National Cheng Kung University

Department of Electrical Engineering

Introduction to VLSI CAD (Spring 2022)

Lab Session 3

Design of Multiplexers, ALUs and Multiplication

Yu-Chi Chu

Objectives:

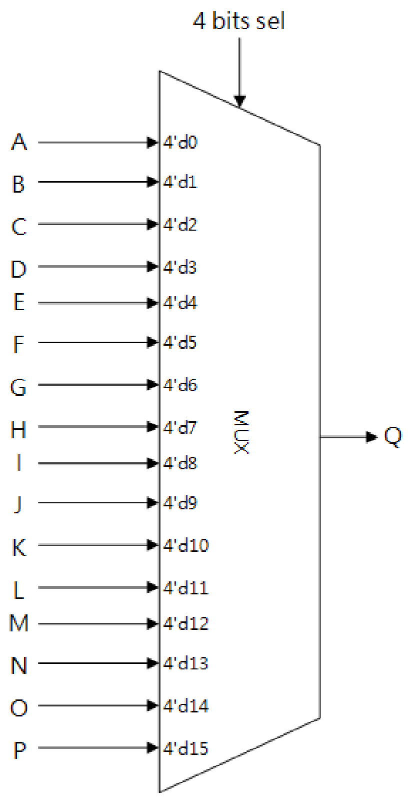
To make you be familiar with some designs of combinational logic, like adders, decoders and some operation that uses multiplication and addition. **You can follow this document to practice, or you have a cleverer way. Please show your best.**

Note that you can extend the spacing if it is not enough for you to answer.

Prob A: A 16-to-1 multiplexer and testbench

1) **Complete a 8-to-1 multiplexer.**

Design a 16-to-1 multiplexer. Design the testbench as well. Testbench need to test all selected inputs and print results.



a. Name your design file *mux16to1.v* and your testbench file *mux16to1_tb.v.*

b. The frame code is given.

c. Inputs: **`A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, sel`**

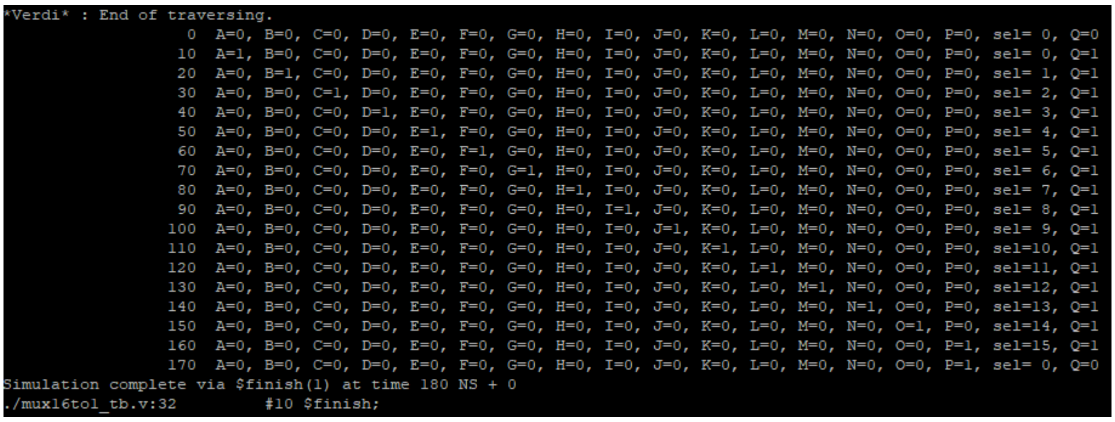
d. Outputs: **`Q`**

e. Include all needed Verilog files in your testbench.

2) Please attach your design waveforms.

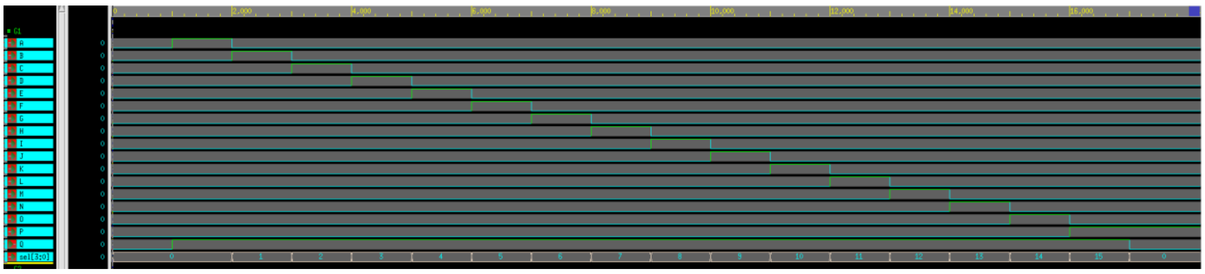
Your simulation result on the terminal.

(You should $monitor the I/O port like the example TB code does!)



```
*Verdi* : End of traversing.
                   0  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 0, Q=0
                  10  A=1, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 0, Q=1
                  20  A=0, B=1, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 1, Q=1
                  30  A=0, B=0, C=1, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 2, Q=1
                  40  A=0, B=0, C=0, D=1, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 3, Q=1
                  50  A=0, B=0, C=0, D=0, E=1, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 4, Q=1
                  60  A=0, B=0, C=0, D=0, E=0, F=1, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 5, Q=1
                  70  A=0, B=0, C=0, D=0, E=0, F=0, G=1, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 6, Q=1
                  80  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=1, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 7, Q=1
                  90  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=1, J=0, K=0, L=0, M=0, N=0, O=0, P=0, sel= 8, Q=1
                 100  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=1, K=0, L=0, M=0, N=0, O=0, P=0, sel= 9, Q=1
                 110  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=1, L=0, M=0, N=0, O=0, P=0, sel=10, Q=1
                 120  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=1, M=0, N=0, O=0, P=0, sel=11, Q=1
                 130  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=1, N=0, O=0, P=0, sel=12, Q=1
                 140  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=1, O=0, P=0, sel=13, Q=1
                 150  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=1, P=0, sel=14, Q=1
                 160  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=1, sel=15, Q=1
                 170  A=0, B=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, J=0, K=0, L=0, M=0, N=0, O=0, P=1, sel= 0, Q=0
Simulation complete via $finish(1) at time 180 NS + 0
./mux16to1_tb.v:32          #10 $finish;
```

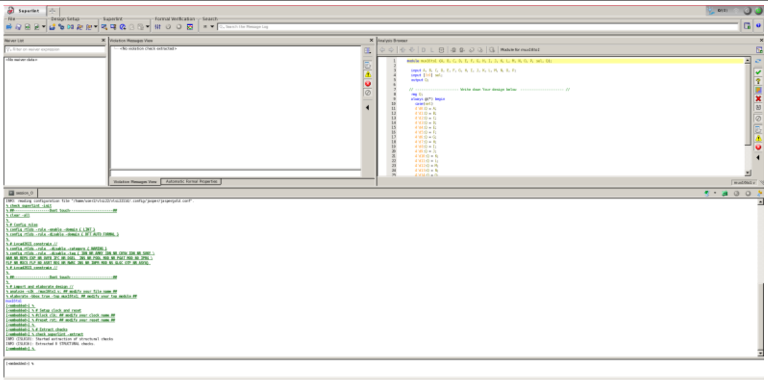
Your waveform :



Explanation of your waveform :

16-1 Mux will decide the output according to the input of ‘sel’. For example, if sel = 1, it will output the value of A, A = 1.

SuperLint Coverage

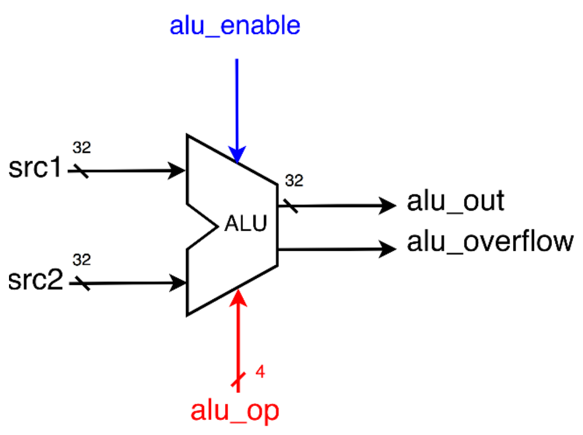


Coverage = 1

Objectives:

Learn how to design an ALU and Register file which are two main components in CPU. This homework you are allowed to use behavioral description to speed up your design process.

Prob B: Arithmetic Logic Unit



1. Based on the reference code we gave you, please implement another 8 operations which are listed as below.(signed : 2's complement)

alu_op	Operation	Description
01000	NOT	alu_out = ~src1
01001	NAND	alu_out = ~(src1 & src2)
01010	MAX	alu_out = max{src1 , src2}
01011	MIN	alu_out = min{src1 , src2}
01100	ABS	alu_out = |src1|
01101	SLT	alu_out = (src1 < src2)?1:0
01110	SLTU	alu_out = ((unsigned)src1 < (unsigned)src2)?1:0
01111	SRA	alu_out = src1 >>> src2
10000	SLL	alu_out = src1 << src2
10001	SLA	alu_out = src1 <<< src2
10010	ROTL	alu_out = src1 rotate left by "src2 bits"
10011	ADDU	alu_out = (unsigned)src1+(unsigned)src2

2. w the file name rules as follow.
   - ALU (provided!)
     - File name: **ALU.v**

- Module name: **ALU**
- ALU testbench
  - File name: **ALU_tb.v**
  - Module name: **ALU_tb**

3. You should verify your code by the following test patterns.

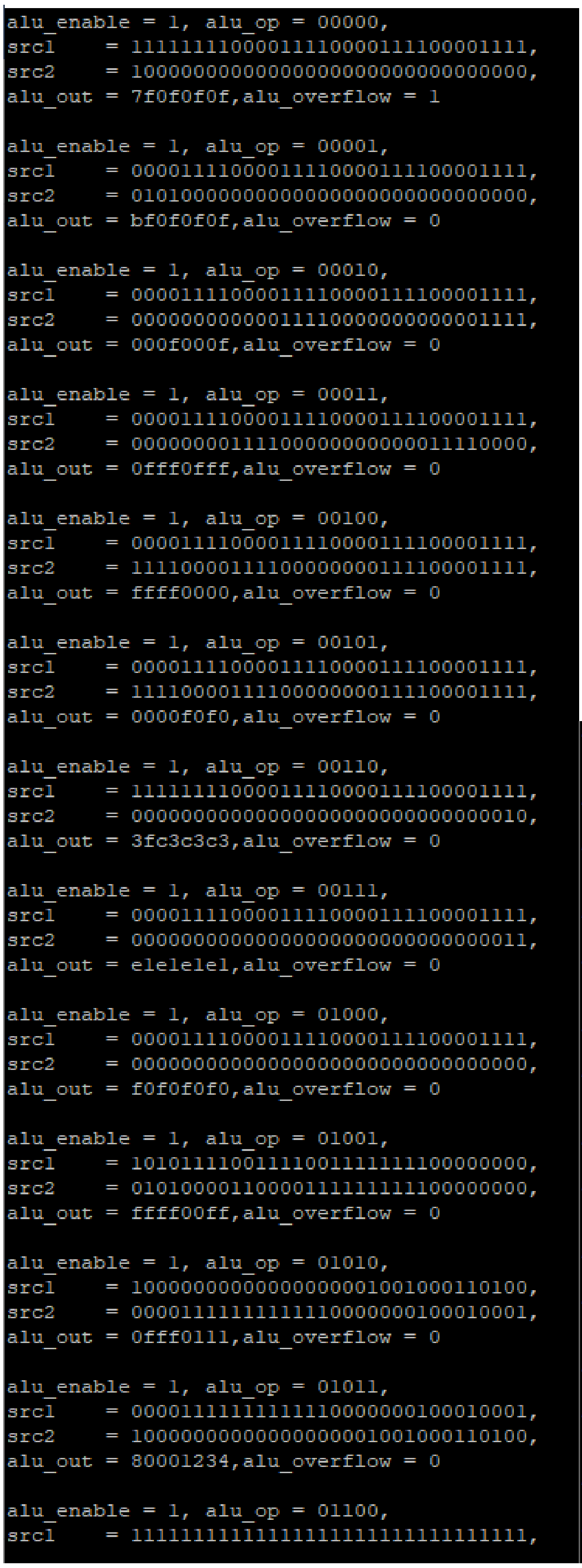
Operation	src1	src2
NOT	32' h0f0f_0f0f	32' h0000_0000
NAND	32' haf3c_ff00	32' h50c3_ff00
MAX	32' h8000_1234	32' h0fff_0111
MIN	32' h0fff_0111	32' h8000_1234
ABS	32' hffff_ffff	32' h0000_0000
SLT	32' h8000_1234	32' h0fff_0111
SLTU	32' hff00_00ff	32' h0000_0002
SRA	32' hff00_00ff	32' h0000_0002
SLL	32' hff00_00ff	32' h0000_0002
SLA	32' hff00_00ff	32' h0000_0002
ROTL	32' h4f00_00ff	32' h0000_0002
ADDU	32' hff00_0000	32' h0000_0002

4. You need to snapshot the waveform of the 12 operations you implemented and explain why they are correct. If you didn't explain you will get no credit!

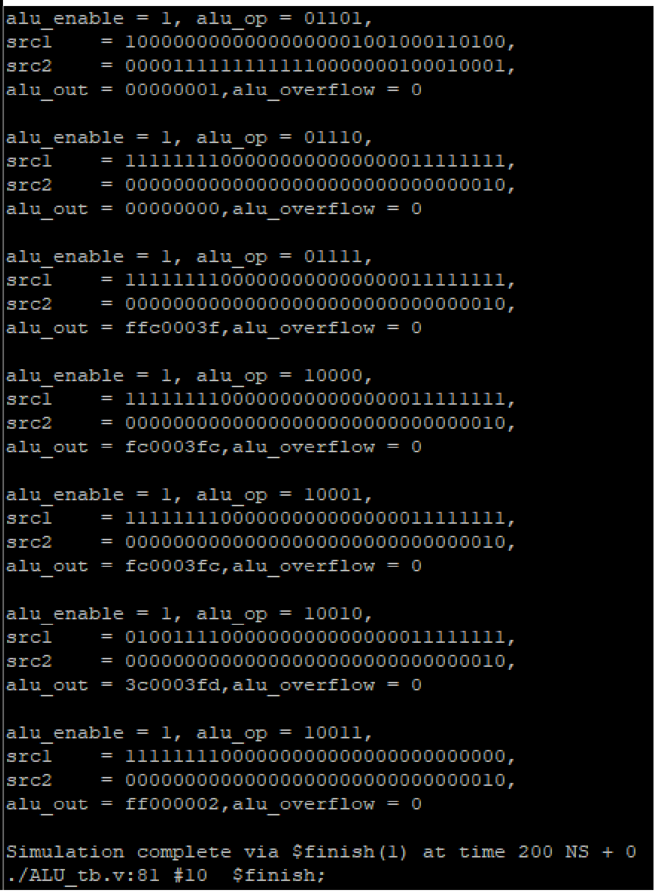
5. Please attach your design waveforms.

Your simulation result on the terminal.

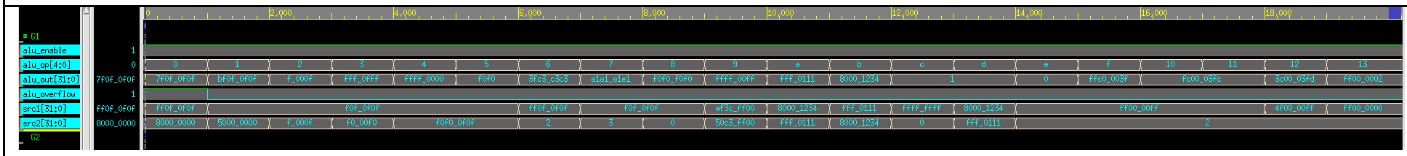
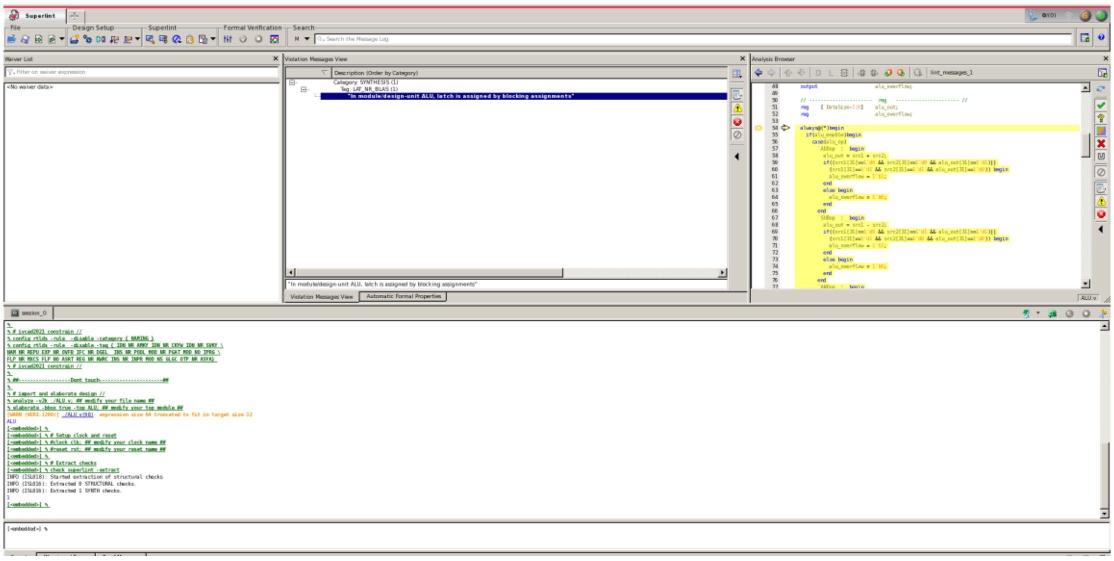
(You should $monitor the I/O port like the example TB code does!)



```
alu_enable = 1, alu_op = 00000,
src1     = 11111111000011110000111100001111,
src2     = 10000000000000000000000000000000,
alu_out = 7f0f0f0f,alu_overflow = 1

alu_enable = 1, alu_op = 00001,
src1     = 00001111000011110000111100001111,
src2     = 01010000000000000000000000000000,
alu_out = bf0f0f0f,alu_overflow = 0

alu_enable = 1, alu_op = 00010,
src1     = 00001111000011110000111100001111,
src2     = 00000000000011110000000000001111,
alu_out = 000f000f,alu_overflow = 0

alu_enable = 1, alu_op = 00011,
src1     = 00001111000011110000111100001111,
src2     = 00000000111100000000000011110000,
alu_out = 0fff0fff,alu_overflow = 0

alu_enable = 1, alu_op = 00100,
src1     = 00001111000011110000111100001111,
src2     = 11110000111100000000111100001111,
alu_out = ffff0000,alu_overflow = 0

alu_enable = 1, alu_op = 00101,
src1     = 00001111000011110000111100001111,
src2     = 11110000111100000000111100001111,
alu_out = 0000f0f0,alu_overflow = 0

alu_enable = 1, alu_op = 00110,
src1     = 11111111000011110000111100001111,
src2     = 00000000000000000000000000000010,
alu_out = 3fc3c3c3,alu_overflow = 0

alu_enable = 1, alu_op = 00111,
src1     = 00001111000011110000111100001111,
src2     = 00000000000000000000000000000011,
alu_out = e1e1e1e1,alu_overflow = 0

alu_enable = 1, alu_op = 01000,
src1     = 00001111000011110000111100001111,
src2     = 00000000000000000000000000000000,
alu_out = f0f0f0f0,alu_overflow = 0

alu_enable = 1, alu_op = 01001,
src1     = 10101111001111001111111100000000,
src2     = 01010000110000111111111100000000,
alu_out = ffff00ff,alu_overflow = 0

alu_enable = 1, alu_op = 01010,
src1     = 10000000000000000001001000110100,
src2     = 00001111111111110000000100010001,
alu_out = 0fff0111,alu_overflow = 0

alu_enable = 1, alu_op = 01011,
src1     = 00001111111111110000000100010001,
src2     = 10000000000000000001001000110100,
alu_out = 80001234,alu_overflow = 0

alu_enable = 1, alu_op = 01100,
src1     = 11111111111111111111111111111111,
```



```
alu_enable = 1, alu_op = 01101,
src1     = 10000000000000000001001000110100,
src2     = 00001111111111110000000100010001,
alu_out = 00000001,alu_overflow = 0

alu_enable = 1, alu_op = 01110,
src1     = 11111111000000000000000011111111,
src2     = 00000000000000000000000000000010,
alu_out = 00000000,alu_overflow = 0

alu_enable = 1, alu_op = 01111,
src1     = 11111111000000000000000011111111,
src2     = 00000000000000000000000000000010,
alu_out = ffc0003f,alu_overflow = 0

alu_enable = 1, alu_op = 10000,
src1     = 11111111000000000000000011111111,
src2     = 00000000000000000000000000000010,
alu_out = fc0003fc,alu_overflow = 0

alu_enable = 1, alu_op = 10001,
src1     = 11111111000000000000000011111111,
src2     = 00000000000000000000000000000010,
alu_out = fc0003fc,alu_overflow = 0

alu_enable = 1, alu_op = 10010,
src1     = 01001111000000000000000011111111,
src2     = 00000000000000000000000000000010,
alu_out = 3c0003fd,alu_overflow = 0

alu_enable = 1, alu_op = 10011,
src1     = 11111111000000000000000000000000,
src2     = 00000000000000000000000000000010,
alu_out = ff000002,alu_overflow = 0

Simulation complete via $finish(1) at time 200 NS + 0
./ALU_tb.v:81 #10  $finish;
```

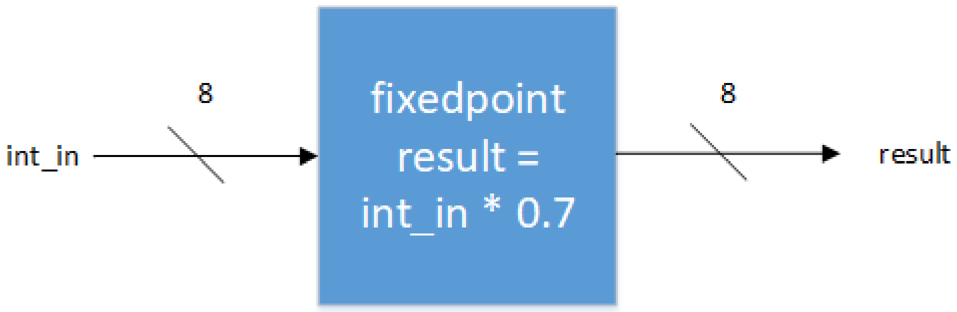
Your waveform :

Explanation of your waveform :
The ALU's operation is determined by the opcode. • **op = 1:** Addition. • **op = 5:** NOR operation. • **op = 9:** Maximum value selection. • **op = 13:** Unsigned addition (ADDU). The example shows that when op = 13, the ALU performs unsigned addition. ff00_0000 + 2 = ff00_0002 = alu_out. This illustrates a simple unsigned addition operation.
SuperLint Coverage
 Coverage = 1

"In module/design-unit ALU, latch is assigned by blocking assignments"

```
output                 alu_overflow;

// -------------------- reg -------------------- //
reg    [`DataSize-1:0]  alu_out;
reg                     alu_overflow;

always@(*)begin
  if(alu_enable)begin
    case(alu_op)
      `ADDop  :  begin
        alu_out = src1 + src2;
        if((src1[31]==1'd0 && src2[31]==1'd0 && alu_out[31]==1'd1)||
          (src1[31]==1'd1 && src2[31]==1'd1 && alu_out[31]==1'd0)) begin
          alu_overflow = 1'b1;
        end
        else begin
          alu_overflow = 1'b0;
        end
      end
      `SUBop  :  begin
        alu_out = src1 - src2;
        if((src1[31]==1'd0 && src2[31]==1'd1 && alu_out[31]==1'd1)||
          (src1[31]==1'd1 && src2[31]==1'd0 && alu_out[31]==1'd0)) begin
          alu_overflow = 1'b1;
        end
        else begin
          alu_overflow = 1'b0;
        end
      end
      `ANDop  :  begin
```

```
%
% # ivcad2021 constrain //
% config rtlds -rule  -disable -category { NAMING }
% config rtlds -rule  -disable -tag { IDN_NR_AMKY IDN_NR_CKYW IDN_NR_SVKY \
NAM_NR_REPU EXP_NR_OVFB IFC_NR_DGEL  INS_NR_PODL MOD_NR_PGAT MOD_NO_IPRG \
FLP_NR_MXCS FLP_NO_ASRT REG_NR_RWRC INS_NR_INPR MOD_NS_GLGC OTP_NR_ASYA}
% # ivcad2021 constrain //
%
% ##------------------Dont touch---------------------##
%
% # import and elaborate design //
% analyze -v2k ./ALU.v; ## modify your file name ##
% elaborate -bbox true -top ALU; ## modify your top module ##
[WARN (VERI-1209)] ./ALU.v(98): expression size 64 truncated to fit in target size 32
ALU
[<embedded>] %
[<embedded>] % # Setup clock and reset
[<embedded>] % #clock clk; ## modify your clock name ##
[<embedded>] % #reset rst; ## modify your reset name ##
[<embedded>] %
[<embedded>] % # Extract checks
[<embedded>] % check superlint -extract
INFO (ISL018): Started extraction of structural checks
INFO (ISL016): Extracted 0 STRUCTURAL checks.
INFO (ISL016): Extracted 1 SYNTH checks.
1
[<embedded>] %
```

Prob C: Practice fixed point



1) Design your Verilog code with the following specifications:

This part is to practice how to use fixed point to multiply an integer and decimal.

First, you need to convert **0.7** to an 8 bits fixed point number (without rounding), and there are 1 bit integer and 7 bits decimal in 8 bits. Next, multiply the 8 bits integer input **int_in** by the converted number and send the **result** to output. The **result** should be the 8 bits integer part of the multiplication result.

a. Number format: unsigned numbers.

b. Name your design file *fixedpoint.v* and your testbench file *fixedpoint_tb.v.*

c. The frame code and testbench are given. Use the testbench to help you verify the result. You don't need to write another testbench in this problem.

d. Inputs: `int_in[7:0]`

e. Outputs: `result[7:0]`

f. The result of the operation should round to an integer. If you find some of your results has a slight difference, like 1, from the expected, it is probable that the rounding is not correct.

a. Follow the PPT file to set the constraints when synthesis.

b. Turn in the synthesized Verilog file named *fixedpoint_syn.v*, the SDF file named *fixedpoint_syn.sdf* and the DDC file named *fixedpoint_syn.ddc*.

c. The simulation command is different for synthesis:

```
ncverilog fixedpoint_tb.v +define+FSDB+syn +access+r
```

2) After you synthesize your design, you may have some information about the circuit. Fill in the following form.

Timing (slack)	**Area (total cell area)**	**Power (total)**
19.42	**30.680399**	**9.0628uW**

3) Please attach your design waveforms.

Explain how you convert 0.7 to a fixed point 8 bits number.

0.7*2 = 1.4

0.4*2 = 0.8

0.8*2 = 1.6

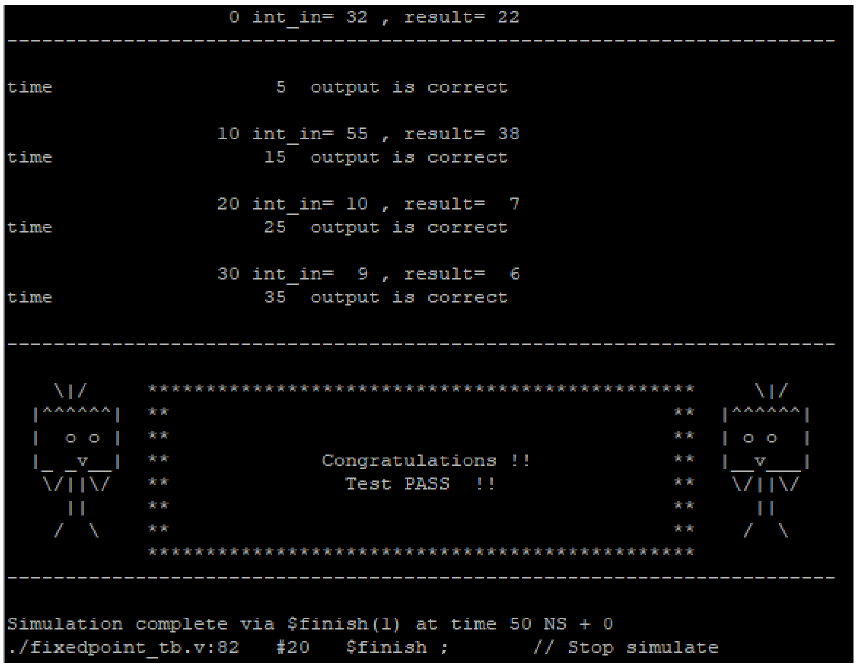
0.6*2 = 1.2

0.2*2 = 0.4

0.4*2 = 0.8

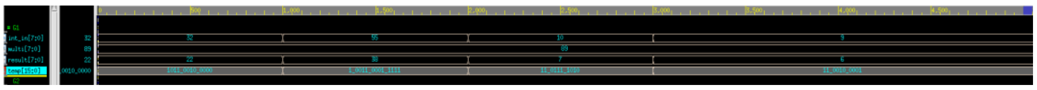
0.8*2 = 1.6

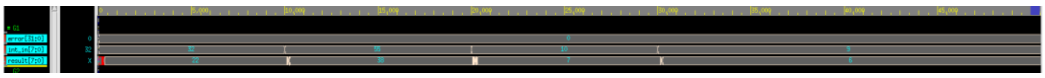
Your simulation result on the terminal. (You should $monitor the I/O port like the example TB code does!)



```
                   0 int_in= 32 , result= 22
----------------------------------------------------------------------

time                       5  output is correct

                  10 int_in= 55 , result= 38
time                      15  output is correct

                  20 int_in= 10 , result=  7
time                      25  output is correct

                  30 int_in=  9 , result=  6
time                      35  output is correct

----------------------------------------------------------------------

    \|/     ***********************************************      \|/
  |^^^^^^|  **                                            **   |^^^^^^|
  |  o o |  **                                            **   | o o  |
  |_ _v__|  **              Congratulations !!            **   |__v___|
   \/||\/   **                Test PASS  !!               **    \/||\/
     ||     **                                            **      ||
    /  \    **                                            **     /  \
            ***********************************************
----------------------------------------------------------------------

Simulation complete via $finish(1) at time 50 NS + 0
./fixedpoint_tb.v:82    #20    $finish ;        // Stop simulate
```

Your waveform (RTL & Synthesis) :

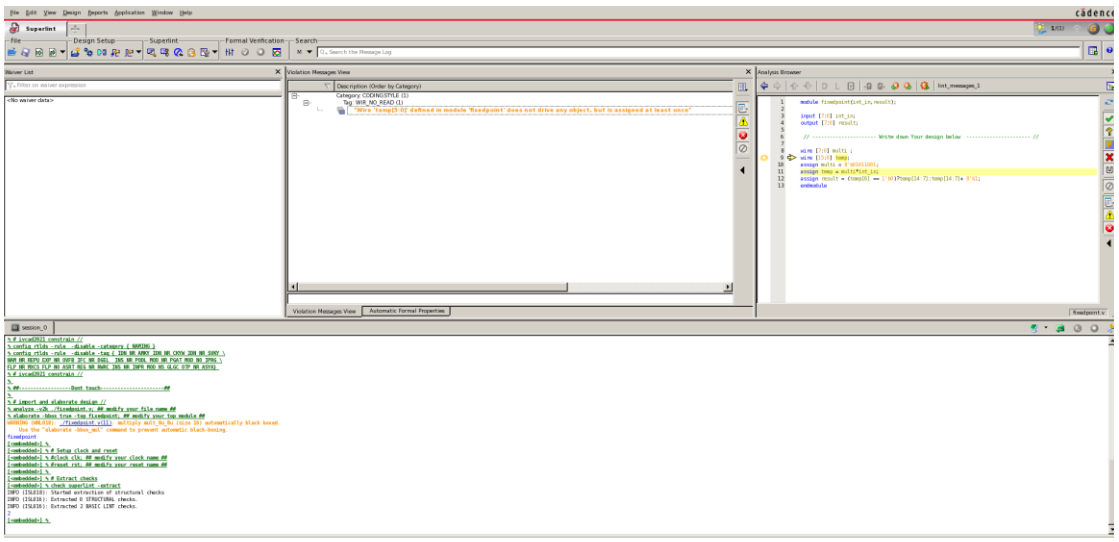




Explanation of your waveform :

The multi value (0.7 in decimal) is represented in binary. The decimal point is assumed to be between multi[6] and multi[7]. When int_in = 10, the value of temp[6] is 1, indicating that rounding is necessary. The output is therefore 00000111 (binary) which is 7 in decimal. This shows how rounding is performed based on the bit after the assumed decimal point.

SuperLint Coverage



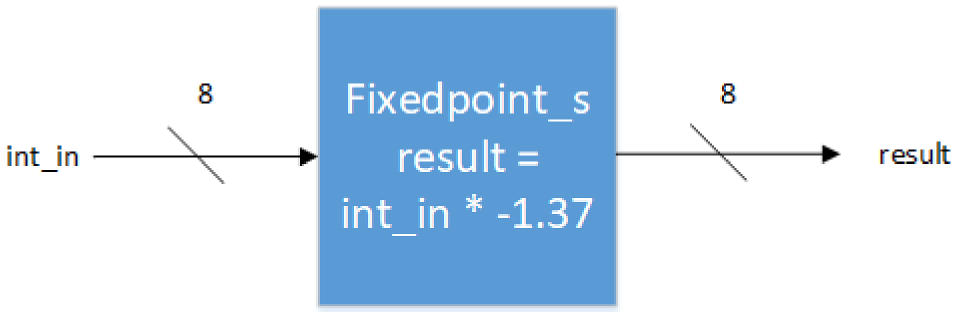
Category: CODINGSTYLE (1)
Tag: WIR_NO_READ (1)
"Wire 'temp[5:0]' defined in module 'fixedpoint' does not drive any object, but is assigned at least once"

```
module fixedpoint(int_in,result);

input [7:0] int_in;
output [7:0] result;

 // --------------------- Write down Your design below  --------------------- //

wire [7:0] multi ;
wire [15:0] temp;
assign multi = 8'b01011001;
assign temp = multi*int_in;
assign result = (temp[6] == 1'b0)?temp[14:7]:temp[14:7]+ 8'b1;
endmodule
```

```
% # ivcad2021 constrain //
% config rtlds -rule  -disable -category { NAMING }
% config rtlds -rule  -disable -tag { IDN NR AMKY IDN NR CKYW IDN NR SVKY \
NAM NR REPU EXP NR OVFB IFC NR DGEL  INS NR PODL MOD NR PGAT MOD NO IPRG \
FLP NR MXCS FLP NO ASRT REG NR RWRC INS NR INPR MOD NS GLGC OTP NR ASYA}
% # ivcad2021 constrain //
%
% ##------------------Dont touch--------------------##
%
% # import and elaborate design //
% analyze -v2k ./fixedpoint.v; ## modify your file name ##
% elaborate -bbox true -top fixedpoint; ## modify your top module ##
WARNING (WNL018): ./fixedpoint.v(11): multiply mult_8u_8u (size 16) automatically black boxed.
    Use the "elaborate -bbox_mul" command to prevent automatic black-boxing.
fixedpoint
[<embedded>] %
[<embedded>] % # Setup clock and reset
[<embedded>] % #clock clk; ## modify your clock name ##
[<embedded>] % #reset rst; ## modify your reset name ##
[<embedded>] %
[<embedded>] % # Extract checks
[<embedded>] % check superlint -extract
INFO (ISL018): Started extraction of structural checks
INFO (ISL016): Extracted 0 STRUCTURAL checks.
INFO (ISL016): Extracted 2 BASIC LINT checks.
2
[<embedded>] %
```

Coverage =   0.9

Prob D: Practice fixed point (signed)



4) Design your Verilog code with the following specifications:

This part is similar to Prob C, but you should do a signed calculation.

First, you need to convert **-1.37** to an 8 bits fixed point number (without rounding), and there are 1 bit signed bit, 1 bit integer, and 6 bits decimal in 8 bits. Next, multiply the 8 bits integer input **int_in** by the converted number and send **result** to output. The **result** should be the 8 bits integer part of the multiplication result.

Remember to deal with **rounding** and **overflow** problems. If the answer is larger than 127, you should output 127. If the answer is lower than -128, you should output -128. The rounding method is on ppt. Please do ensure that the negative situation is considered.

d. Number format: signed numbers.

e. Name your design file *fixedpoint_s.v* and your testbench file *fixedpoint_s_tb.v.*

f. The frame code and testbench are given. Use the testbench to help you verify the result. You don't need to write another testbench in this problem.

g. Inputs: **`int_in[7:0]`**

h. Outputs: **`result[7:0]`**

i. The result of the operation should round to an integer. If you find some of your result has a slightly difference, like 1, from the expected, it is probable that the rounding is not correct.

j. Follow the PPT file to set the constrains when synthesis.

k. Turn in the synthesized Verilog file named *fixedpoint_s_syn.v*, the SDF file named *fixedpoint_s_syn.sdf* and the DDC file named *fixedpoint_s_syn.ddc*.

l. The simulation command is different for synthesis:

```
ncverilog fixedpoint_tb.v +define+FSDB+syn +access+r
```

5) After you synthesize your design, you may have some information about the

circuit. Fill in the following form.

Timing (slack)	**Area (total cell area)**	**Power (total)**
19.29	**35.431199**	**10.3010uW**

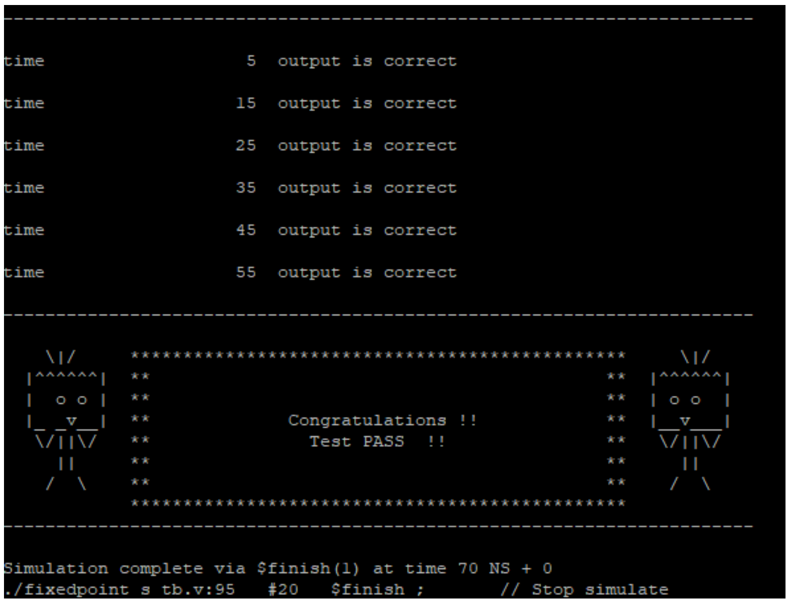
6) Please attach your design waveforms.

Explain how you convert -1.37 to a fixed point 8 bits number.

The value 1.37 is converted to its binary representation: 01010111. This representation uses 1 bit for the sign, 1 bit for the integer part, and 6 bits for the fractional part. The fractional part (0.37) is multiplied by 2 repeatedly to generate the binary fractional bits.
The value 0.37 is then used as an example to demonstrate two's complement conversion for negative numbers.

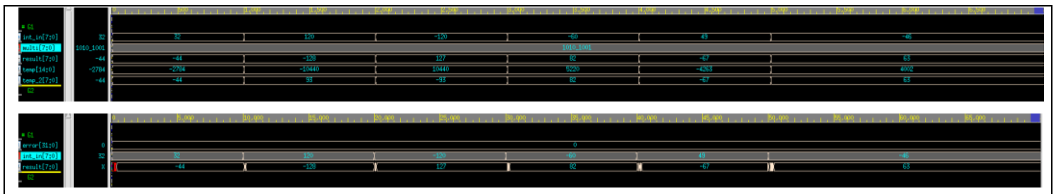
0.37*2 = 0.74	2’s complement to negative value
0.74*2 = 1.48	10101000 + 1 = 10101001
0.48*2 = 0.96	
0.96*2 = 1.92	
0.92*2 = 1.84	
0.84*2 = 1.68	

Your simulation result on the terminal. (You should $monitor the I/O port like the example TB code does!)



```
----------------------------------------------------------------------

time                   5  output is correct

time                  15  output is correct

time                  25  output is correct

time                  35  output is correct

time                  45  output is correct

time                  55  output is correct

----------------------------------------------------------------------

    \|/     ********************************************     \|/
  |^^^^^^|  **                                        **  |^^^^^^|
  |  o o |  **                                        **  | o o  |
  |_ _v__|  **            Congratulations !!          **  |__v___|
   \/||\/   **              Test PASS  !!             **   \/||\/
     ||     **                                        **     ||
    /  \    **                                        **    /  \
            ********************************************

----------------------------------------------------------------------

Simulation complete via $finish(1) at time 70 NS + 0
./fixedpoint s tb.v:95   #20   $finish ;       // Stop simulate
```

Your waveform (RTL & Synthesis) :



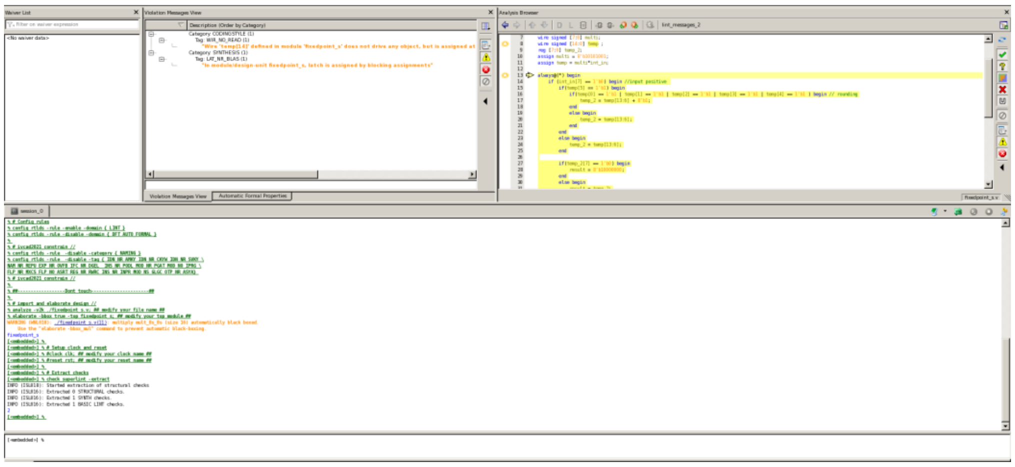
Explanation of your waveform :

When int_in = 32, the product of int_in and multi is stored in temp. Since 32 is positive, the final result should be negative. Rounding is applied to temp, and the relevant bits (from temp[6] to temp[13]) are extracted into temp2. If temp2[7] is 1, the output is negative and represents the correct value. If temp2[7] is 0, it indicates an overflow, and the output is -128. The example shows that when int_in = 120, temp[2] = 93, which means the most significant bit is 0, indicating an overflow, so the output is -128.

For negative int_in values, the overflow detection logic is reversed. Overflow occurs when the most significant bit is 1. The example shows that when int_in = -120, temp2 = -93, which indicates an overflow, and the output is 127.

The synthesized waveform shows the error, int_in, and result signals. Again, the result signals are slightly delayed due to the synthesis process.

SuperLint Coverage



```
<No waiver data>

Description (Order by Category)
Category: CODINGSTYLE (1)
  Tag: WIR_NO_READ (1)
    "Wire 'temp[14]' defined in module 'fixedpoint_s' does not drive any object, but is assigned at
Category: SYNTHESIS (1)
  Tag: LAT_NR_BLAS (1)
    "In module/design-unit fixedpoint_s, latch is assigned by blocking assignments"
```

```
wire signed [7:0] multi;
wire signed [14:0] temp ;
reg [7:0] temp_2;
assign multi = 8'b10101001;
assign temp = multi*int_in;

always@(*) begin
    if (int_in[7] == 1'b0) begin //input positive
        if(temp[5] == 1'b1) begin
            if(temp[0] == 1'b1 | temp[1] == 1'b1 | temp[2] == 1'b1 | temp[3] == 1'b1 | temp[4] == 1'b1 ) begin // rounding
                temp_2 = temp[13:6] + 8'b1;
            end
            else begin
                temp_2 = temp[13:6];
            end
        end
        else begin
            temp_2 = temp[13:6];
        end

        if(temp_2[7] == 1'b0) begin
            result = 8'b10000000;
        end
        else begin
```

```
% # Config rules
% config rtlds -rule -enable -domain { LINT }
% config rtlds -rule -disable -domain { DFT AUTO FORMAL }
%
% # ivcad2021 constrain //
% config rtlds -rule  -disable -category { NAMING }
% config rtlds -rule  -disable -tag { IDN_NR_AMKY IDN_NR_CKYW IDN_NR_SVKY \
NAR_NR_REPU EXP_NR_OVFB IFC_NR_DGEL  INS_NR_PODL MOD_NR_PGAT MOD_NS_IPRG \
FLP_NR_MXCS FLP_NO ASRT REG_NR_RWRC INS_NR_INPR MOD_NS_GLGC OTP_NR_ASYA}
% # ivcad2021 constrain //
%
% ##-------------------Dont touch----------------------##
%
% # import and elaborate design //
% analyze -v2k ./fixedpoint_s.v; ## modify your file name ##
% elaborate -bbox true -top fixedpoint_s; ## modify your top module ##
WARNING (WNL018): ./fixedpoint_s.v(11): multiply mult_8s_8s (size 16) automatically black boxed.
    Use the "elaborate -bbox_mul" command to prevent automatic black-boxing.
fixedpoint_s
[<embedded>] %
[<embedded>] % # Setup clock and reset
[<embedded>] % #clock clk; ## modify your clock name ##
[<embedded>] % #reset rst; ## modify your reset name ##
[<embedded>] %
[<embedded>] % # Extract checks
[<embedded>] % check superlint -extract
INFO (ISL018): Started extraction of structural checks
INFO (ISL016): Extracted 0 STRUCTURAL checks.
INFO (ISL016): Extracted 1 SYNTH checks.
INFO (ISL016): Extracted 1 BASIC LINT checks.
}
[<embedded>] %

[<embedded>] %
```

Coverage = 0.95

Problem		Command
ProbA	Compile	% ncverilog mux16to1.v
	Simulate	% ncverilog mux16to1_tb.v +define+FSDB +access+r
ProbB	Compile	% ncverilog ALU.v
	Simulate	% ncverilog ALU_tb.v +define+FSDB +access+r
ProbC	Compile	% ncverilog fixedpoint.v
	Simulate	% ncverilog fixedpoint_tb.v +define+FSDB +access+r
	Synthesis	% ncverilog fixedpoint_tb.v +define+FSDB+syn +access+r
ProbD	Compile	% ncverilog fixedpoint_s.v
	Simulate	% ncverilog fixedpoint_s_tb.v +define+FSDB +access+r
	Synthesis	% ncverilog fixedpoint_s_tb.v +define+FSDB+syn +access+r

Appendix A : Commands we will use to check your homework