Arithmetic & Logic Unit (ALU)

KIM DO
San Jose State University
dohoangkimpy@gmail.com

Arithmetic & Logic Unit (ALU)

Abstract: This report mainly deals the construction of 32-bit Arithmetic and Logic Unit (ALU) using ModelSim simulation tool. This project report also includes the following things.

1) Steps to install Digital Simulation tool named 'Model Sim' and its setup.
2) Implementation of Arithmetic & Logic Unit (ALU) module using Verilog HDL and its requirement.
3) Implementation of test bench code to test the ALU using HDL.
4) Simulation and observation of output waveforms of ALU using ALU test bench code

General Information

Name	Company	User for	Free?
ModelSim	Mentor Graphics	Simulation & Test	Yes(student version)

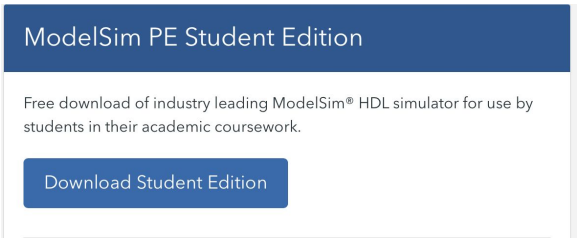
I. INSTALLATION MODELSIM, SIMULATION TOOL

There are many tools available in the market for simulation such as Xilinx ISE, VCS, Modelsim, and Altera etc. However, all tools are not free. Graphics Mentor

Company provides free Modelsim Student edition tool especially for students only.

Following are the steps that show how to get free Modelsim Student Edition.

1. Download from this link.
2. Click on "**Download Student Edition**" button



3. Open the installation file, run it and complete the installation steps.
4. Once the installation is done, a form will appear in the browser that requires filing student's name, address, phone number, email [**use .edu email]**, university name etc. Fill out the information and click finish.
5. You will receive the email from Modelsim that includes the license attachment file named '**student license.dat**'
6. Copy the '**student license.dat**' file to **C:/modeltech_pe_edu/** directory that contains the subdirectory "**win32pe_edu"**
7. Go to Desktop and run **ModelSim PE Student Edition**

II. PROJECT CREATION

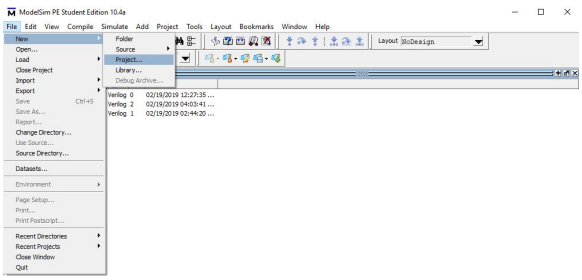
This section contains the steps of how to simulate the Verilog code for ALU and its testing using the test bench code using ModelSim tool which the existed .v files.

Follow the steps in order to simulate the Verilog code for ALU:

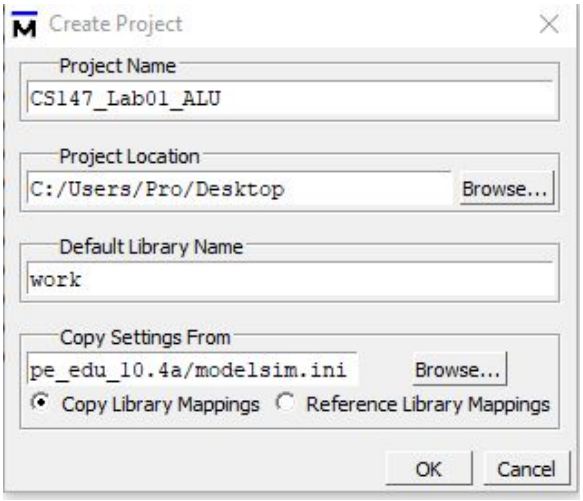
1. Download the Zip file attached with this report and
2. Extract it into the new folder. This zip file contains:

a. **alu.v** contains the code of alu
b. **prj_01_tb.v** contains the test bench for ALU
c. **Prj_definition.v** contains the definition of ALU module

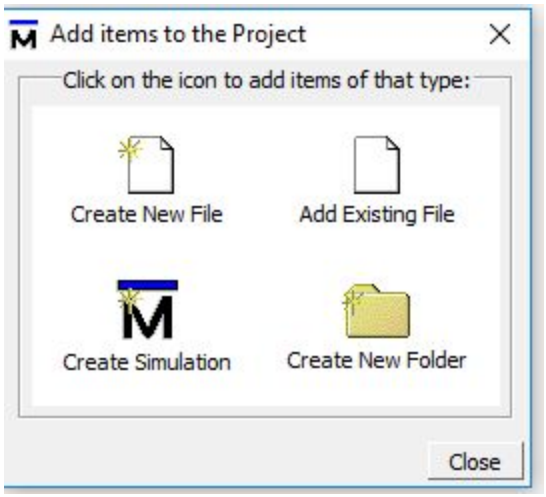
3. Open the Modelsim tool and navigate to **File->New->Project.**



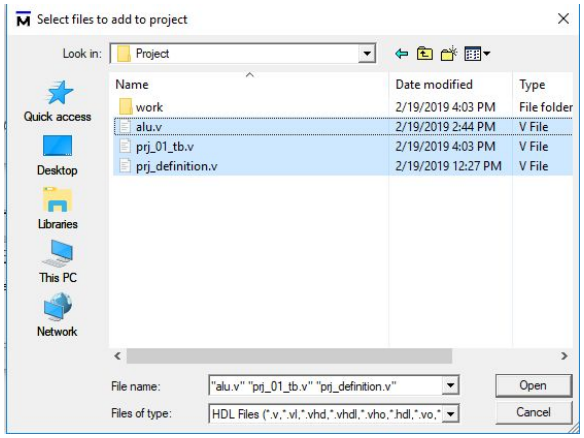
4. Name the project “CS147_Project1_ALU” with the desired directory after clicking on **Browser** button



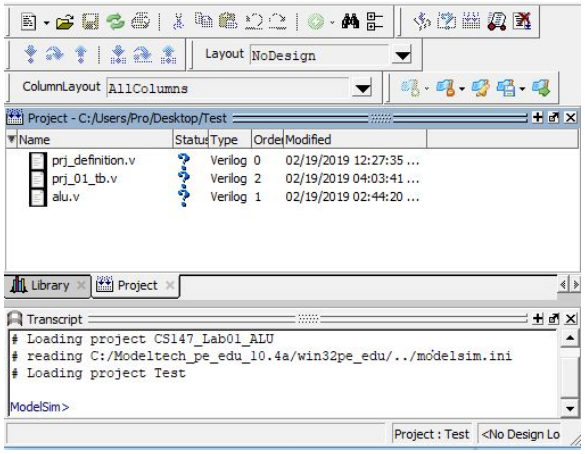
5. Click on **Add Existing File** button

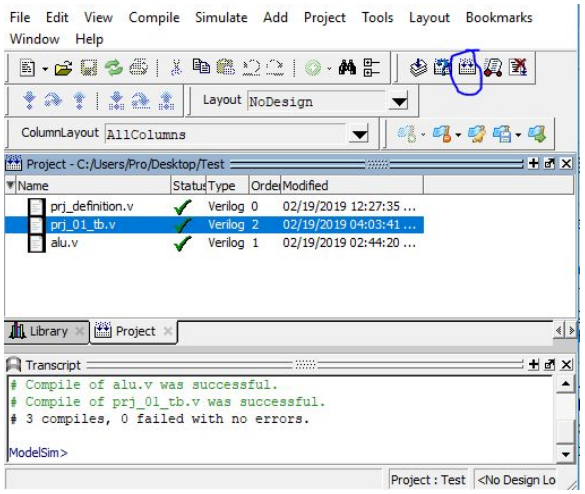


6. Click Browse to go to the right directory that contains the **three .v files**
7. Hold **Control** key while selecting all three files or Hold **Shift and down arrow**

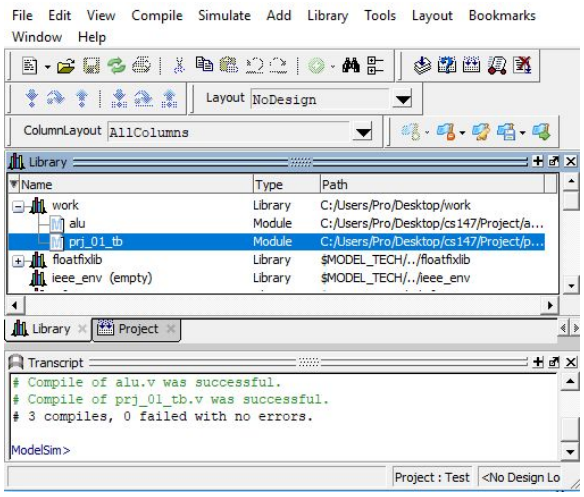


8. Make sure three files are compiled by clicking the **compile all** button under **Project** tab

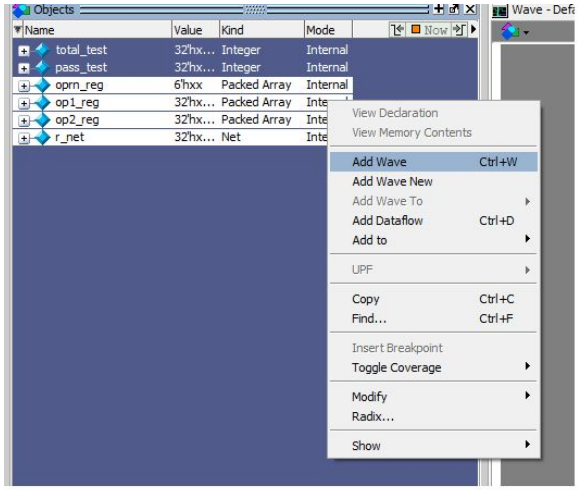




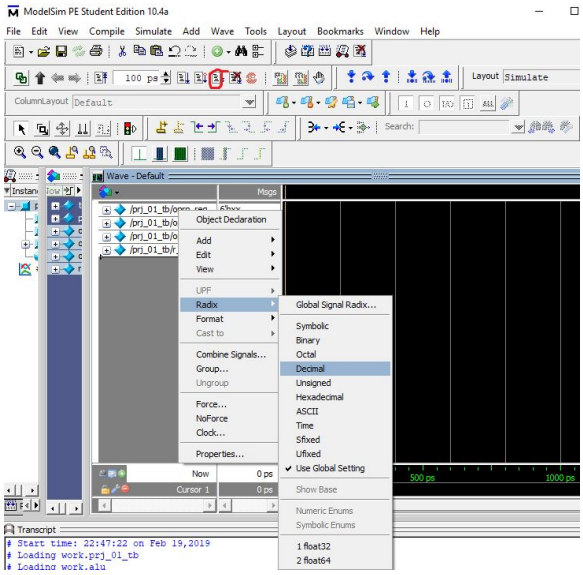
9. Click on **Library** tab at the bottom
10. Click the **+** sign on the right of the **work** tree
11. Double click on **prj_01_tb**



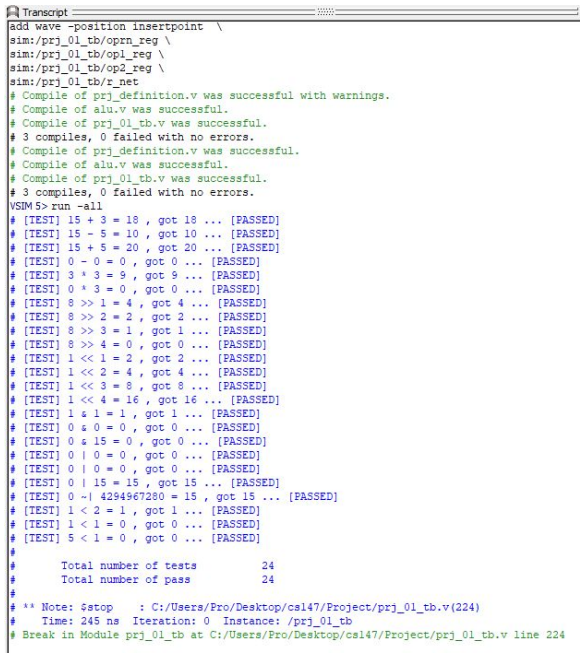
12. A **sim** tab should pop up and double click on **prj_01_tb** file
13. Hold **Control** key or **Shift** to select
    a. **Oprn_reg**
    b. **Op1_reg**
    c. **Op2_reg**
    d. **R_net**
14. Press **Control + W** or **Right Click -> Add Wave**



15. A new **Wave** tab should pop up. Select all the objects in the **wave** tab
16. **Right-click -> Radix -> Decimal**
17. Click **Run All** button

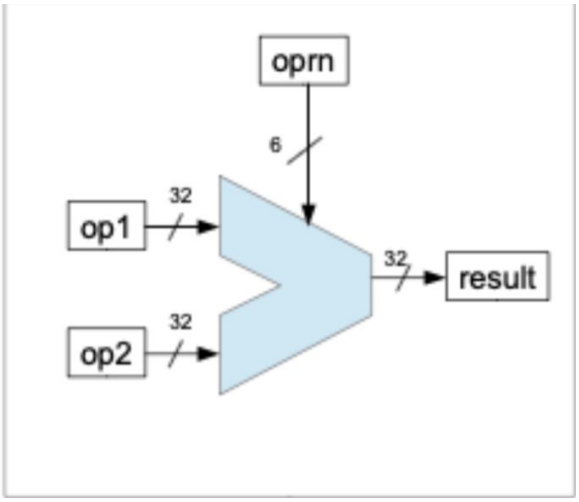


18. The output should be in the **transcript** tab



III. REQUIREMENTS FOR ALU

An arithmetic and logic unit (ALU) is a digital circuit that comprises the combination logic that implements log operations such as AND, OR, NOT and arithmetic operations such as Addition, Subtraction, Multiplication, Bitwise AND, Bitwise Or, Shift right, Shift Left, and etc. ALU is the fundamental block of the central processing unit (CPU) of a computer.



ALU block diagram notations:

- **Op1 and op2:** input to ALU as operands (32 bits each)
- **Opm:** operator method / Instruction from Control Unit known as Op-code (6 bits)
- **Result:** output (32 bits)

ALU accepts 2 inputs and one instruction and returns the return to the user.

IV. DESIGN AND IMPLEMENTATION OF ALU

Verilog is a Hardware Description Language and it is simple to implement ALU using Verilog. All we have to do is describing the operation and data flows and Verilog simulator such as ModelSim, Xilinx, and Synopsys VCS etc. performs the rest of the things like RTL level and Gate level circuit implementation that can be further used for the physical design of ALU.

For example, add instruction R[rd] = R[rs] + R[rt] shall be translated into result = op1 + op2 in HDL implementation.

Operation code	Action	Operation

h01	Addition	op1 + op2
h02	Subtraction	op1 - op2
h03	Multiplication	op1 * op2
h04	Shift Right	op1 >> op2
h05	Shift Left	op1 << op2
h06	Bitwise And	op1 AND op2
h07	Bitwise Or	op1 OR op2
h08	Bitwise Nor	op1 NOR op2
h09	Less Than	op1 < op2

1. Addition

Add operation is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h01**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h01 :
                result = op1 + op2;
```

2. Subtraction

Subtract operation is done by subtracting op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h02**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h02 :
                result = op1 + op2;
```

3. Multiplication

Multiplication operation is done by multiplication op1 and op2 input parameters and assigns the result to result in an output parameter. The operation code is **h03**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h03 :
                result = op1 * op2;
```

4. Shift Right

Shift Right is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h04**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h04 :
                result = op1 >> op2;
```

5. Shift Left

Shift left is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h05**.

```
case (oprn)
        `ALU_OPRN_WIDH'h05 :
                result = op1 << op2;
```

6. Bitwise And

Bitwise And is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h06**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h06 :
                result = op1 & op2;
```

7. Bitwise Or

Bitwise Or is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h07**.

```
case (oprn)
        `ALU_OPRN_WIDTH'h07 :
                result = op1 | op2;
```

8. Bitwise Nor

Bitwise Nor is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h08**.

case (oprn)
`ALU_OPRN_WIDTH'h08 :
result = ~(op1 | op2);

9. Less Than

Nor operation is done by adding op1 and op2 input parameters and assign the result to result in an output parameter. The operation code is **h09**.

case (oprn)
`ALU_OPRN_WIDTH'h09 :
result = op1 < op2;

```
case (oprn)
    `ALU_OPRN_WIDTH'h01 : result = op1 + op2; // a
    `ALU_OPRN_WIDTH'h02 : result = op1 - op2; // s
    `ALU_OPRN_WIDTH'h03 : result = op1 * op2; // m
    `ALU_OPRN_WIDTH'h04 : result = op1 >> op2; //
    `ALU_OPRN_WIDTH'h05 : result = op1 << op2; //
    `ALU_OPRN_WIDTH'h06 : result = op1 & op2; // a
    `ALU_OPRN_WIDTH'h07 : result = op1 | op2; // o
    `ALU_OPRN_WIDTH'h08 : result = ~(op1 | op2); /
    `ALU_OPRN_WIDTH'h09 : result = op1 < op2; // o

    // TBD: fill up rest of the operations from he
    //
    default: result = `DATA_WIDTH'hxxxxxxxx;

endcase
```

V. Test Strategy and Test Implementation

Once the Verilog implementation code of ALU is done, the module should be tested in order to check whether it is performing the operation correctly and as per the user's requirements or not.

The diagram for the simulation block is shown in figure

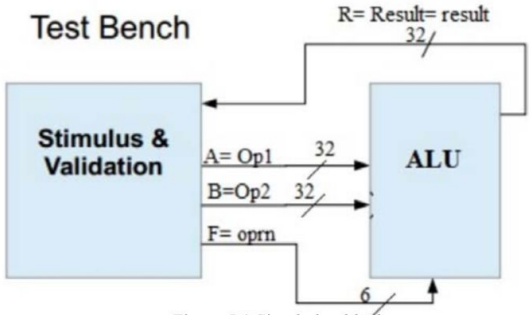


Figure 5.1 Simulation block

1. **Addition (h1)**
   a. # [TEST] 15 + 3 = 18 , got 18 ... [PASSED]
   b. # [TEST] 15 + 5 = 20 , got 20 ... [PASSED]

2. **Subtraction (h2)**
   a. # [TEST] 15 - 5 = 10 , got 10 ... [PASSED]
   b. # [TEST] 0 - 0 = 0 , got 0 ... [PASSED]

3. **Multiplication (h3)**
   a. # [TEST] 3 * 3 = 9 , got 9 ... [PASSED]
   b. # [TEST] 0 * 3 = 0 , got 0 ... [PASSED]

4. **Shift Right (h4)**
   a. # [TEST] 8 >> 1 = 4 , got 4 ... [PASSED]
   b. # [TEST] 8 >> 2 = 2 , got 2 ... [PASSED]
   c. # [TEST] 8 >> 3 = 1 , got 1 ... [PASSED]
   d. # [TEST] 8 >> 4 = 0 , got 0 ... [PASSED]

5. **Shift Left(h5)**
    a. # [TEST] 1 << 1 = 2 , got 2 ... [PASSED]
    b. # [TEST] 1 << 2 = 4 , got 4 ... [PASSED]
    c. # [TEST] 1 << 3 = 8 , got 8 ... [PASSED]
    d. # [TEST] 1 << 4 = 16 , got 16 ... [PASSED]

6. **Bitwise And(h6)**
    a. # [TEST] 1 & 1 = 1 , got 1 ... [PASSED]
    b. # [TEST] 0 & 0 = 0 , got 0 ... [PASSED]
    c. # [TEST] 0 & 15 = 0 , got 0 ... [PASSED]

7. **Bitwise OR(h7)**
    a. # [TEST] 0 | 0 = 0 , got 0 ... [PASSED]
    b. # [TEST] 0 | 0 = 0 , got 0 ... [PASSED]
    c. # [TEST] 0 | 15 = 15 , got 15 ... [PASSED]

8. **Bitwise Nor(h8)**
    a. # [TEST] 0 ~| 4294967280 = 15 , got 15 ... [PASSED]

9. **Less Than(h9)**
    a. # [TEST] 1 < 2 = 1 , got 1 ... [PASSED]
    b. # [TEST] 1 < 1 = 0 , got 0 ... [PASSED]
    c. # [TEST] 5 < 1 = 0 , got 0 ... [PASSED]

VI. CONCLUSION

From this Project, I learned the process of installing the ModelSim Tool and simulation of Verilog file on ModelSim. Furthermore, I designed and Implemented 32 bit ALU and implemented 9 different operations that can be performed by ALU and these operations are operated by the opcode. Moreover, I learned much about Verilog coding and Verilog code simulation and observation of output waveforms in ModelSim.

VII. REFERENCES

1. Digital Design (4th Edition) by M. Morris Mano and Michael D. Ciletti (Dec 15, 2006)

2. Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition By Samir Palnitkar.