Arithmetic & Logic Unit (ALU)

MOHITKUMAR PATEL
San Jose State University
mohitkumar.patel@sjsu.edu.

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

Table I.1: List of Tools Used

Name	Company	Used for	Free? (Y/N)
ModelSim	Mentor Graphics	Simulation & test	Yes (for Students only)

1. STEPS TO INSTALL THE SIMULATION TOOL 'MODELSIM'

There are many tools available in market for simulation such as Xilinx ISE, VCS, Modelsim, and Altera etc. However, all tools are not for free. Graphic 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) Open this link in your browser http://www.mentor.com/company/higher_ed/modelsim-student-edition
2) Click on 'Download Student Edition' button. This will start downloading the installation file into your computer drive.
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 filling student's name, address, phone number, email, university name etc. Fill out the information and click finish.
5) After that you will receive the email from Modelsim that includes the license attachment file named 'student_license.dat'.
6) Save the attached file with the name to the top level installation directory for ModelSim PE Student Edition (e.g., c:/modeltech_pe_edu). This is the directory that contains that sub-directory 'win32pe_edu.' Do not edit the file 'student_license.dat' in any way, or the license will not work.
7) You should now be able to run ModelSim PE Student Edition.

2. STEPS OF SIMULATION PROJECT CREATION

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

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

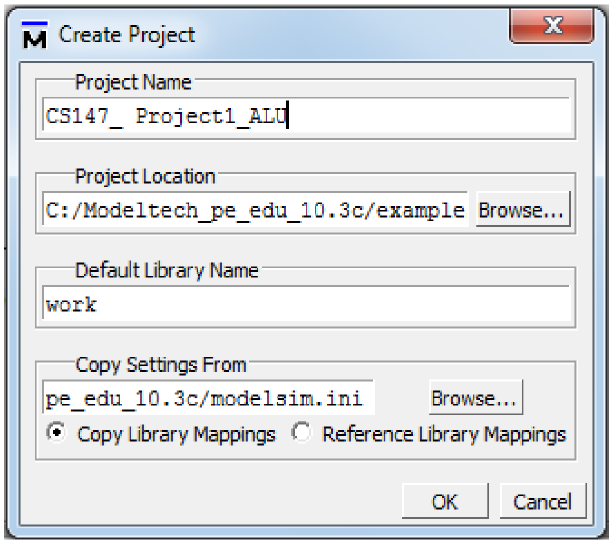
1) Download the Zip file attached with this report and extract it into the new folder. This zip file contains three Verilog files with extension '.v'.

- The first file 'alu.v' contains the module code for ALU.
- The second file 'prj_01_tb.v' is the test bench file used for testing the functions of ALU with some specific input values.
- The third file 'prj_definition' contains the specific bits and definitions that are used to create the ALU module.

2) Open the Modelsim tool and navigate to File→New→Project.

- Create Project window will open as shown in figure 4.1.

- Give a project name let say ‘CS147_Project1_ALU’ and click OK.


Create Project

Project Name: CS147_ Project1_ALU

Project Location: C:/Modeltech_pe_edu_10.3c/example

Default Library Name: work

Copy Settings From: pe_edu_10.3c/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

Figure 4.1 Create Project Window

3) After giving a project name the following window (figure 4.2) will appear and since we already have the Verilog code files, click on ‘Add Existing File’ option.

- After that one window will appear named ‘Add file to Project’ as figure 4.3 and click ‘Browse’.
- After that another window will appear named ‘select files to project’ as figure 4.4 and select those three Verilog files that are attached with this project report and click ‘Open’ button.

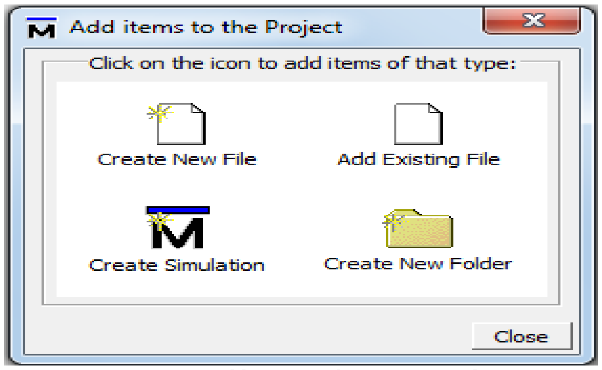
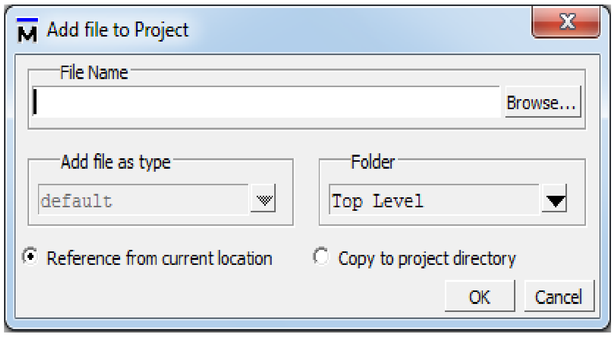


Figure 4.2 Add items to the Project window


Add file to Project

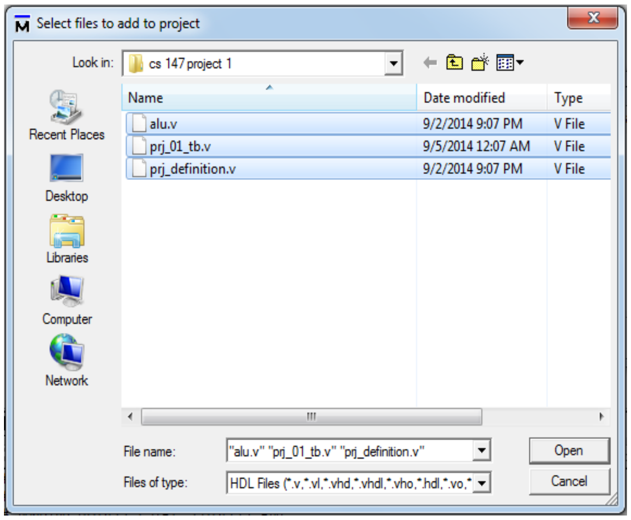
File Name:

Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

Figure 4.3 Add file to Project Window


Select files to add to project

Look in: cs 147 project 1

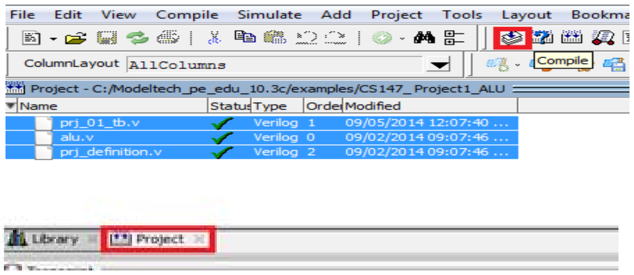
Name	Date modified	Type
alu.v	9/2/2014 9:07 PM	V File
prj_01_tb.v	9/5/2014 12:07 AM	V File
prj_definition.v	9/2/2014 9:07 PM	V File

File name: "alu.v" "prj_01_tb.v" "prj_definition.v"

Files of type: HDL Files (*.v,*.vl,*.vhd,*.vhdl,*.vho,*.hdl,*.vo,*

Figure 4.4 Select files to add to project window

4) Once all three files are added to the project, select all three files from the ‘Project’ tab and press the compile button. If the compilation is done successfully, ‘ ✓ ’ symbol will appear to each Verilog file as shown in figure 4.5.



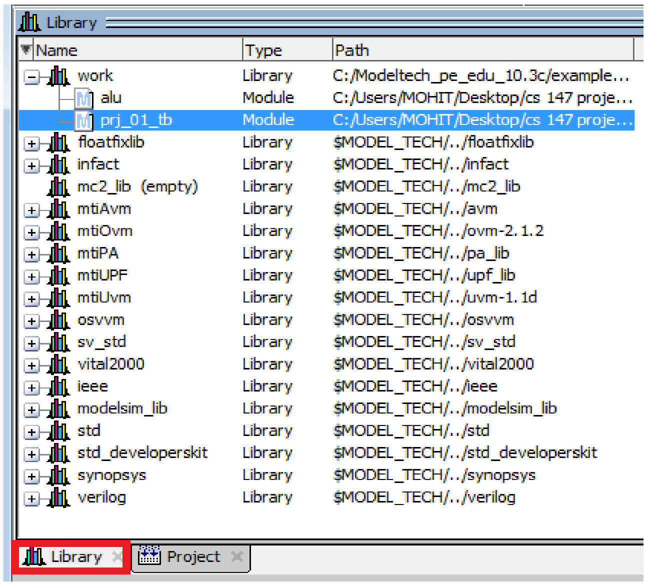
ColumnLayout AllColumns

Project - C:/Modeltech_pe_edu_10.3c/examples/CS147_ Project1_ALU

Name	Status	Type	Order	Modified
prj_01_tb.v	✓	Verilog	1	09/05/2014 12:07:40 ...
alu.v	✓	Verilog	0	09/02/2014 09:07:46 ...
prj_definition.v	✓	Verilog	2	09/02/2014 09:07:46 ...

Figure 4.5 Project window and Compilation of files

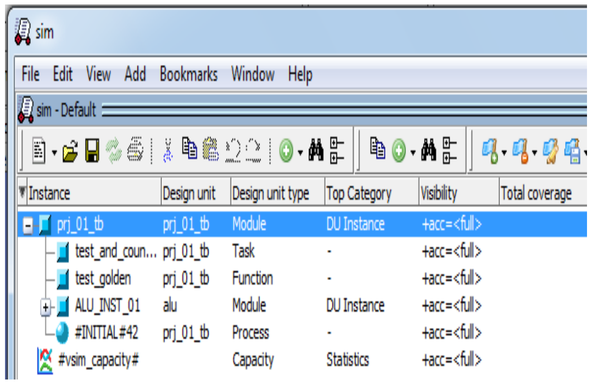
5) Then click on Library tab and navigate to work folder and double click on test bench file named as ‘proj_01tb.v shown in figure4.6.


Library

Name	Type	Path
work	Library	C:/Modeltech_pe_edu_10.3c/example...
alu	Module	C:/Users/MOHIT/Desktop/cs 147 proje...
prj_01_tb	Module	C:/Users/MOHIT/Desktop/cs 147 proje...
floatfixlib	Library	$MODEL_TECH/../floatfixlib
infact	Library	$MODEL_TECH/../infact
mc2_lib (empty)	Library	$MODEL_TECH/../mc2_lib
mtiAvm	Library	$MODEL_TECH/../avm
mtiOvm	Library	$MODEL_TECH/../ovm-2.1.2
mtiPA	Library	$MODEL_TECH/../pa_lib
mtiUPF	Library	$MODEL_TECH/../upf_lib
mtiUvm	Library	$MODEL_TECH/../uvm-1.1d
osvvm	Library	$MODEL_TECH/../osvvm
sv_std	Library	$MODEL_TECH/../sv_std
vital2000	Library	$MODEL_TECH/../vital2000
ieee	Library	$MODEL_TECH/../ieee
modelsim_lib	Library	$MODEL_TECH/../modelsim_lib
std	Library	$MODEL_TECH/../std
std_developerskit	Library	$MODEL_TECH/../std_developerskit
synopsys	Library	$MODEL_TECH/../synopsys
verilog	Library	$MODEL_TECH/../verilog

Figure 4.6 Test bench code run

6) After double clicking on test bench file, the new window will appear named 'Sim' and double click on 'prj_01_tb' file as shown in figure 4.7.

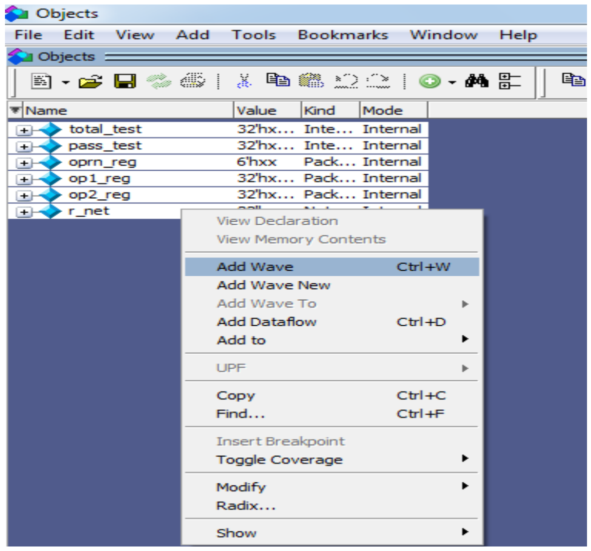


Instance	Design unit	Design unit type	Top Category	Visibility	Total coverage
prj_01_tb	prj_01_tb	Module	DU Instance	+acc=<full>	
test_and_coun...	prj_01_tb	Task	-	+acc=<full>	
test_golden	prj_01_tb	Function	-	+acc=<full>	
ALU_INST_01	alu	Module	DU Instance	+acc=<full>	
#INITIAL#42	prj_01_tb	Process	-	+acc=<full>	
#vsim_capacity#		Capacity	Statistics	+acc=<full>	

Figure 4.7 Sim window

7) After opening 'prj_01_tb' file from above step, one 'Object' window will pop up as shown in figure 4.8 that includes all ALU modules input, output and test's name.

- Select all options that appear on object window, right click them and select 'Add Wave'.

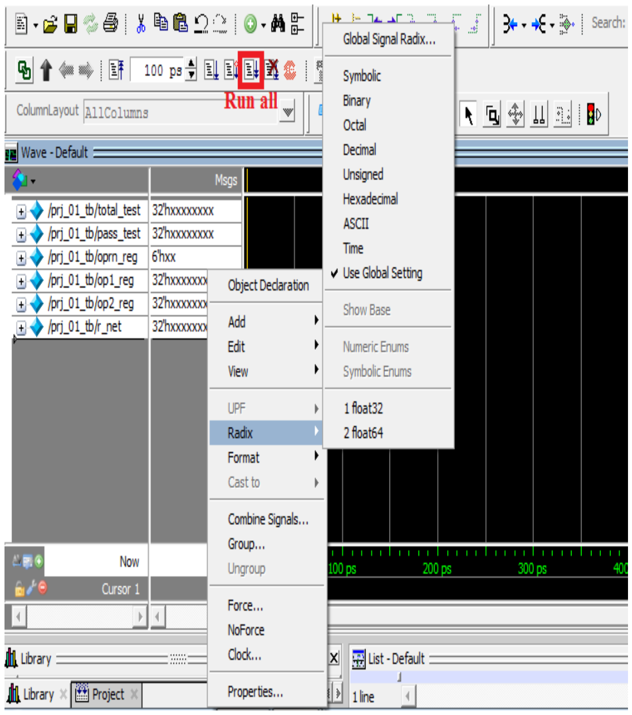


Name	Value	Kind	Mode
total_test	32'hx...	Inte...	Internal
pass_test	32'hx...	Inte...	Internal
oprn_reg	6'hxx	Pack...	Internal
op1_reg	32'hx...	Pack...	Internal
op2_reg	32'hx...	Pack...	Internal
r_net			

Figure 4.8 Object window

8) After that one new window will appeare that will have black background and on the left side all input, outputs and operation will be placed. This window is used to see the output waveforms.

- The maps values as shown in figure 4.9 will be in hex. Therefore, in order to change them to decimal, octal or in binary, select all values and right click on them and select 'Radix' option and then select Decimal, octal or whatever you want.
- Next, click on 'Run All' button as shown in figure 4.9 and you will be able to see the output waveforms.



	Msgs
/prj_01_tb/total_test	32'hxxxxxxxx
/prj_01_tb/pass_test	32'hxxxxxxxx
/prj_01_tb/oprn_reg	6'hxx
/prj_01_tb/op1_reg	32'hxxxxxxxx
/prj_01_tb/op2_reg	32'hxxxxxxxx
/prj_01_tb/r_net	32'hxxxxxxxx

Figure 4.9 Waveform window

3. REQUIREMENTS OF ALU

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

The Control Unit register (CU) transfers the opcode into the ALU for decoding. The data is already fed to the ALU through its input registers. The opcode describes how to manipulate the data.

Any complex mathematical and logical program are broken down in terms two operand operations. For example: R = (G+H-I*J) is broken down in to following series of operations by compiler.

- ✓ P1 = I * J
- ✓ P2 = H – P1
- ✓ R = G + P2

The Logical Block shown of ALU describes the ALU operation in detail.

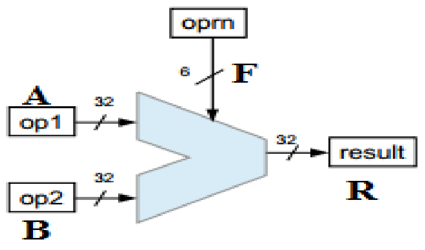


Figure 3.1 ALU Interference Diagram

In the shown ALU block diagram the notations are as follows.
A and B: the inputs to the ALU known as operands (32 bits each)
R: Output or Result (32 bits)
F: Code or Instruction from the Control Unit known as Op-code (6 bits)

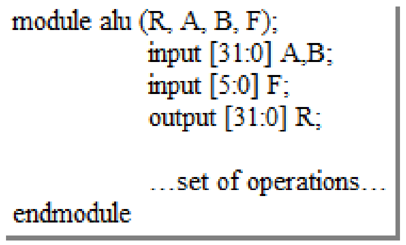
ALU takes input data from operands A and B and also fetch the op-code to perform required operation from the user. Based on the computation given as opcode, ALU provides the output data at R.

4. DESIGN AND IMPLEMENTATION OF ALU

Verilog is a Hardware Descripting 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 Moedelsim, 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 physical design of ALU.

Design Strategy:

- Treat the ALU as a module and note which are the inputs and output ports of ALU. As shown in figure 3.1, ALU modules has three input ports named as A,B and F and one output port R.
- The basic module code for ALU is as follows.



```
module alu (R, A, B, F);
            input [31:0] A,B;
            input [5:0] F;
            output [31:0] R;

            ...set of operations...
endmodule
```

- In the above code, A and B are declared as input ports, each of 32 bits and R is declared as an output port of 32 bits and F is declared as input port of 6 bits.
  Note:
  In the Verilog file attached with this project, A is named as ‘op1’, B is named as ‘op2’, R is named as ‘result’ and F is named is ‘oprn’.
- Set of operation of ALU is declared after declaring the port types and after declaration of each operation of ALU, the module ends and leaves output wire R out of the ALU module to read the output waveforms.

Set of Operations of ALU:

In this project, there are limited numbers of operations included and they are as follows.

A. Addition
B. Subtraction
C. Multiplication
D. Shift Right
E. Shift Left
F. Bitwise AND
G. Bitwise OR
H. Bitwise NOR
I. Set less than

- Each operation can be operated by the opcode that user has provided as an input to F.
  For example: if user wants that whenever there is an opcode ‘01h’, an addition should be performed between two operand registers A and B. Moreover, whenever opcode has a value of ‘02h’, a subtraction should be performed between two operand registers A and B and so on…
- These operations are described in Verilog code below.

```
always @ (op1 or op2 or oprn)
begin
        case (oprn)
        6'h01 : result = op1 + op2;
        6'h02 : result = op1 - op2;
        6'h03 : result = op1 * op2;
        6'h04 : result = op1 >> op2;
        6'h05 : result = op1 << op2;
        6'h06 : result = op1 & op2;
        6'h07 : result = op1 | op2;
        6'h08 : result = ~(op1 | op2);
        6'h09 : result = op1 < op2;
        endcase
end
```

Note:
6’h01: 6 indicate **ALU Opcode width** i.e. 6 bits and h describes hex format and 01 is the number in hex and so on…

- Whenever opcode or any input changes, ALU will performing the operation based on the provided opcode in ‘oprn’ register. When the value is h01, ALU will perform addition and so on… and if any input’s value or opcode is unknown, then ALU will provide ‘X’ values after the operation and this operation will be called operation failure.

5. 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.

ALU operation can be tested using Simulation and Validation code and it is commonly known as 'Test Bench' code. This code is in Verilog language and certain numbers are defined as inputs and outputs of are tested based on input number calculations. That's how user can validate the ALU design.

The diagram for simulation block is shown in figure 5.1.

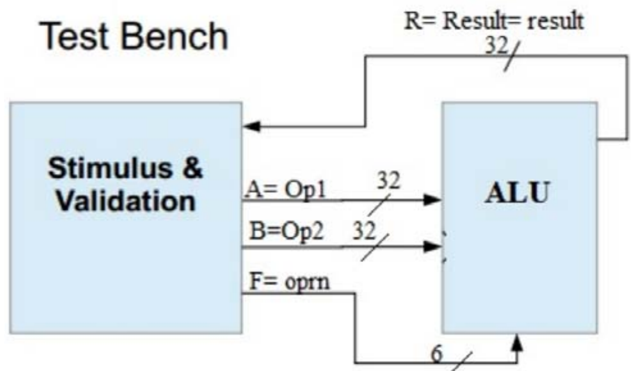
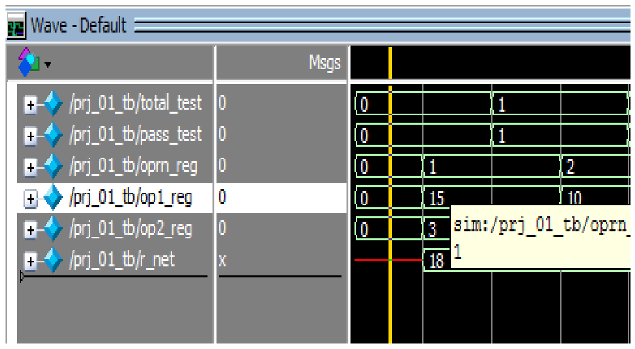


Figure 5.1 Simulation block

As shown in figure 5.1, the simulation block contains values of op1, op2 and op2 and check the output from ALU at result register.

Each test operation and its waveform are described below.

Note: if any of operands or opcode is unknown or undefined then result register will give 'x' as an output as shown in figure 5.2.

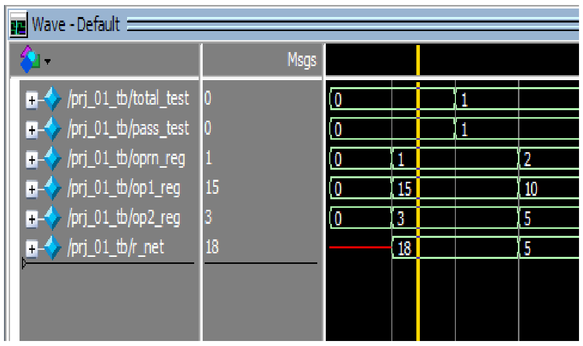


```
Wave - Default
                         Msgs
/prj_01_tb/total_test    0
/prj_01_tb/pass_test     0
/prj_01_tb/oprn_reg      0
/prj_01_tb/op1_reg       0
/prj_01_tb/op2_reg       0
/prj_01_tb/r_net         x
sim:/prj_01_tb/oprn_
1
```

Figure 5.2 Unknown result waveform

A. Addition

op1= 15;
op2= 3;
oprn= 01



```
Wave - Default
                         Msgs
/prj_01_tb/total_test    0
/prj_01_tb/pass_test     0
/prj_01_tb/oprn_reg      1
/prj_01_tb/op1_reg       15
/prj_01_tb/op2_reg       3
/prj_01_tb/r_net         18
```

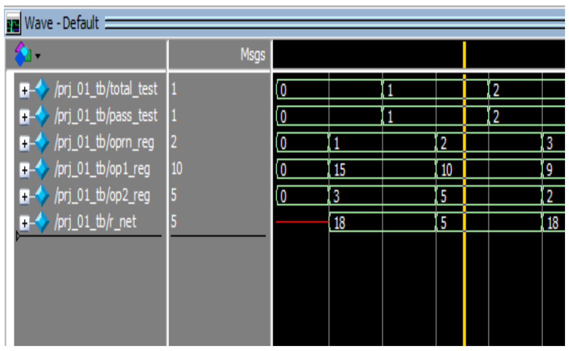
Figure 5.3 Addition output waveform

Since the oprn value is 01h, ALU will perform Addition.

result= op1 + op2 = 18

B. Subtraction

op1= 10;
op2= 5;
oprn= 02;



```
Wave - Default
                         Msgs
/prj_01_tb/total_test    1
/prj_01_tb/pass_test     1
/prj_01_tb/oprn_reg      2
/prj_01_tb/op1_reg       10
/prj_01_tb/op2_reg       5
/prj_01_tb/r_net         5
```

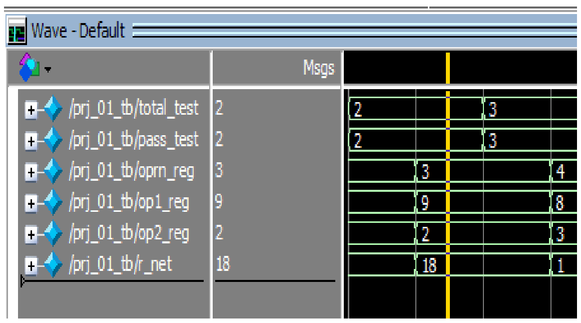
Figure 5.4 Subtraction output waveform

Since the oprn value is 02h, ALU will perform Subtraction.

result= op1 - op2 = 5

C. Multiplication

op1= 9;
op2= 2;
oprn= 03;



```
Wave - Default
                         Msgs
/prj_01_tb/total_test    2
/prj_01_tb/pass_test     2
/prj_01_tb/oprn_reg      3
/prj_01_tb/op1_reg       9
/prj_01_tb/op2_reg       2
/prj_01_tb/r_net         18
```

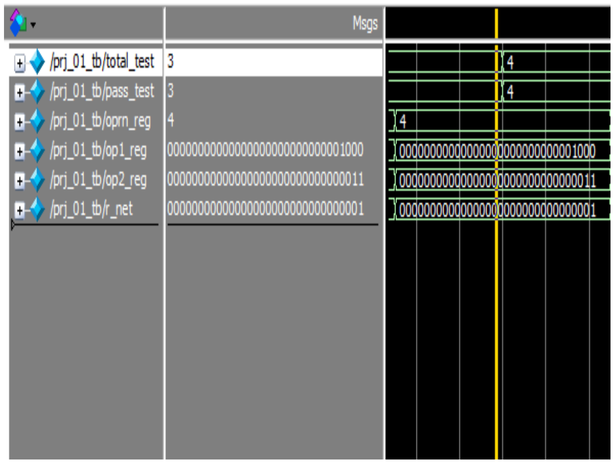
Figure 5.5 Multiplication output waveform

Since the oprn value is 03h, ALU will perform Multiplication.

result= op1 * op2 = 18

D. Shift Right

op1= 8 (1000);
op2= 3 (0011);
oprn= 04;



```
                     Msgs
/prj_01_tb/total_test  3                                  4
/prj_01_tb/pass_test   3                                  4
/prj_01_tb/oprn_reg    4                                  4
/prj_01_tb/op1_reg     00000000000000000000000000001000   00000000000000000000000000001000
/prj_01_tb/op2_reg     00000000000000000000000000000011   00000000000000000000000000000011
/prj_01_tb/r_net       00000000000000000000000000000001   00000000000000000000000000000001
```

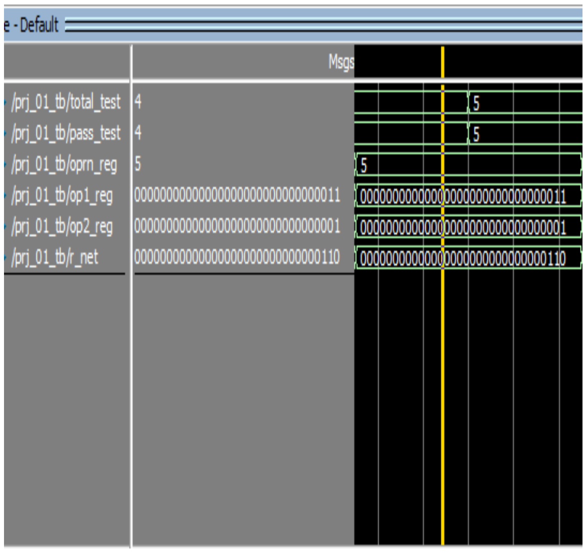
Figure 5.6 Shift Right output waveform

Since the oprn value is 04h, ALU will perform Shift Right operation.

result= op1 >> op2 = 1 (0001)

E. Shift Left

op1= 3 (0011);
op2= 1 (0001);
oprn= 05;



```
e - Default
                     Msgs
/prj_01_tb/total_test  4                                  5
/prj_01_tb/pass_test   4                                  5
/prj_01_tb/oprn_reg    5                                  5
/prj_01_tb/op1_reg     00000000000000000000000000000011   00000000000000000000000000000011
/prj_01_tb/op2_reg     00000000000000000000000000000001   00000000000000000000000000000001
/prj_01_tb/r_net       00000000000000000000000000000110   00000000000000000000000000000110
```

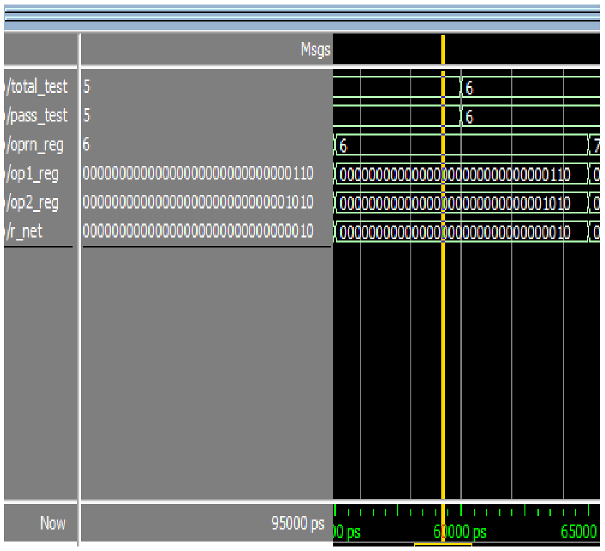
Figure 5.7 Shift Left output waveform

Since the oprn value is 05h, ALU will perform Shift Left operation.

result= op1 << op2 = 6 (0110)

F. Bitwise AND

op1= 6 (0110);
op2= 10 (1010);
oprn= 06;



```
                Msgs
/total_test  5                                  6
/pass_test   5                                  6
/oprn_reg    6                                  6                                  7
/op1_reg     00000000000000000000000000000110   00000000000000000000000000000110   0
/op2_reg     00000000000000000000000000001010   00000000000000000000000000001010   0
/r_net       00000000000000000000000000000010   00000000000000000000000000000010   0
Now          95000 ps    0 ps    60000 ps    65000
```

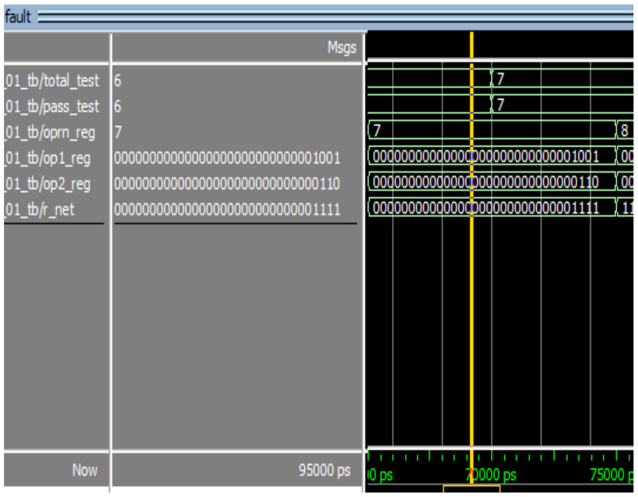
Figure 5.8 Bitwise AND output waveform

Since the oprn value is 06h, ALU will perform Bitwise AND operation.

result= op1 & op2 = 0 (0010)

G. Bitwise OR

op1= 9 (1001);
op2= 6 (0110);
oprn= 07



```
fault
                   Msgs
01_tb/total_test  6                                  7
01_tb/pass_test   6                                  7
01_tb/oprn_reg    7                                  7                                  8
01_tb/op1_reg     00000000000000000000000000001001   00000000000000000000000000001001   00
01_tb/op2_reg     00000000000000000000000000000110   00000000000000000000000000000110   00
01_tb/r_net       00000000000000000000000000001111   00000000000000000000000000001111   11
Now               95000 ps    0 ps    70000 ps    75000 p
```

Figure 5.9 Bitwise OR output waveform

Since the oprn value is 07h, ALU will perform Bitwise OR operation.

result= op1 | op2 = 15 (1111)

H. Bitwise NOR

op1=00000000000000000000000000000001;
op2=00000000000000000000000000000100;
oprn=08;

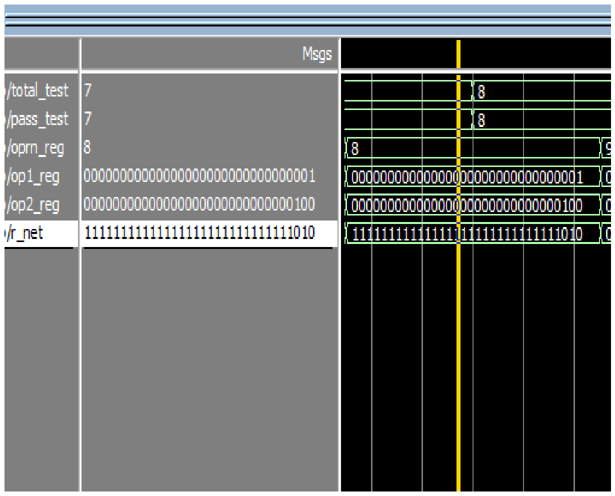

Msgs
/total_test 7
/pass_test 7
/oprn_reg 8
/op1_reg 00000000000000000000000000000001
/op2_reg 00000000000000000000000000000100
/r_net 11111111111111111111111111111010

Figure 5.9 Bitwise NOR output waveform

Since the oprn value is 08h, ALU will perform Bitwise NOR operation.
result= ~(op1 | op2)
=11111111111111111111111111111010

I. Set less than
op1= 4;
op2= 5;
oprn= 09;

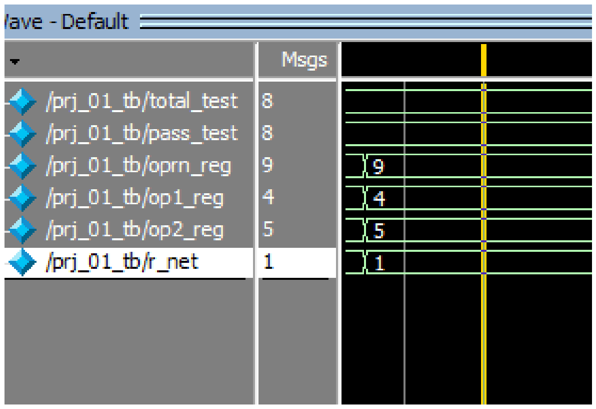
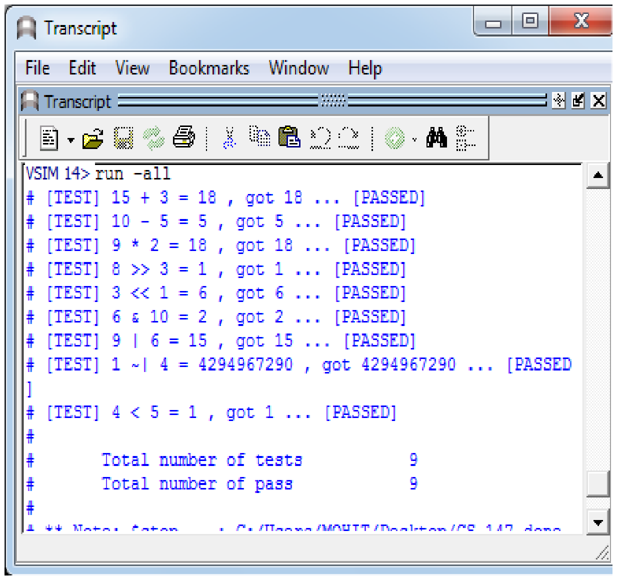


Figure 5.10 Set less than output waveform

Since the oprn value is 09h, ALU will perform Set less than operation.
result= (op1 < op2) = 1 (Since 4 less than 5 hence True)

Transcript Window

Once the simulation is performed, Modelsim also shows the operation process in Transcript window. In this window user can check the outputs of ALU, how many tests are completed and how many failed that is coded by the user in the test bench code. Moreover, if the programme is not compiled then user can check the error type and error line number of Verilog code in this window and can correct it. The transcript window is shown in figure 5.11.



```
VSIM 14> run -all
# [TEST] 15 + 3 = 18 , got 18 ... [PASSED]
# [TEST] 10 - 5 = 5 , got 5 ... [PASSED]
# [TEST] 9 * 2 = 18 , got 18 ... [PASSED]
# [TEST] 8 >> 3 = 1 , got 1 ... [PASSED]
# [TEST] 3 << 1 = 6 , got 6 ... [PASSED]
# [TEST] 6 & 10 = 2 , got 2 ... [PASSED]
# [TEST] 9 | 6 = 15 , got 15 ... [PASSED]
# [TEST] 1 ~| 4 = 4294967290 , got 4294967290 ... [PASSED
]
# [TEST] 4 < 5 = 1 , got 1 ... [PASSED]
#
#       Total number of tests           9
#       Total number of pass            9
#
# ** Note: $stop    : C:/Users/MOHIT/Desktop/CS 147 demo
```

Figure 5.11 Transcript window

Since there are 9 operation performed and 9 tests passed, the transcript window shows total number of tests= 9 and total number of pass = 9.

ALL OPERATION'S OUTPUT WAVEFORM

Figure 5.12 shows all operation's output waveforms together in decimal form.

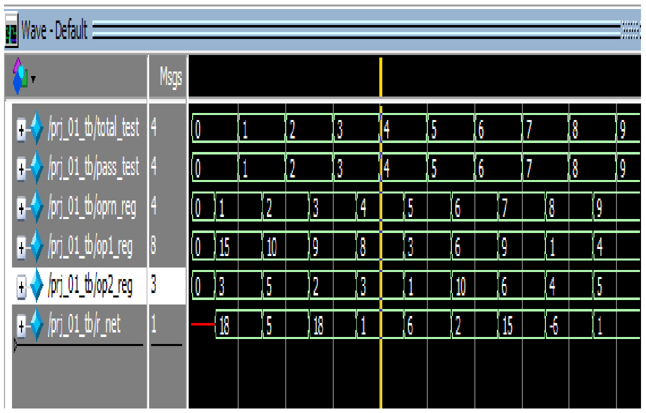


Figure 5.12 Output waveforms of all operations

6. CONCLUSION

From this Project, I learnt the process of installing 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 opcode. Moreover, I learnt much about Verilog coding and Verilog code simulation and observation of output waveforms on ModelSim.

7. 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.