction, multiplication, and set less than gives standard values. Shift moves the bits of the first operand and the second operand decides how many times the bit shifts. Bitwise/Logical (use interchangeably) specifically answers in binary results to show true and false rather than numerical decimal values. Here is the table of operations using HDL, arithmetic, and/or logic with name, mnemonic, operation, and operation code

Name	Mnemonic	Operation	Operation Code
Addition	add	$R[rd] = R[rs] + R[rt]$	1
Subtraction	sub	$R[rd] = R[rs] - R[rt]$	2
Multiplication	mul	$R[rd] = R[rs] * R[rt]$	3
Shift right logical	srl	$R[rd] = R[rs] \gg R[rt]$	4
Shift left logical	sll	$R[rd] = R[rs] \ll R[rt]$	5
Bitwise AND	and	$R[rd] = R[rs] \& R[rt]$	6
Bitwise OR	or	$R[rd] = R[rs] R[rt]$	7

Bitwise NOR	not	$R[rd] = \sim(R[rs] \mid R[rt])$	8
Set less than	sit	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	9

In this class, the CS147DV instruction set (ISA) has 3 types of instructions, R, I & J. The addressability of this instruction set's memory model is word addressable. During write, data goes from the processor to the memory. During read, data goes from the memory to the processor. As a result, information flows bidirectional. Since an ALU has 12 operations, the minimum bus width or BUS bits of the OPRN operation control bus is 4 ($\log_2(12)$).

The Control Unit gives the ALU an opcode for decoding. The registers of the ALU take in the data and the opcode will tell the register how to direct this data. The program converts an operation into multiple operations.

The Logical Block tells the process of an ALU. Op1 and op2 are the operands (32-bit) that can use a maximum of 4 GB of addressable memory and produce 32-bit addresses. They go through oprn (6-bit) to get decoded, and then it outputs the "result".

4. Design and Implementation of ALU

Verilog is a Hardware Description Language that implements ALU. Modelsim describes the operation and data flow of Verilog with RTL level and Gate level circuit implementation.

Design wise, an ALU has 3 inputs (A or op1, B or op2, F or oprn) and an output (R or result). A, B, and R are 32 bit while F is 6 bit.

In between the inputs and outputs are a set of operations that works with F the oprn. The set of operations of ALU are as followed: Addition, Subtraction, Multiplication, Shift Right, Shift Left, Bitwise AND, Bitwise OR, Bitwise NOR, and Set Less Than. 6 means 6 bits of the ALU opcode and "01h" is the hex number of addition, "02h" is subtraction, and so on in Verilog code. here is the code to continue the rest of the process. If the input is unknown, "X" will be the value after the operation telling that there is a failure.

Here is what each set of operations and what they do. Note, each starts with case(oprn) before the Verilog code in order to use opcode. Also, 'ALU_OPRN_WIDTH' is the same as 6.

alu.v

1. `ALU_OPRN_WIDTH'h01 : result = op1 + op2; //addition
2. `ALU_OPRN_WIDTH'h02 : result = op1 - op2; //subtraction
3. `ALU_OPRN_WIDTH'h03 : result = op1 * op2; //multiplication
4. `ALU_OPRN_WIDTH'h04 : result = op1 & op2; //AND
5. `ALU_OPRN_WIDTH'h05 : result = op1 | op2; //OR
6. `ALU_OPRN_WIDTH'h06 : result = ~(op1 | op2); //NOR

7. `ALU_OPRN_WIDTH'h07 : result = (op1 | op2)?1:0; //Less Than
8. `ALU_OPRN_WIDTH'h08 : result = op1 << op2; //Shift Left
9. `ALU_OPRN_WIDTH'h09 : result = op1 >> op2; //Shift Right

prj_01_tb.v

```
`ALU_OPRN_WIDTH'h01 : begin $write("+ "); golden = op1 + op2; end //Addition
`ALU_OPRN_WIDTH'h02 : begin $write("- "); golden = op1 - op2; end //Subtraction
`ALU_OPRN_WIDTH'h03 : begin $write("* "); golden = op1 * op2; end //Multiplication
`ALU_OPRN_WIDTH'h05 : begin $write("& "); golden = op1 & op2; end //Logical AND
`ALU_OPRN_WIDTH'h04 : begin $write("|| "); golden = op1 | op2; end //Logical OR
`ALU_OPRN_WIDTH'h06 : begin $write("~| "); golden = ~(op1 | op2); end // Logical NOR
`ALU_OPRN_WIDTH'h07 : begin $write("< "); golden = op1 < op2; end //Set Less Than
`ALU_OPRN_WIDTH'h08 : begin $write("<< "); golden = op1 << op2; end // Shift Left
`ALU_OPRN_WIDTH'h09 : begin $write(">> "); golden = op1 >> op2; end // Shift Right
```

5. Test Strategy and Test Implementation

Users can input some test cases to see if the Verilog code has been successfully implemented and test results confirm if the operations are correct. A test bench code is when the ALU operations are being checked using simulation and validation code. Test cases are the inputs and test results are the output. The given code gave two presettest test cases for addictions (will use the latter) and one case for subtract. It ended 110,000 ps.

The '?' represents the inputs that will be different from each test scenario.

```
#5 op1_reg=?;
   op2_reg=?;
   oprn_reg=`ALU_OPRN_WIDTH'h0?;
#5 test_and_count(total_test, pass_test,
                  test_golden(op1_reg,op2_reg,oprn_reg,r_net));
```

A. Test Cases & Results

Addition (+) op1 = 15; op2 = 5; oprn = 01; result = 20	Subtraction (-) op1 = 15; op2 = 5; oprn = 02; result = 10	Multiplication (*) op1 = 7; op2 = 3; oprn = 03; result = 21
Bitwise AND (&) op1 = 8; op2 = 13; oprn = 04; result = 8	Bitwise OR () op1 = 12; op2 = 4; oprn = 05; result = 12	Bitwise NOR (~()) op1 = 4; op2 = 1; oprn = 06 result = -6
Set Less Than (<) op1 = 3; op2 = 6; oprn = 07;	Shift Left Logical (<<) op1 = 5; op2 = 2; oprn = 08;	Shift Right Logical (>>) op1 = 11; op2 = 4; oprn = 09;

result = 1	result = 20	result = 0
------------	-------------	------------

B. Picture Representation

	Msgs	
+ /prj_01_tb/total_test	0	0
+ /prj_01_tb/pass_test	0	0
+ /prj_01_tb/oprn_reg	0	0
+ /prj_01_tb/op1_reg	0	0
+ /prj_01_tb/op2_reg	0	0
+ /prj_01_tb/r_net	x	

Addition

	Msgs	
+ /prj_01_tb/total_test	1	1
+ /prj_01_tb/pass_test	1	1
+ /prj_01_tb/oprn_reg	1	1
+ /prj_01_tb/op1_reg	15	15
+ /prj_01_tb/op2_reg	3	3
+ /prj_01_tb/r_net	18	18

Subtraction

	Msgs	
+ /prj_01_tb/total_test	2	2
+ /prj_01_tb/pass_test	2	2
+ /prj_01_tb/oprn_reg	2	2
+ /prj_01_tb/op1_reg	15	15
+ /prj_01_tb/op2_reg	5	5
+ /prj_01_tb/r_net	10	10

Addition 2

	Msgs	
+ /prj_01_tb/total_test	3	3
+ /prj_01_tb/pass_test	3	3
+ /prj_01_tb/oprn_reg	1	1
+ /prj_01_tb/op1_reg	15	15
+ /prj_01_tb/op2_reg	5	5
+ /prj_01_tb/r_net	20	20

Multiplication

	Msgs	
+ /prj_01_tb/total_test	4	4
+ /prj_01_tb/pass_test	4	4
+ /prj_01_tb/oprn_reg	3	3
+ /prj_01_tb/op1_reg	7	7
+ /prj_01_tb/op2_reg	3	3
+ /prj_01_tb/r_net	21	21

Bitwise AND

	Msgs	
+ /prj_01_tb/total_test	5	5
+ /prj_01_tb/pass_test	5	5
+ /prj_01_tb/oprn_reg	4	4
+ /prj_01_tb/op1_reg	8	8
+ /prj_01_tb/op2_reg	13	13
+ /prj_01_tb/r_net	8	8

Bitwise OR

	Msgs	
+ /prj_01_tb/total_test	6	6
+ /prj_01_tb/pass_test	6	6
+ /prj_01_tb/oprn_reg	5	5
+ /prj_01_tb/op1_reg	12	12
+ /prj_01_tb/op2_reg	4	4
+ /prj_01_tb/r_net	12	12







Bitwise NOR

	Msgs	
+ /prj_01_tb/total_test	7	7
+ /prj_01_tb/pass_test	7	7
+ /prj_01_tb/oprn_reg	6	6
+ /prj_01_tb/op1_reg	4	4
+ /prj_01_tb/op2_reg	1	1
+ /prj_01_tb/r_net	-6	-6







Set Less Than

	Msgs	
+ /prj_01_tb/total_test	8	8
+ /prj_01_tb/pass_test	8	8
+ /prj_01_tb/oprn_reg	7	7
+ /prj_01_tb/op1_reg	3	3
+ /prj_01_tb/op2_reg	6	6
+ /prj_01_tb/r_net	1	1

Shift Left Logical

			Msgs	
+ 	/prj_01_tb/total_test	9		9
+ 	/prj_01_tb/pass_test	9		9
+ 	/prj_01_tb/oprn_reg	8		8
+ 	/prj_01_tb/op1_reg	5		5
+ 	/prj_01_tb/op2_reg	2		2
+ 	/prj_01_tb/r_net	20		20

Shift Right Logical

			Msgs	
+ 	/prj_01_tb/total_test	10		10
+ 	/prj_01_tb/pass_test	10		10
+ 	/prj_01_tb/oprn_reg	9		9
+ 	/prj_01_tb/op1_reg	11		11
+ 	/prj_01_tb/op2_reg	4		4
+ 	/prj_01_tb/r_net	0		0

6. Conclusion

This project has taught me on how to obtain Model Sim free as a student and adding in Verilog source files to create a project of designing and implementing a ALU with a set of operations in opcode. Verilog is complex code that can output waveforms using a test bench and the project has successfully passed all test cases showing that the ALU is successful.