Arithemetic & Logic Unit(ALU)

SHUANG PAN

San Jose State University

shuang.pan@sjsu.edu

***Abstract*—The first project is to use ModelSim Student Edition to implement ALU. The followings 4 steps are the objectives of this project.**

1) To install digital simulation tool and setup
2) To implement arithmetic & logic unit (ALU) module using HDL Verilog.
3) To implement testing of implemented ALU using HDL
4) To simulate and observe signal waveforms using ALU test bunch

The ModelSim is a product of Mentor Graphics. I will use the free student edition to implement ALU and write test cases for the ALU by simulation.

1. Install the simulation tool

The appplication called ModelSim which is from Graphic Mentor Company, and the version I used is for students. The followings are how to install ModelSim.

1) Put the link at any broswer

https://www.mentor.com/company/higher_ed/modelsim-student-edition

2) Click on the Download Student Edition and fill out some personal information. We will get an email with a link from them.

3) Click on that link and it will start to download.

4) When it is done, click on the file and it will start to install. When it is finished, a form will pop up from your default broswer. It is about your name, phone number, company and so on.

5) After submitting it, we will receive an email again which let us download the license using ModelSim. It is named “student_license.dat”.

6) Download it and move it to a directory called “Modeltech_pe_edu_10.4a” (it is in C drive for me).

7) After that, we are ready to use it for this project.

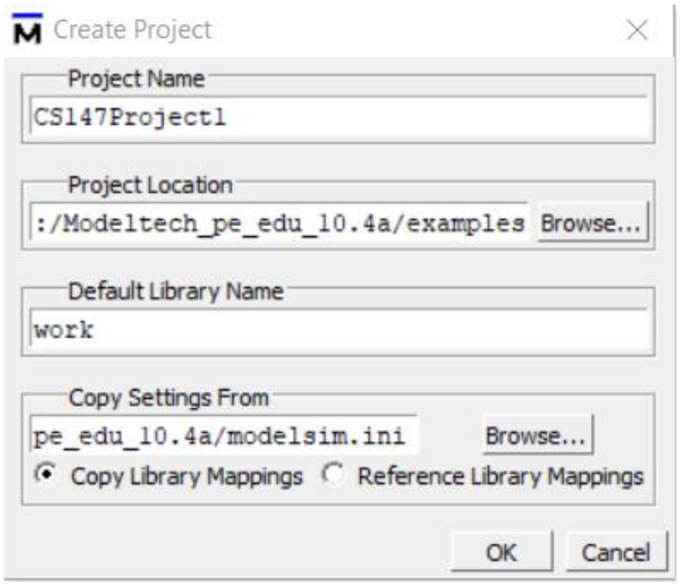
2. Simulation project creation

The step is creating a project and adding files. It also includes the way to simulate the ALU implementation using bench test and waveform.

1) Get the zip file which is provided by the professor from Canvas. Decompress it and there are three .v files in it.

- prj_definition.v is a file which contains some data initialization.
- alu.v is a file that is for us to implement ALU.
- prj_01_tb.v is to let us add testing codes to whether the functions we write work correctly.

2) Open the ModelSim and navigate the cursor to file at the top left corner and click on it. Put cursor on the new and click on the project button. Then it will pop up the window at the following figure 2.1



Create Project

Project Name: CS147Project1

Project Location: :/Modeltech_pe_edu_10.4a/examples

Default Library Name: work

Copy Settings From: pe_edu_10.4a/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

Figure 2.1 Create Project

Give a project name called CS147Project1 and click on the *OK* button.

3) After that, it will pop up a window called Add items to the Project (Figure 2.2). Because the files required for this project have been downloaded from Canvas. We just click on *Add Existing File*.

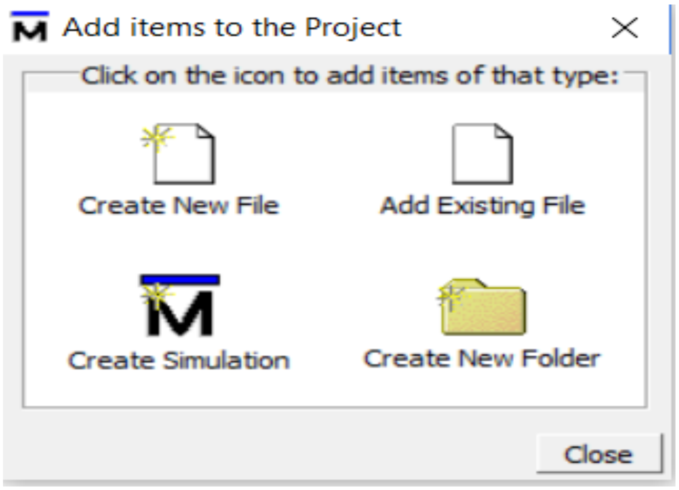
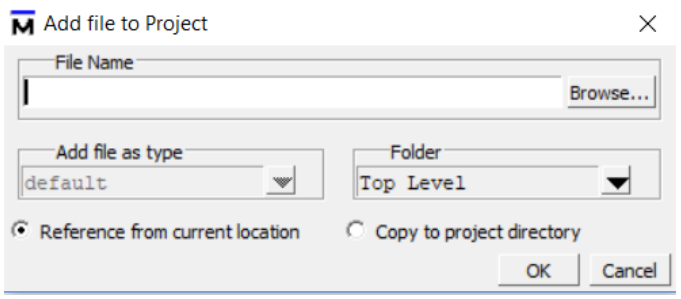


Figure 2.2 Add items

4) After clicking on the Add Existing File button, it will pop up another window called Add file to Project (Figure 2.3). Then click on the *Browser*, it will pop up another window called *Select files to add to project* (Figure 2.4). Then select three files we got from the Canvas. Click on the *Open* button.



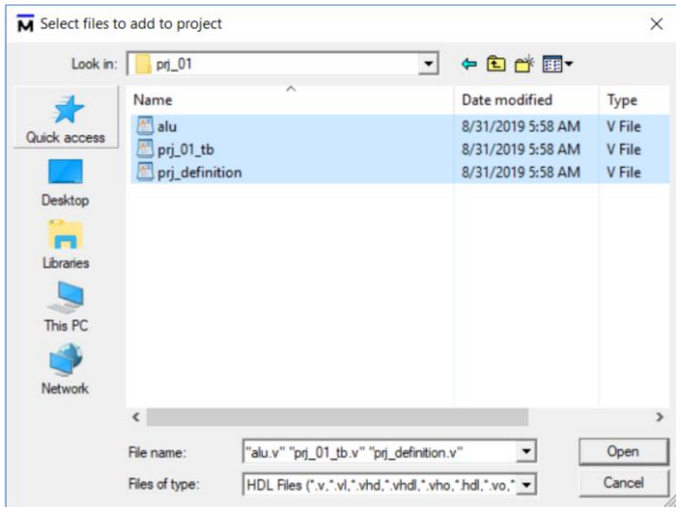
File Name:

Add file as type: default

Folder: Top Level

[✓ Reference from current location] Copy to project directory

Figure 2.3 Add file



Look in: prj_01

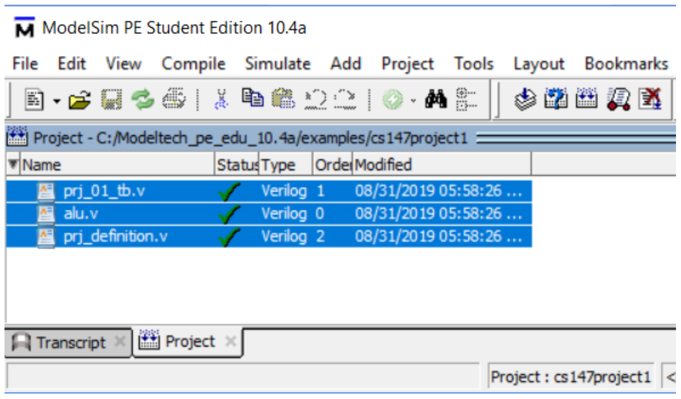
Name	Date modified	Type
alu	8/31/2019 5:58 AM	V File
prj_01_tb	8/31/2019 5:58 AM	V File
prj_definition	8/31/2019 5:58 AM	V File

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

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

Figure 2.4 Select files

5) After that, files are added to the project. Select all three files and Navigate to *Compile* and select *Compile All* (Figure 2.5). If there is no compilation error, the file will be marked as the green checkmark at the following. If any compilation errors happen to the certain file, that file will be marked as red cross.



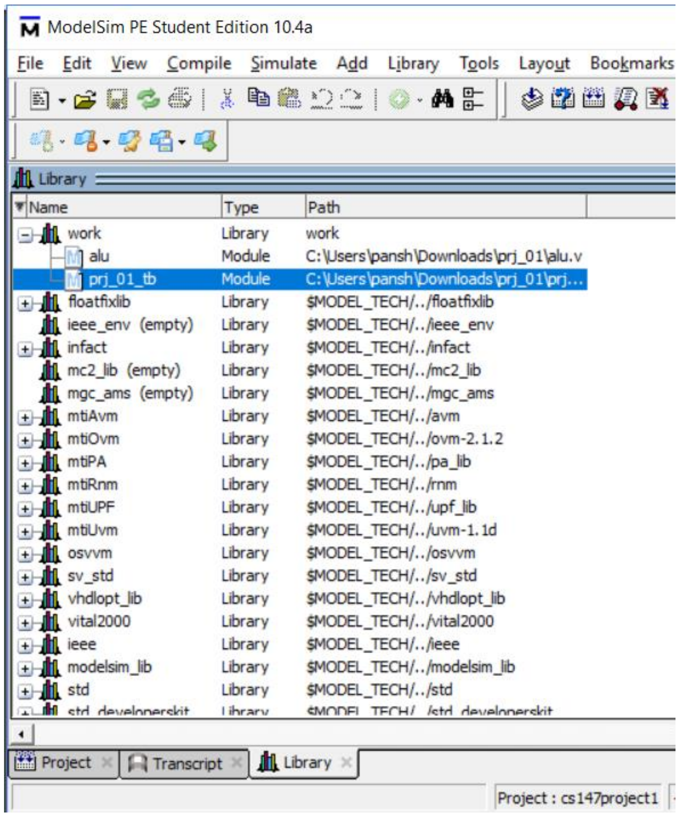
Project - C:/Modeltech_pe_edu_10.4a/examples/cs147project1

Name	Status	Type	Order	Modified
prj_01_tb.v	✓	Verilog	1	08/31/2019 05:58:26 ...
alu.v	✓	Verilog	0	08/31/2019 05:58:26 ...
prj_definition.v	✓	Verilog	2	08/31/2019 05:58:26 ...

Project : cs147project1

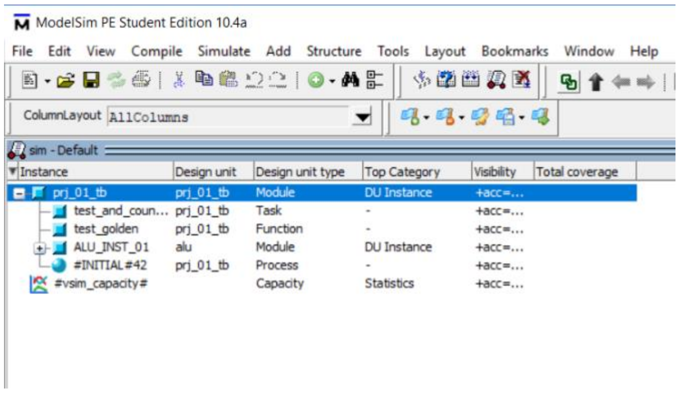
Figure 2.5 Files after compilation

6) Double click on *prj_01_tb* in the work directory of library tab (Figure 2.6). It will produce a tab called sim (Figure 2.7).


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Library Tools Layout Bookmarks
Library
Project Transcript Library
Project : cs147project1

Name	Type	Path
work	Library	work
alu	Module	C:\Users\pansh\Downloads\prj_01\alu.v
prj_01_tb	Module	C:\Users\pansh\Downloads\prj_01\prj...
floatfixlib	Library	$MODEL_TECH/../floatfixlib
ieee_env (empty)	Library	$MODEL_TECH/../ieee_env
infact	Library	$MODEL_TECH/../infact
mc2_lib (empty)	Library	$MODEL_TECH/../mc2_lib
mgc_ams (empty)	Library	$MODEL_TECH/../mgc_ams
mtiAvm	Library	$MODEL_TECH/../avm
mtiOvm	Library	$MODEL_TECH/../ovm-2.1.2
mtiPA	Library	$MODEL_TECH/../pa_lib
mtiRnm	Library	$MODEL_TECH/../rnm
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
vhdlopt_lib	Library	$MODEL_TECH/../vhdlopt_lib
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

Figure 2.6 Test bench


ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Structure Tools Layout Bookmarks Window Help
ColumnLayout AllColumns
sim - Default

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

Figure 2.7 Sim

7) Then right click on the prj_01_tb (Figure 2.8), we can see the add wave. Click on it. It will produce a wave tab. Then select all files, and right click on it. The default number for data is in hexadecimal format. We can change it in the *Radix* to Octal, Binary or Decimal (Figure 2.9).

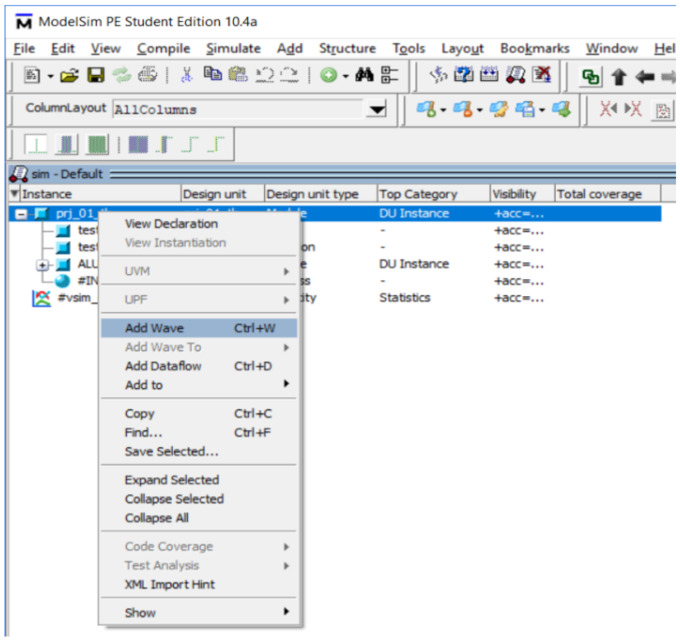


Figure 2.8 Add wave

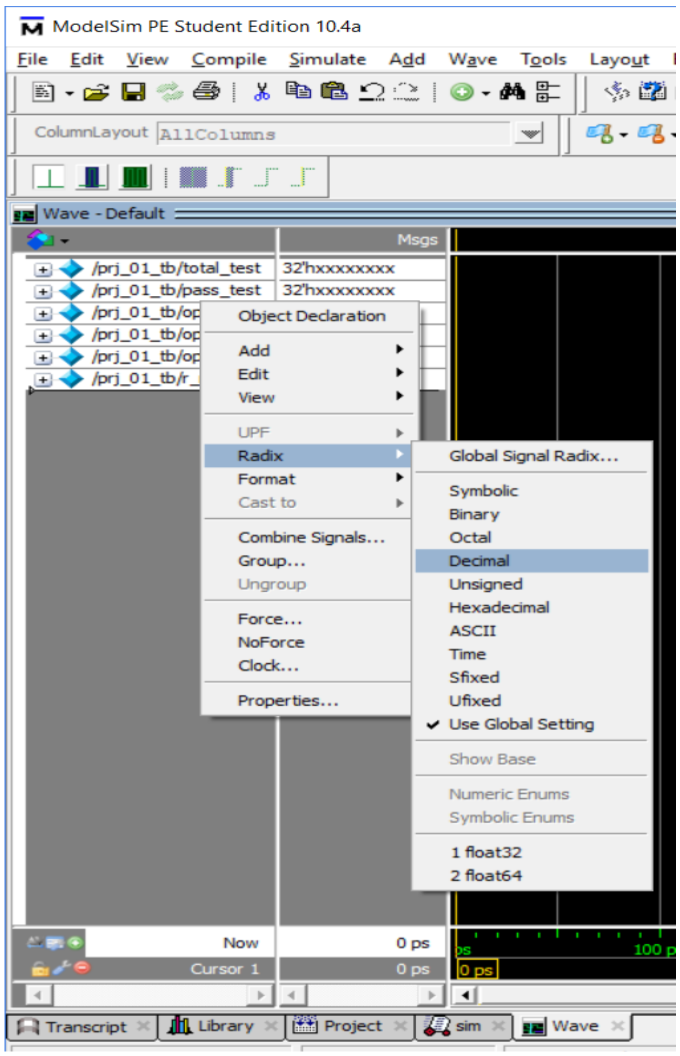


Figure 2.9 Change format

8) After that, we can do *Run All* operation (circle in the figure 2.10) button. It will show the wave the current test program.

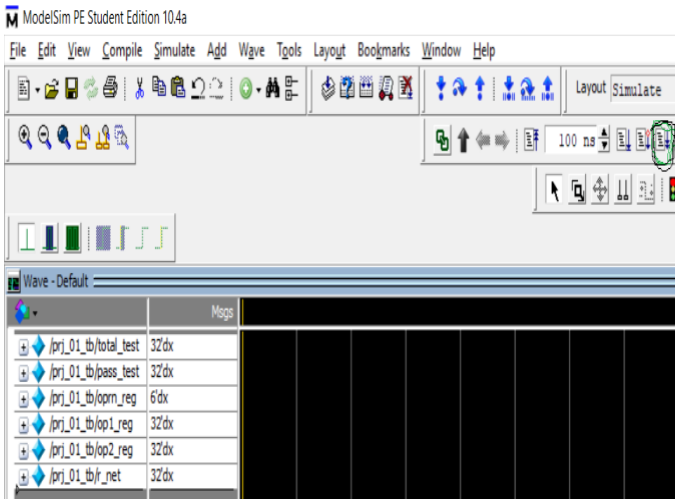

ModelSim PE Student Edition 10.4a
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
100 ns
Wave - Default
Msgs
/prj_01_tb/total_test 32'dx
/prj_01_tb/pass_test 32'dx
/prj_01_tb/oprn_reg 6'dx
/prj_01_tb/op1_reg 32'dx
/prj_01_tb/op2_reg 32'dx
/prj_01_tb/r_net 32'dx

Figure 1 Run the program

3. Requirement of ALU

ALU is a very complex digital circuit, and it is the most imporatant part of CPU. For arithmetic and logic operation we command the computer to do, it is done by ALU. Any complex mathematical and logic program are broken down in terms two operand operations. For example: r = a / b - c / d. It can be broken down at the following,

- T1 = a / b
- T2 = c / b
- r = T1 - T2

In our program, it can be described at the following figure 3.1.

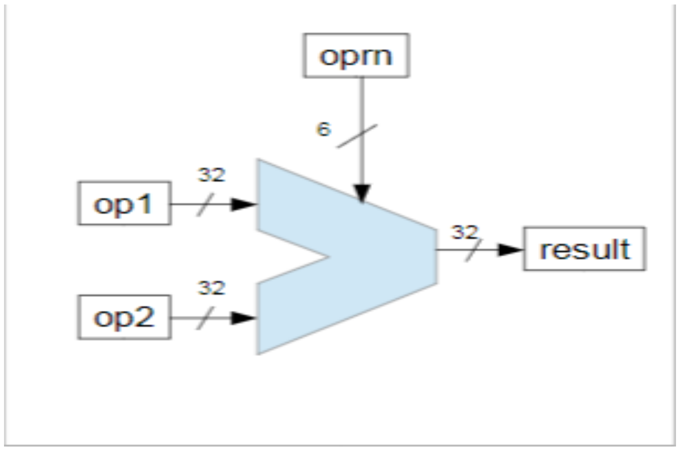


Figure 3.1 Interface Diagram

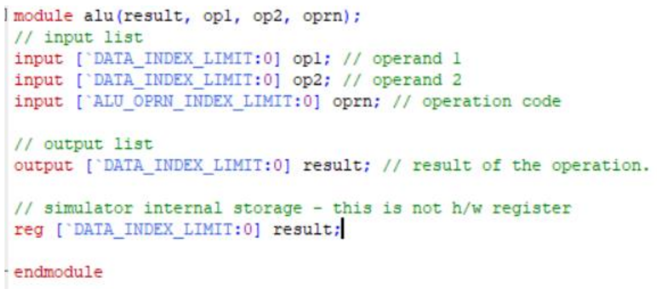
From the diagram, we can see the ALU will get two 32 bits operands and one six bits operation code for the whole the computation. After that, it will produce a 32 bits result.

4. Design and implementation of ALU

For our program, we will Verilog to implement ALU.

Design Strategy:

Define a module in the alu.v, we will use that module as the ALU. I have mentioned the inputs and output above. op1, op2 and oprn are inputs. The result is the output. The following figure 4.1 is the basic module:

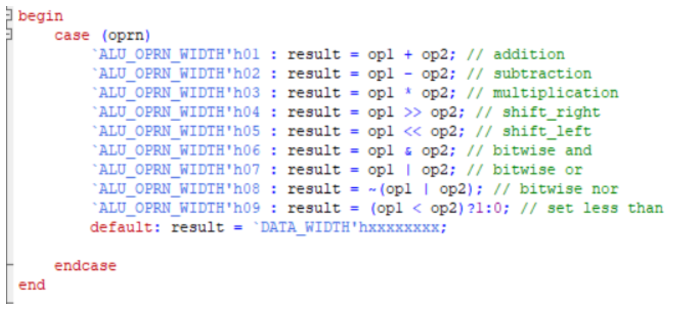


```
module alu(result, op1, op2, oprn);
// input list
input [`DATA_INDEX_LIMIT:0] op1; // operand 1
input [`DATA_INDEX_LIMIT:0] op2; // operand 2
input [`ALU_OPRN_INDEX_LIMIT:0] oprn; // operation code

// output list
output [`DATA_INDEX_LIMIT:0] result; // result of the operation.

// simulator internal storage - this is not h/w register
reg [`DATA_INDEX_LIMIT:0] result;

endmodule
```

Figure 4.1 ALU module

Integer operands are 32 bits and a operation code is 6 bits. The result is a 32 bits integer. We will perform 9 operations in this ALU. They are Integer addition, subtraction and multiplication. The following two operations are shift bit to the left and right. After that, we will do bitwise AND, OR and NOR. Finally, it is set less than. If op1 is less than op2, the result is 0; otherwise, it is 1.

Look at the figure 4.2. Different operation is performed by different operation code which is from ‘h01 to ‘h09. `ALU_OPRN_WIDTH is 6 provided in the prj_defination.v file. It means the operation code is 6 bits. h means hexadecimal 01 to 09. Whenever operands or operation codes are changes, the new operation will be performed. The default value is x if there is no operands or operation codes. Otherwise, the operation will be based on the operation codes.

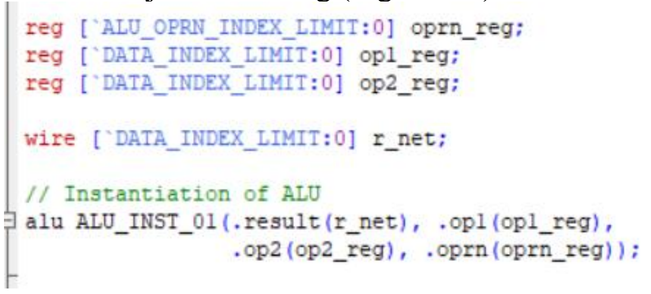


```
begin
    case (oprn)
        `ALU_OPRN_WIDTH'h01 : result = op1 + op2; // addition
        `ALU_OPRN_WIDTH'h02 : result = op1 - op2; // subtraction
        `ALU_OPRN_WIDTH'h03 : result = op1 * op2; // multiplication
        `ALU_OPRN_WIDTH'h04 : result = op1 >> op2; // shift_right
        `ALU_OPRN_WIDTH'h05 : result = op1 << op2; // shift_left
        `ALU_OPRN_WIDTH'h06 : result = op1 & op2; // bitwise and
        `ALU_OPRN_WIDTH'h07 : result = op1 | op2; // bitwise or
        `ALU_OPRN_WIDTH'h08 : result = ~(op1 | op2); // bitwise nor
        `ALU_OPRN_WIDTH'h09 : result = (op1 < op2)?1:0; // set less than
        default: result = `DATA_WIDTH'hxxxxxxxx;

    endcase
end
```

Figure 4.2 Operation

5. Test strategy and test implementation

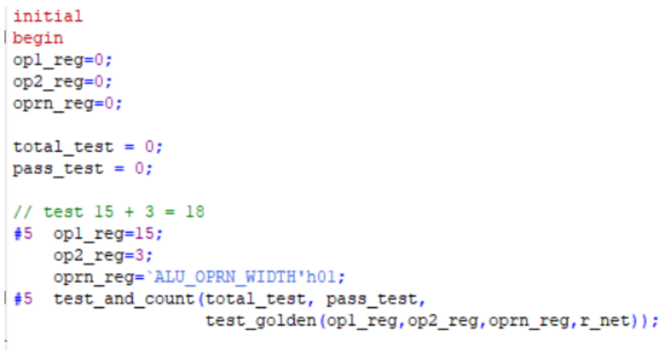
At the beginning, test bench will provide three registers to test the ALU performance and instantiate an ALU object for testing (Figure 5.1).



```
reg [`ALU_OPRN_INDEX_LIMIT:0] oprn_reg;
reg [`DATA_INDEX_LIMIT:0] op1_reg;
reg [`DATA_INDEX_LIMIT:0] op2_reg;

wire [`DATA_INDEX_LIMIT:0] r_net;

// Instantiation of ALU
alu ALU_INST_01(.result(r_net), .op1(op1_reg),
                .op2(op2_reg), .oprn(oprn_reg));
```

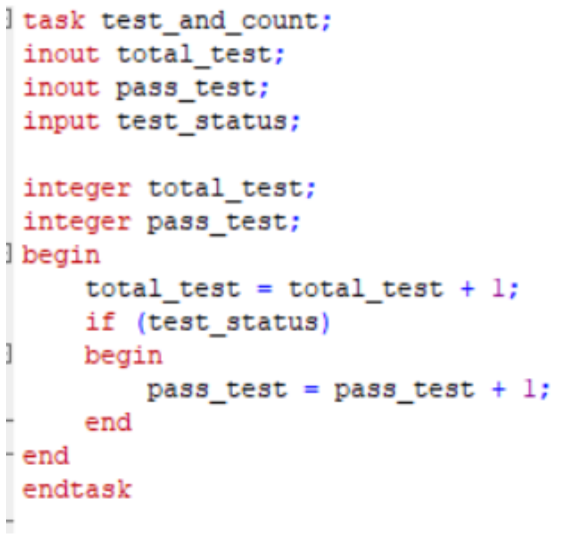
Figure 5.1 Preparation

we start our test cases. I will illustrate one example at the following (Figure 5.2): At the beginning, the three registers are set to 0. They are changed, but the operation code is 0. Thus, the result will be unknown after calling ALU and this is not the first case. The first case is to test 15 + 3 = 8. After giving the three registers new value, ALU will calculate 15 + 3 and return the result to the test bench. The result will be stored in the r_net. Then we do the task test_and_count (figure 5.3). It has three inputs including total_test, pass_test, and test_status. It will the help of test_golden to finish the task to set the passed_test because test_golden will be either ture or false.



```
initial
begin
op1_reg=0;
op2_reg=0;
oprn_reg=0;

total_test = 0;
pass_test = 0;

// test 15 + 3 = 18
#5  op1_reg=15;
    op2_reg=3;
    oprn_reg=`ALU_OPRN_WIDTH'h01;
#5  test_and_count(total_test, pass_test,
                   test_golden(op1_reg,op2_reg,oprn_reg,r_net));
```

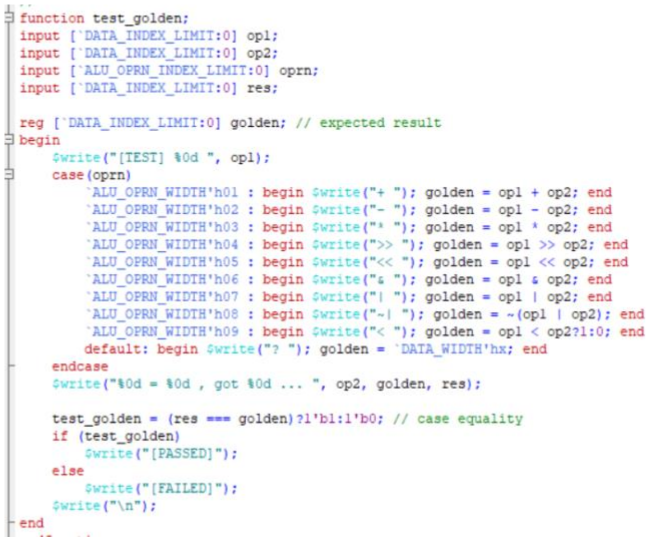
Figure 5.2 Test case



```
task test_and_count;
inout total_test;
inout pass_test;
input test_status;

integer total_test;
integer pass_test;
begin
    total_test = total_test + 1;
    if (test_status)
    begin
        pass_test = pass_test + 1;
    end
end
endtask
```

Figure 5.3 test and count

The test_golden function will be used to test whether the ALU work correctly (figure 5.4).

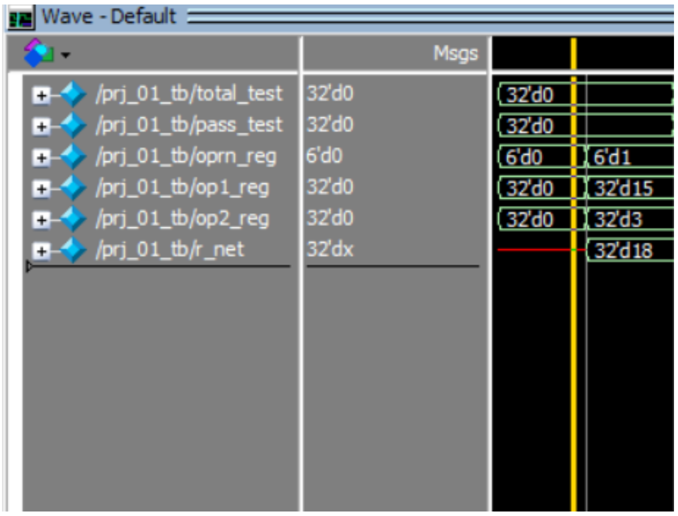


```
function test_golden;
input [`DATA_INDEX_LIMIT:0] op1;
input [`DATA_INDEX_LIMIT:0] op2;
input [`ALU_OPRN_INDEX_LIMIT:0] oprn;
input [`DATA_INDEX_LIMIT:0] res;

reg [`DATA_INDEX_LIMIT:0] golden; // expected result
begin
    $write("[TEST] %0d ", op1);
    case(oprn)
        `ALU_OPRN_WIDTH'h01 : begin $write("+ "); golden = op1 + op2; end
        `ALU_OPRN_WIDTH'h02 : begin $write("- "); golden = op1 - op2; end
        `ALU_OPRN_WIDTH'h03 : begin $write("* "); golden = op1 * op2; end
        `ALU_OPRN_WIDTH'h04 : begin $write(">> "); golden = op1 >> op2; end
        `ALU_OPRN_WIDTH'h05 : begin $write("<< "); golden = op1 << op2; end
        `ALU_OPRN_WIDTH'h06 : begin $write("& "); golden = op1 & op2; end
        `ALU_OPRN_WIDTH'h07 : begin $write("| "); golden = op1 | op2; end
        `ALU_OPRN_WIDTH'h08 : begin $write("~| "); golden = ~(op1 | op2); end
        `ALU_OPRN_WIDTH'h09 : begin $write("< "); golden = op1 < op2?1:0; end
        default: begin $write("? "); golden = `DATA_WIDTH'hx; end
    endcase
    $write("%0d = %0d , got %0d ... ", op2, golden, res);

    test_golden = (res === golden)?1'b1:1'b0; // case equality
    if (test_golden)
        $write("[PASSED]");
    else
        $write("[FAILED]");
    $write("\n");
end
```

Figure 5.4 test golden

For 15 + 3, it will test the result and store the result to the golden. res is r_gen which is the result from the ALU. If they are equal, it means the ALU works correctly and write PASSED. If it works correctly, pass_test will be increased by 1 in the test_and_count. This is the whole process to test one operation 15 + 3 = 18.

At the following, I will show every case and their waveform with decimal and binary number.

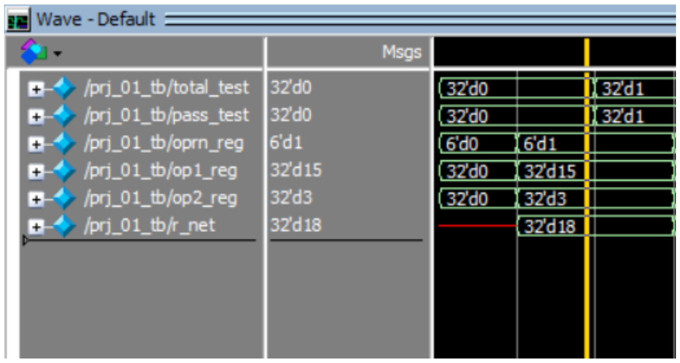


```
Wave - Default
                          Msgs
/prj_01_tb/total_test     32'd0     32'd0
/prj_01_tb/pass_test      32'd0     32'd0
/prj_01_tb/oprn_reg       6'd0      6'd0    6'd1
/prj_01_tb/op1_reg        32'd0     32'd0   32'd15
/prj_01_tb/op2_reg        32'd0     32'd0   32'd3
/prj_01_tb/r_net          32'dx             32'd18
```

Figure 5.5 x output

1. Unknown result
   op1 = 0
   op2 = 0
   oprn = 0

result is x because there is no such oprn – 'h0 in the ALU, the result is unknown with default value.

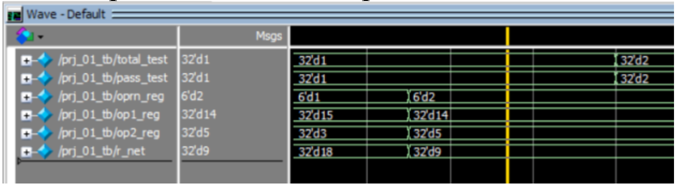


```
Wave - Default
                          Msgs
/prj_01_tb/total_test     32'd0     32'd0             32'd1
/prj_01_tb/pass_test      32'd0     32'd0             32'd1
/prj_01_tb/oprn_reg       6'd1      6'd0    6'd1
/prj_01_tb/op1_reg        32'd15    32'd0   32'd15
/prj_01_tb/op2_reg        32'd3     32'd0   32'd3
/prj_01_tb/r_net          32'd18            32'd18
```

Figure 5.6 Addition output

2. Addition
   op1 = 15
   op2 = 3
   oprn = 1

Decimal 1 is also 1 in the hexadecimal format. The ALU implements addition operation: 15 + 3 = 18.

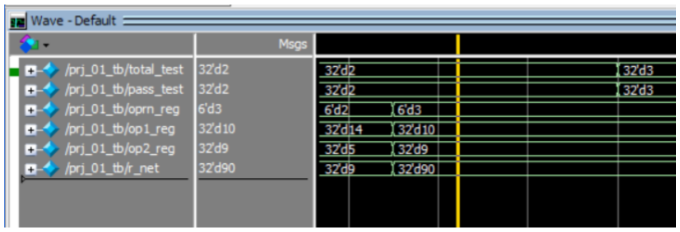


```
Wave - Default
                          Msgs
/prj_01_tb/total_test     32'd1     32'd1             32'd2
/prj_01_tb/pass_test      32'd1     32'd1             32'd2
/prj_01_tb/oprn_reg       6'd2      6'd1     6'd2
/prj_01_tb/op1_reg        32'd14    32'd15   32'd14
/prj_01_tb/op2_reg        32'd5     32'd3    32'd5
/prj_01_tb/r_net          32'd9     32'd18   32'd9
```

Figure 5.7Subtraction output

3. Subtraction
   op1 = 14
   op2 = 5
   oprn = 2

Decimal 2 is also 2 in the hexadecimal format. The ALU implements subtraction operation: 14 – 5 = 9.

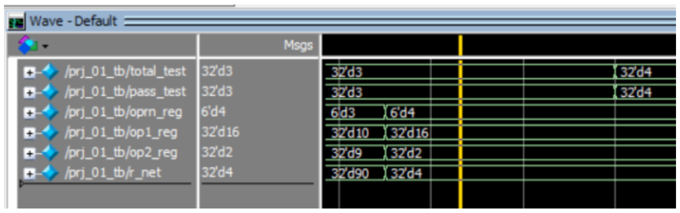


```
Wave - Default
                          Msgs
/prj_01_tb/total_test     32'd2     32'd2             32'd3
/prj_01_tb/pass_test      32'd2     32'd2             32'd3
/prj_01_tb/oprn_reg       6'd3      6'd2     6'd3
/prj_01_tb/op1_reg        32'd10    32'd14   32'd10
/prj_01_tb/op2_reg        32'd9     32'd5    32'd9
/prj_01_tb/r_net          32'd90    32'd9    32'd90
```

Figure 5.8 Multiplication output

4. Multiplication
   op1 = 10
   op2 = 9
   oprn = 3

Decimal 3 is also 3 in the hexadecimal format. The ALU implements multiplication operation: 10 * 9 = 90.

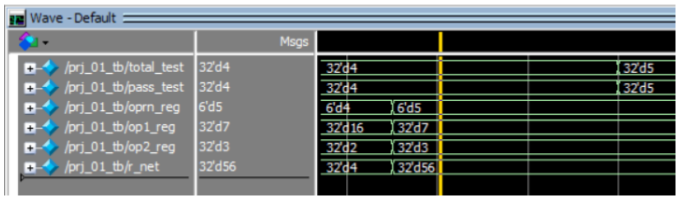


```
Wave - Default
                          Msgs
/prj_01_tb/total_test     32'd3     32'd3             32'd4
/prj_01_tb/pass_test      32'd3     32'd3             32'd4
/prj_01_tb/oprn_reg       6'd4      6'd3     6'd4
/prj_01_tb/op1_reg        32'd16    32'd10   32'd16
/prj_01_tb/op2_reg        32'd2     32'd9    32'd2
/prj_01_tb/r_net          32'd4     32'd90   32'd4
```

Figure 5.9 Shift right

5. Shift right
   op1 = 16
   op2 = 2
   oprn = 4

Decimal 4 is also 4 in the hexadecimal format. The ALU implements shift-right operation: 16 >> 2 = 16 / 2 / 2 = 4. In bit shifting, shifting one bit towards right means the original number will be divided by 2. 16 = 10000 >> 2 = 00100 = 4.



```
Wave - Default
                        Msgs
/prj_01_tb/total_test   32'd4    32'd4                      32'd5
/prj_01_tb/pass_test    32'd4    32'd4                      32'd5
/prj_01_tb/oprn_reg     6'd5     6'd4     6'd5
/prj_01_tb/op1_reg      32'd7    32'd16   32'd7
/prj_01_tb/op2_reg      32'd3    32'd2    32'd3
/prj_01_tb/r_net        32'd56   32'd4    32'd56
```

Figure 5.10 Shift left

6. Shift left

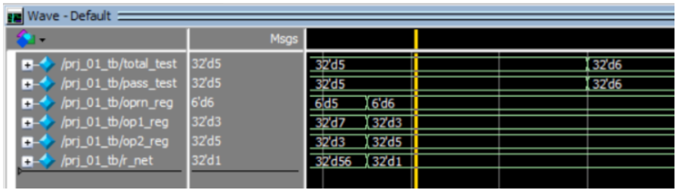
op1 = 7

op2 = 3

oprn = 5

Decimal 5 is also 5 in the hexadecimal format. The ALU implements shift-left operation: 7 * 2 * 2 * 2= 56. In bit shifting, shifting one bit towards left means the original number will be doubled.

7 = 111 << 2 = 111000 = 32 + 16 + 8 = 56



```
Wave - Default
                        Msgs
/prj_01_tb/total_test   32'd5    32'd5                      32'd6
/prj_01_tb/pass_test    32'd5    32'd5                      32'd6
/prj_01_tb/oprn_reg     6'd6     6'd5     6'd6
/prj_01_tb/op1_reg      32'd3    32'd7    32'd3
/prj_01_tb/op2_reg      32'd5    32'd3    32'd5
/prj_01_tb/r_net        32'd1    32'd56   32'd1
```

Figure 5.11 Bitwise AND

7. Bitwise AND

op1 = 3

op2 = 5

oprn = 6

Decimal 6 is also 6 in the hexadecimal format. The ALU implements bitwise AND operation: 3 & 5 = 1. In the AND operation, if one bit is 0, the result will be zero.

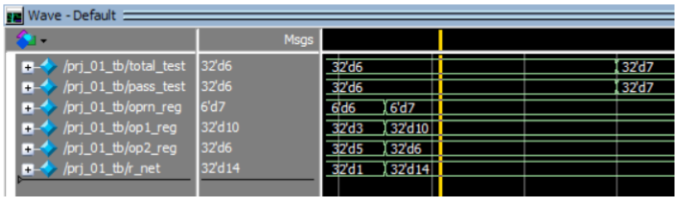
3 = 011 5 = 101

011

101

-----

001 = 1



```
Wave - Default
                        Msgs
/prj_01_tb/total_test   32'd6    32'd6                      32'd7
/prj_01_tb/pass_test    32'd6    32'd6                      32'd7
/prj_01_tb/oprn_reg     6'd7     6'd6     6'd7
/prj_01_tb/op1_reg      32'd10   32'd3    32'd10
/prj_01_tb/op2_reg      32'd6    32'd5    32'd6
/prj_01_tb/r_net        32'd14   32'd1    32'd14
```

Figure 5.12 Bitwise OR

8. Bitwise OR

op1 = 10

op2 = 6

oprn = 7

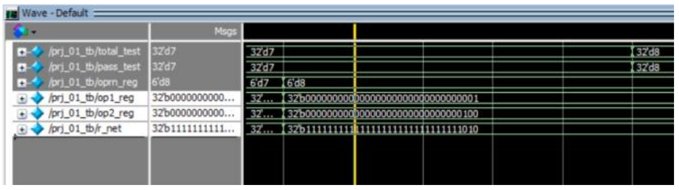
Decimal 7 is also 7 in the hexadecimal format. The ALU implements bitwise OR operation: 10 & 6 = 14. In the OR operation, if one bit is 1, the result will be 1.

10 = 1010 6 = 0110

1010

0110

1110 = 8 + 4 + 2 = 14



```
Wave - Default
                        Msgs
/prj_01_tb/total_test   32'd7            32'd7                                                32'd8
/prj_01_tb/pass_test    32'd7            32'd7                                                32'd8
/prj_01_tb/oprn_reg     6'd8             6'd7    6'd8
/prj_01_tb/op1_reg      32'b0000000000...  32'...  32'b00000000000000000000000000000001
/prj_01_tb/op2_reg      32'b0000000000...  32'...  32'b00000000000000000000000000000100
/prj_01_tb/r_net        32'b1111111111...  32'...  32'b11111111111111111111111111111010
```

Figure 5.13 Bitwise NOR

9. Bitwise NOR

op1 = 1

op2 = 4

oprn = 8

Decimal 8 is also 8 in the hexadecimal format. The ALU implements bitwise NOR operation. In the NOR operation, if one bit is 1, the result will be zero.

1 = 0000 0000 0000 0000 0000 0000 0000 0001

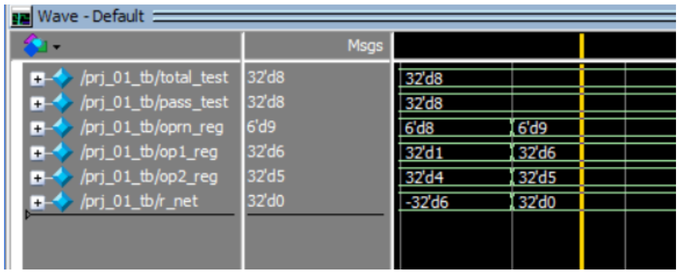
4 = 0000 0000 0000 0000 0000 0000 0000 0100

0000 0000 0000 0000 0000 0000 0000 0001

0000 0000 0000 0000 0000 0000 0000 0100

1111 1111 1111 1111 1111 1111 1111 1010

= 4294967290 for unsigned integer



```
Wave - Default
                        Msgs
/prj_01_tb/total_test   32'd8    32'd8
/prj_01_tb/pass_test    32'd8    32'd8
/prj_01_tb/oprn_reg     6'd9     6'd8     6'd9
/prj_01_tb/op1_reg      32'd6    32'd1    32'd6
/prj_01_tb/op2_reg      32'd5    32'd4    32'd5
/prj_01_tb/r_net        32'd0    -32'd6   32'd0
```

Figure 5.14 Set less than

10. Set less than

op1 = 6

op2 = 5

oprn = 9

Decimal 9 is also 9 in the hexadecimal format. The ALU implements set-less-than operation. Since 6 is greater than 5, 6 > 6 = 0.

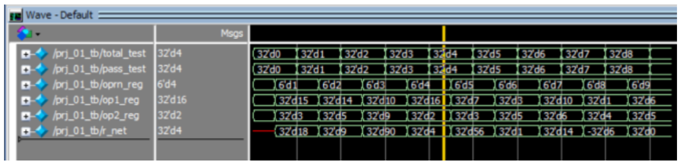
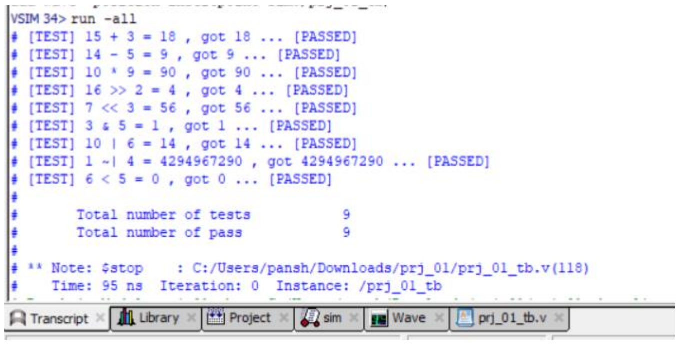

Wave - Default
Msgs
/prj_01_tb/total_test 32'd4 32'd0 32'd1 32'd2 32'd3 32'd4 32'd5 32'd6 32'd7 32'd8
/prj_01_tb/pass_test 32'd4 32'd0 32'd1 32'd2 32'd3 32'd4 32'd5 32'd6 32'd7 32'd8
/prj_01_tb/oprn_reg 6'd4 6'd1 6'd2 6'd3 6'd4 6'd5 6'd6 6'd7 6'd8 6'd9
/prj_01_tb/op1_reg 32'd16 32'd15 32'd14 32'd10 32'd16 32'd7 32'd3 32'd10 32'd1 32'd6
/prj_01_tb/op2_reg 32'd2 32'd3 32'd5 32'd9 32'd2 32'd3 32'd5 32'd6 32'd4 32'd5
/prj_01_tb/r_net 32'd4 32'd18 32'd9 32'd90 32'd4 32'd56 32'd1 32'd14 -32'd6 32'd0

Figure 5.15 All output

The above figure is all operations' output.



```
VSIM 34> run -all
# [TEST] 15 + 3 = 18 , got 18 ... [PASSED]
# [TEST] 14 - 5 = 9 , got 9 ... [PASSED]
# [TEST] 10 * 9 = 90 , got 90 ... [PASSED]
# [TEST] 16 >> 2 = 4 , got 4 ... [PASSED]
# [TEST] 7 << 3 = 56 , got 56 ... [PASSED]
# [TEST] 3 & 5 = 1 , got 1 ... [PASSED]
# [TEST] 10 | 6 = 14 , got 14 ... [PASSED]
# [TEST] 1 ~| 4 = 4294967290 , got 4294967290 ... [PASSED]
# [TEST] 6 < 5 = 0 , got 0 ... [PASSED]
#
#       Total number of tests          9
#       Total number of pass           9
#
# ** Note: $stop    : C:/Users/pansh/Downloads/prj_01/prj_01_tb.v(118)
#    Time: 95 ns  Iteration: 0  Instance: /prj_01_tb
```

Figure 5.16 Final result

At the time of simulating the program, Modelsim will show the test result in the transcript window. From the result, we can the overall condition about our ALU implementation. We can whether we pass all test or we need to some modification. If any error happens, we can also find the location quickly from the transcript window.

6. Conclusion

I learned how to download and set up the ModelSim from this simple project. I got familiar with the basic operation of ModelSim to run programs. I designed and implemented a 32-bits ALU. Also, I use 9 operation cases to test it. Furthermore, I knew the basic semantics and syntax of Verilog. I also learned how the waveform is related to our computation in the process of simulation.

7. Reference

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.