MARIAM MOHAMED MARZOUK

ASSIGNMENT_1

Question 1

1) Add more test vectors to the adder testbench done in the class to reach 100% code coverage.

Question 1

I. Verification Plan

Label	Desciription	Stimulus Generation	Functionality check
Testbench Setup	Initialize testbench signals and clock generation.	clk = 0; forever #1 clk =~clk;	Verify that the clock signal is toggling correctly.
Reset Assertion	Assert and de-assert reset signal.	reset = 1; reset = 0;	Check that the output is zero when reset is asserted.
Test Case 1	Test the minimum value for both A and B (MAXNEG + MAXNEG).	A = -8; B = -8;	Verify that C = -16.
Test Case 2	Test zero values for both A and B (ZERO + ZERO).	A = 0; B = 0;	Verify that C = 0.
Test Case 3	Test the minimum and maximum values (MAXNEG + MAXPOS).	A = -8; B = 7;	Verify that C = -1.
Test Case 4	Test maximum positive value and zero (MAXPOS + ZERO).	A = 7; B = 0;	Verify that C = 7.
Test Case 5	Test zero and minimum value (ZERO + MAXNEG).	A = 0; B = -8;	Verify that C = -8.
Test Case 6	Test the maximum value for both A and B (MAXPOS + MAXPOS).	A = 7; B = 7;	Verify that C = 14.
Test Case 7	Test minimum value and zero (MAXNEG + ZERO).	A = -8; B = 0;	Verify that C = -8.
Test Case 8	Test maximum positive and minimum values (MAXPOS + MAXNEG).	A = 7; B = -8;	Verify that C = -1.
Test Case 9	Test zero and maximum value (ZERO + MAXPOS).	A = 0; B = 7;	Verify that C = 7.
Error Count	Count the number of incorrect results.	Increment error_count when C !== expected_result.	Display the total number of errors.
Correct Count	Count the number of correct results.	Increment correct_count when C === expected_result.	Display the total number of correct results.
Stop Simulation	End the simulation and display results.	$stop;	Ensure simulation halts and results are displayed.

II. RTL Design

```
module adder (
    input  clk,
    input  reset,
    input  signed [3:0] A, // Input data A in 2's complement
    input  signed [3:0] B, // Input data B in 2's complement
    output reg signed [4:0] C // Adder output in 2's complement
        );

    // Register output C
    always @(posedge clk or posedge reset) begin
       if (reset)
         C <= 5'b0;
       else
         C <= A + B;
    end
```

III. TestBench

```
module ADDER_TB;
     logic clk;
     logic reset;
     logic signed [3:0] A;   // Input data A in 2's complement
     logic signed [3:0] B;  // Input data B in 2's complement
     logic signed [4:0] C;     // Adder output in 2's complement
  adder DUT (.*);
 initial begin
  clk =0;
  forever
  #1
  clk =~clk;
  end
localparam MAXPOS=7, ZERO=0 ,MAXNEG=-8;
int error_count;
int correct_count;
initial begin
assert_reset;
A=MAXNEG ;B=MAXNEG ;check_result(-16);
A=ZERO ;B=ZERO ;check_result(-0);
A=MAXNEG ;B=MAXPOS ;check_result(-1);
A=MAXPOS ;B=ZERO ;check_result(7);
A=ZERO ;B=MAXNEG ;check_result(-8);
A=MAXPOS ;B=MAXPOS ;check_result(14);
A=MAXNEG ;B=ZERO ;check_result(-8);
A=MAXPOS ;B=MAXNEG ;check_result(-1);
A=ZERO ;B=MAXPOS ;check_result(7);
assert_reset;
$display("error_count=%d , correct_count=%d",error_count,correct_count);
$stop;
end
task check_result(input signed [4:0] expected_result);
 @(negedge clk);
if(expected_result !== C )begin
 $display ("incorrect result");
error_count++;
end
else correct_count++;
endtask
task assert_reset;
reset=1;
check_result(0);
reset =0;
endtask
endmodule
```

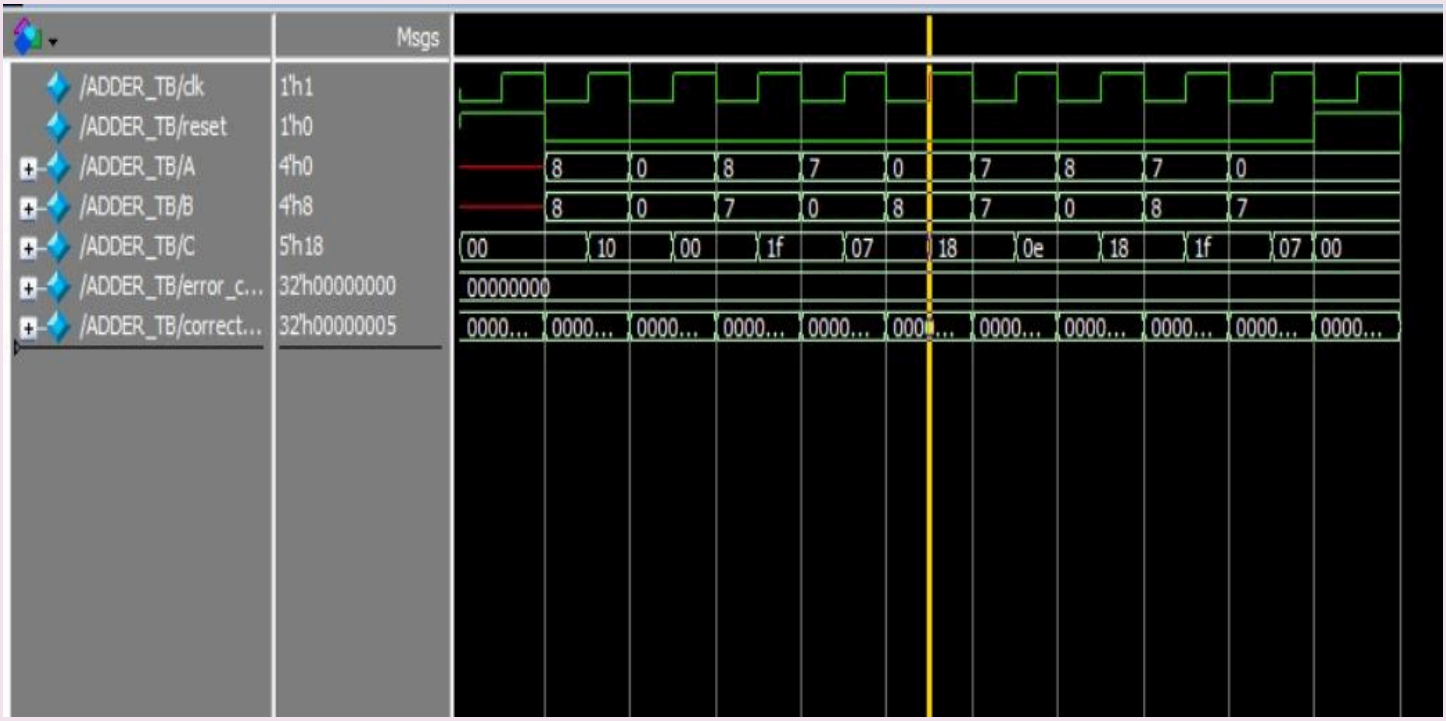
IV. Counters

```
# error_count=          0 , correct_count=          11
```

V. Do File

```
vlib work
vlog adder.v ADDER_TB.sv  +cover -covercells
vsim -voptargs=+acc work.ADDER_TB -cover
add wave *
coverage save ADDER_TB.ucdb -onexit -du work.adder
run -all
```

VI. Waveform



VII. Coverage Report

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\ADDER_TB#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File adder.v
-----------------------------------IF Branch-----------------------------------
    11                                        13     Count coming in to IF
    11             1                           4
    13             1                           9
Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       3         3         0   100.00%

============================Statement Details=============================

Statement Coverage for instance /\ADDER_TB#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File adder.v
    10             1                          13
    12             1                           4
    14             1                           9
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         30        30         0   100.00%

============================Toggle Details============================

Toggle Coverage for instance /\ADDER_TB#DUT  --

                                    Node      1H->0L      0L->1H  "Coverage"
                                    ----------------------------------------
                                   A[0-3]          1           1      100.00
                                   B[0-3]          1           1      100.00
                                   C[4-0]          1           1      100.00
                                     clk           1           1      100.00
                                   reset           1           1      100.00

Total Node Count     =         15
Toggled Node Count   =         15
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (30 of 30 bins)


Total Coverage By Instance (filtered view): 100.00%
```

Question 2

2) Verify the functionality of the following 4 to 2 priority encoder The design consists of 3 inputs which are the following:

- 4-bit input D
- clk
- synchronous active high rst which resets the outputs to 0

The design has 2 output ports (2-bit output Y and 1-bit output valid) which update with the positive edge of clock. Output valid is set to 1 when more than one input line is high. If all the inputs are '0', then valid output is zero. In this case, the output Y is considered as don't care conditions denoted by 'X'.

D3	D2	D1	D0	Y1	Y0	valid
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Since the design is simple, it is required to perform exhaustive verification which means generating all possible input combinations for the input D and make sure that the outputs are correct.

Question 2

I. Verification plan

Label	Description	Stimulus Generation	Functionality check
Testbench Setup	Initialize testbench signals and clock generation.	clk = 0; forever #1 clk = ~clk;	Verify that the clock signal is toggling correctly.
Reset Assertion	Assert and de-assert reset signal.	rst = 1; rst = 0;	Check that output Y and valid are both zero when reset is asserted.
Error Count Init	Initialize error count.	error_count = 0;	Ensure the error count starts from 0.
Correct Count Init	Initialize correct count.	correct_count = 0;	Ensure the correct count starts from 0.
Test Case 1	Test with no input set (D = 0000).	D = 4'b0000;	Verify that Y = 0 and valid = 0.
Test Case 2	Test with highest priority bit set (D = 1000).	D = 4'b1000;	Verify that Y = 0 and valid = 1.
Test Case 3	Test with the next highest priority bit set (D = 0100).	D = 4'b0100;	Verify that Y = 1 and valid = 1.
Test Case 4	Test with the next highest priority bit set (D = 0010).	D = 4'b0010;	Verify that Y = 2 and valid = 1.
Test Case 5	Test with the lowest priority bit set (D = 0001).	D = 4'b0001;	Verify that Y = 3 and valid = 1.
Looping Tests	Repeat test cases 1-5 multiple times to check consistency.	Run a loop to repeat test cases for i < 10.	Ensure consistency in results over multiple cycles.
Error Reporting	Display error count at the end of the test.	$display("%t: error count=%0d", $time, error_count);	Ensure all errors are logged and displayed.
Correct Reporting	Display correct count at the end of the test.	$display("%t: correct count=%0d", $time, correct_count);	Ensure all correct results are logged and displayed.
Stop Simulation	End the simulation and display results.	$stop;	Ensure simulation halts and results are displayed.

II. RTL design

```
1  module priority_enc (
2  input  clk,
3  input  rst,
4  input  [3:0] D,
5  output reg [1:0] Y, //we should add reg
6  output  reg valid        //we should add reg
7  );
8
9  always @(posedge clk or posedge rst) begin
10 if (rst) begin
11 Y <= 2'b00; //2 bits
12 valid <= 1'b0; //2nd out =0
13 end
14 else begin
15     casex (D)
16       4'b1000: Y <= 2'b00;
17       4'bx100: Y <= 2'b01;
18       4'bxx10: Y <= 2'b10;
19       4'bxxx1: Y <= 2'b11;
20       default: Y <= 2'b00; // Default case
21     endcase
22 end
23     valid <= (~|D)? 1'b0: 1'b1;
24 end
25 endmodule
```

III. Testbench

```
module PE_TB;
logic  clk;
logic  rst;
logic  [3:0] D;
logic  [1:0] Y;
logic  valid;

priority_enc DUT(.*);
integer error_count;
integer correct_count;

initial begin
clk=0;
forever #1 clk=~clk;
end
initial begin
D=4'b0000;
error_count=0;
correct_count=0;
assert_reset;
for (int i = 0; i < 10; i++) begin
D=4'b0000;check_result_Y(0);check_result_valid(0);
D=4'b1000;check_result_Y(0);check_result_valid(1);
D=4'b0100;check_result_Y(1);check_result_valid(1);
D=4'b0010;check_result_Y(2);check_result_valid(1);
D=4'b0001;check_result_Y(3);check_result_valid(1);
end
$display("%t:error count=
assert_reset;count=
$stop,error_count,correct_count);
end
task check_result_Y(input [1:0] expected_result_Y );
@(negedge clk);
if(expected_result_Y!==Y) begin
$display("%t: Error: For D=0d
Y should equal 0d
but is 0d
",$time,D,expected_result_Y,Y);
endtask

task check_result_valid(input expected_result_valid );
@(negedge clk);
if(expected_result_valid!==valid) begin
$display("%t: Error: For D=0d
valid should equal 0d
but is 0d
",$time,D,expected_result_valid,valid);
endtask

task assert_reset;
rst=1;
check_result_Y(0);
check_result_valid(0);
rst=0;
endtask
endmodule
```

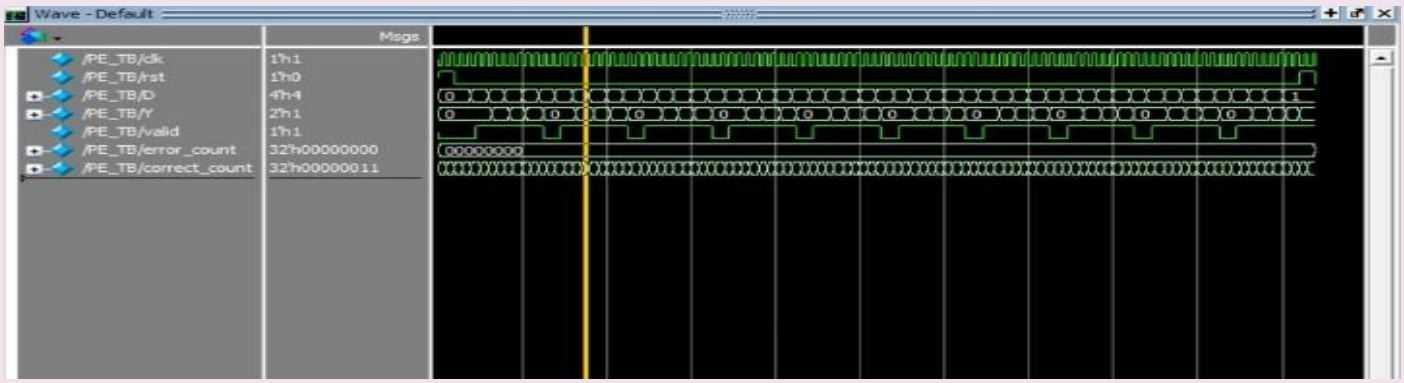
IV. Counters

```
204:error count=0 and correct count=102
```

V. Do File

```
vlib work
vlog priority_enc.v PE_TB.sv  +cover -covercells
vsim -voptargs=+acc work.PE_TB -cover
add wave *
coverage save PE_TB.ucdb -onexit -du work.priority_enc
run -all
```

VI. Waveform



VII. Coverage Report

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         7         7         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\PE_TB#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File priority_enc.v
-----------------------------------IF Branch-------------------------------------
    10                                       104     Count coming in to IF
    10             1                           5
    14             1                          99
Branch totals: 2 hits of 2 branches = 100.00%

---------------------------------CASE Branch-------------------------------------
    15                                       99     Count coming in to CASE
    16             1                         20     |
    17             1                         20
    18             1                         20
    19             1                         20
    20             1                         19
Branch totals: 5 hits of 5 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       9         9         0   100.00%

================================Statement Details================================

Statement Coverage for instance /\PE_TB#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File priority_enc.v
    9              1                         104
    11             1                           5
    12             1                           5
    16             1                          20
    17             1                          20
    18             1                          20
    19             1                          20
    20             1                          19
    23             1                         104
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         18        18         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\PE_TB#DUT  --

                                      Node      1H->0L      0L->1H  "Coverage"
                                     ----------------------------------------
                                    D[0-3]           1           1      100.00
                                    Y[1-0]           1           1      100.00
                                       clk           1           1      100.00
                                       rst           1           1      100.00
                                     valid           1           1      100.00

Total Node Count     =          9
Toggled Node Count   =          9
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (18 of 18 bins)


Total Coverage By Instance (filtered view): 100.00%
```

Question 3

3) ALU design will be provided and has the following characteristics.

- Reset which resets C to 0.
- 4-bit signed inputs, A and B
- 5-bit registered signed output C
- 4 op-codes
    - add
    - sub (A-B)
    - bitwise invert input A
    - reduction OR input B

Assume the following encoding of the opcodes.

Opcode	Encoding
Add	2'b00
Sub	2'b01
bitwise invert input A	2'b10
reduction OR input B	2'b11

Do a directed verification the same we did in the class with the adder to apply test vectors hitting the boundaries (extreme values) for the addition and subtraction and think how to approach the bitwise invert and reduction OR to make sure the output is correct.

Question 3

I. Verification Plan

Label	Description	Stimulus Generation	Functionality check
Testbench Setup	Initialize testbench signals and clock generation.	clk = 0; forever #1 clk = ~clk;	Verify that the clock signal is toggling correctly.
Reset Assertion	Assert and de-assert reset signal.	rst = 1; rst = 0;	Check that output C is zero when reset is asserted.
Error Count Init	Initialize error count.	error_count = 0;	Ensure the error count starts from 0.
Correct Count Init	Initialize correct count.	correct_count = 0;	Ensure the correct count starts from 0.
Add Test: MAXNEG + MAXNEG	Test addition with both A and B as MAXNEG.	Opcode = Add; A = -8; B = -8;	Verify that C = -16 when A = -8 and B = -8.
Add Test: MAXNEG + ZERO	Test addition with A as MAXNEG and B as ZERO.	Opcode = Add; A = -8; B = 0;	Verify that C = -8 when A = -8 and B = 0.
Add Test: MAXPOS + MAXNEG	Test addition with A as MAXPOS and B as MAXNEG.	Opcode = Add; A = 7; B = -8;	Verify that C = -1 when A = 7 and B = -8.
Add Test: MAXPOS + ZERO	Test addition with A as MAXPOS and B as ZERO.	Opcode = Add; A = 7; B = 0;	Verify that C = 7 when A = 7 and B = 0.
Add Test: MAXPOS + MAXPOS	Test addition with both A and B as MAXPOS.	Opcode = Add; A = 7; B = 7;	Verify that C = 14 when A = 7 and B = 7.
Add Test: ZERO + ZERO	Test addition with both A and B as ZERO.	Opcode = Add; A = 0; B = 0;	Verify that C = 0 when A = 0 and B = 0.
Sub Test: MAXNEG - MAXNEG	Test subtraction with both A and B as MAXNEG.	Opcode = Sub; A = -8; B = -8;	Verify that C = 0 when A = -8 and B = -8.
Sub Test: MAXNEG - ZERO	Test subtraction with A as MAXNEG and B as ZERO.	Opcode = Sub; A = -8; B = 0;	Verify that C = -8 when A = -8 and B = 0.
Sub Test: MAXNEG - MAXPOS	Test subtraction with A as MAXNEG and B as MAXPOS.	Opcode = Sub; A = -8; B = 7;	Verify that C = -15 when A = -8 and B = 7.
Sub Test: MAXPOS - MAXNEG	Test subtraction with A as MAXPOS and B as MAXNEG.	Opcode = Sub; A = 7; B = -8;	Verify that C = 15 when A = 7 and B = -8.
Not_A Test: ~MAXNEG	Test bitwise NOT operation on A as MAXNEG.	Opcode = Not_A; A = -8;	Verify that C = 7 when A = -8.
Not_A Test: ~MAXPOS	Test bitwise NOT operation on A as MAXPOS.	Opcode = Not_A; A = 7;	Verify that C = -8 when A = 7.
Not_A Test: ~ZERO	Test bitwise NOT operation on A as ZERO.	Opcode = Not_A; A = 0;	Verify that C = -1 when A = 0.
ReductionOR_B Test: Reduction OR of MAXNEG	Test reduction OR operation on B as MAXNEG.	Opcode = ReductionOR_B; B = -8;	Verify that C = 1 when B = -8.
ReductionOR_B Test: Reduction OR of ZERO	Test reduction OR operation on B as ZERO.	Opcode = ReductionOR_B; B = 0;	Verify that C = 0 when B = 0.
ReductionOR_B Test: Reduction OR of MAXPOS	Test reduction OR operation on B as MAXPOS.	Opcode = ReductionOR_B; B = 7;	Verify that C = 1 when B = 7.
Sub Test: MAXPOS - ZERO	Test subtraction with A as MAXPOS and B as ZERO.	Opcode = Sub; A = 7; B = 0;	Verify that C = 7 when A = 7 and B = 0.
Sub Test: MAXPOS - MAXPOS	Test subtraction with both A and B as MAXPOS.	Opcode = Sub; A = 7; B = 7;	Verify that C = 0 when A = 7 and B = 7.
Sub Test: ZERO - MAXNEG	Test subtraction with A as ZERO and B as MAXNEG.	Opcode = Sub; A = 0; B = -8;	Verify that C = 8 when A = 0 and B = -8.
Sub Test: ZERO - ZERO	Test subtraction with both A and B as ZERO.	Opcode = Sub; A = 0; B = 0;	Verify that C = 0 when A = 0 and B = 0.
Sub Test: ZERO - MAXPOS	Test subtraction with A as ZERO and B as MAXPOS.	Opcode = Sub; A = 0; B = 7;	Verify that C = -7 when A = 0 and B = 7.
Error Reporting	Display the total number of errors at the end of the test.	$display("Error count: %0d", error_count);	Ensure all errors are logged and displayed.
Correct Reporting	Display the total number of correct results at the end of the test.	$display("Correct count: %0d", correct_count);	Ensure all correct results are logged and displayed.
Stop Simulation	End the simulation and display results.	$stop;	Ensure simulation halts and results are displayed.

II. RTL Design

```
module ALU (
    input  clk,
    input  reset,
    input  [1:0] Opcode,    // The opcode
    input  signed [3:0] A, // Input data A in 2's complement
    input  signed [3:0] B, // Input data B in 2's complement
    output reg signed [4:0] C // ALU output in 2's complement

        );

    reg signed [4:0]       Alu_out; // ALU output in 2's complement

    localparam          Add             = 2'b00; // A + B
    localparam               Sub             = 2'b01; // A - B
    localparam          Not_A       = 2'b10; // ~A
    localparam          ReductionOR_B   = 2'b11; // |B

    // Do the operation
    always @* begin
       case (Opcode)
          Add:              Alu_out = A + B;
          Sub:              Alu_out = A - B;
          Not_A:            Alu_out =~A;
          ReductionOR_B:    Alu_out = |B;
        default:   Alu_out = 5'b0;
       endcase
    end // always @ *

    // Register output C
    always @(posedge clk or posedge reset) begin
       if (reset)
         C <= 5'b0;
       else
         C<= Alu_out;
    end

endmodule
```

III. TestBench

```
module ALU_TB;
    // SIGNAL DECLARATION
    logic clk;
    logic reset;
    logic [1:0] Opcode;  // The opcode
    logic signed [3:0] A;  // Input data A in 2's complement
    logic signed [3:0] B;  // Input data B in 2's complement
    logic signed [4:0] C;  // ALU output in 2's complement
    // DUT INSTANTIATION
    ALU DUT (
        .clk(clk),
        .reset(reset),
        .Opcode(Opcode),
        .A(A),
        .B(B),
        .C(C)
    );
    // CLOCK GENERATION
    initial begin
        clk = 0;
        forever #1 clk = ~clk;
    end
    // LOCAL PARAMETERS
    localparam  Add = 2'b00; // A + B
    localparam  Sub = 2'b01; // A - B
    localparam  Not_A = 2'b10; // ~A
    localparam  ReductionOR_B = 2'b11; // |B
    localparam signed MAXPOS = 7, ZERO = 0, MAXNEG = -8;
    // COUNTER
    integer error_count;
    integer correct_count;
    // TEST
    initial begin
        // INITIALIZE SIGNALS
        A = 4'sb0000;
        B = 4'sb0000;
        error_count = 0;
        correct_count = 0;
        // TEST
        assert_reset;
        // ADD TEST
        Opcode = Add;#20;
        A = MAXNEG; B = MAXNEG; #20; check_result(-16);
        A = MAXNEG; B = ZERO; #20; check_result(-8);
        A = MAXPOS; B = MAXNEG; #20; check_result(-1);
        A = MAXPOS; B = ZERO; #20; check_result(7);
        A = MAXPOS; B = MAXPOS; #20; check_result(14);
        A = ZERO; B = ZERO; #20; check_result(0);
        // SUB TEST
```

```
// SUB TEST
    Opcode = Sub;#20;
    A = MAXNEG; B = MAXNEG; #20; check_result(0);
    A = MAXNEG; B = ZERO; #20; check_result(-8);
    A = MAXNEG; B = MAXPOS; #20; check_result(-15);
    A = MAXPOS; B = MAXNEG; #20; check_result(15);
    // NOT A TEST
    Opcode = Not_A;#20;
    A = MAXNEG; #20; check_result(7);  // ~(-8) should be 7 for 4-bit signed
    A = MAXPOS; #20; check_result(-8); // ~(7) should be -8 for 4-bit signed
    A = ZERO; #20; check_result(-1);   // ~(0) should be -1 for 4-bit signed
    // REDUCTION OR B TEST
    Opcode = ReductionOR_B;#20;
    B = MAXNEG; #20; check_result(1); // Reduction OR of -8 should be 1
    B = ZERO; #20; check_result(0);   // Reduction OR of 0 should be 0
    B = MAXPOS; #20; check_result(1); // Reduction OR of 7 should be 1
    // SUB TEST
    Opcode = Sub;#20;
    A = MAXPOS; B = ZERO; #20; check_result(7);
    A = MAXPOS; B = MAXPOS; #20; check_result(0);
    A = ZERO; B = MAXNEG; #20; check_result(8);
    A = ZERO; B = ZERO; #20; check_result(0);
    A = ZERO; B = MAXPOS; #20; check_result(-7);
    // End simulation
    $display("Correct count:
, Errors count:
", correct_count, error_count);
  end
```

```
77    // TASK CHECK
78 v  task check_result(input signed [4:0] expected_result);
79       @(negedge clk)
80 v     if (expected_result !== C) begin
81          $display("%t: Error: For Opcode = %0b A = %0d and B = %0d and C should be equal to %0d but it is equal %0d", $time, Opcode, A, B, expected_result, C);
82          error_count++;
83       end
84 v     else begin
85          correct_count++;
86       end
87    endtask
```

```
1     // TASK RESET
2       task assert_reset();
3           // SET RESET
4           reset = 1;
5           #10;
6           // Check if output C is zero
7           if (C !== 5'b0) begin
8
9               $display("%t: Error: After asserting reset, C should be 0 but is
10  ", $time, C)error_count++;
11      end
12          else begin
13              correct_count++;
14          end
15          // DEASSERT RESET
16          reset = 0;
17          #10;
18      endtask
19  endmodule
```

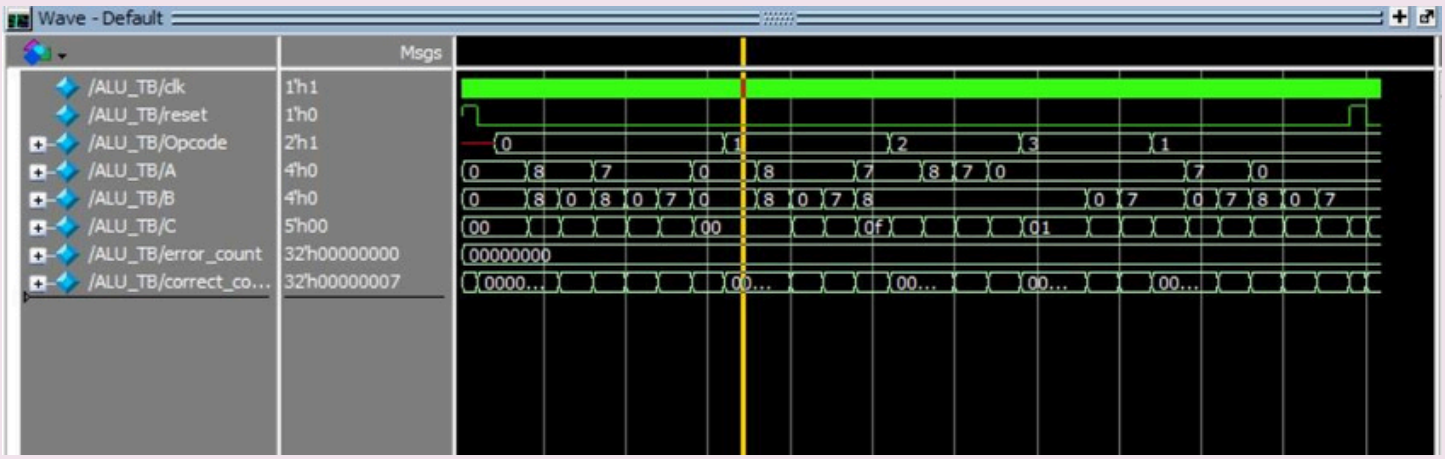
IV. Counters

```
# Correct count: 22, Error count: 0
```

V. Do File

```
vlib work
vlog ALU.v ALU_TB.sv  +cover -covercells
vsim -voptargs=+acc work.ALU_TB -cover
add wave *
coverage save ALU_TB.ucdb -onexit -du work.ALU
run -all
```

VI. Waveform



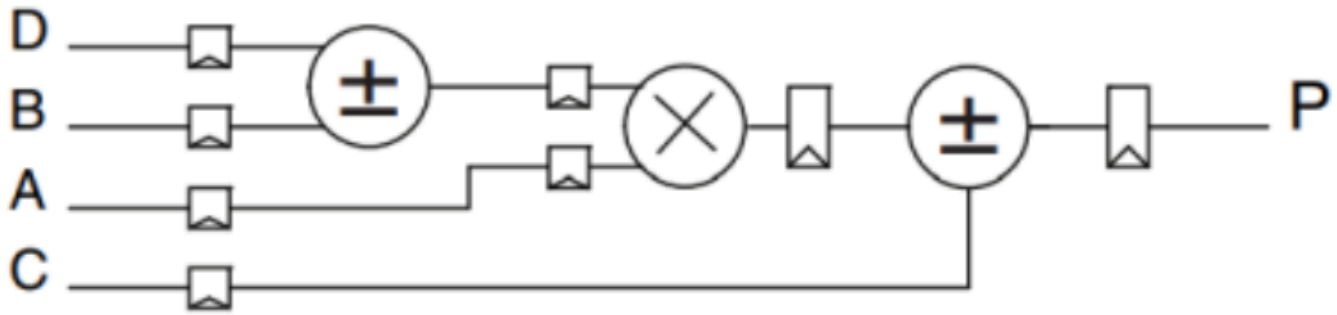
VII. Coverage Report

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         7         7         0   100.00%

================================Branch Details================================

Branch Coverage for instance /\ALU_TB#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ALU.v
------------------------------------CASE Branch------------------------------------
    20                                        26     Count coming in to CASE
    21             1                           7
    22             1                          11
    23             1                           4
    24             1                           3
    25             1                           1
Branch totals: 5 hits of 5 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    31                                        51     Count coming in to IF
    31             1                           4
    33             1                          47
Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       9         9         0   100.00%

================================Statement Details================================

Statement Coverage for instance /\ALU_TB#DUT  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ALU.v
    19             1                          26
    21             1                           7
    22             1                          11
    23             1                           4
    24             1                           3
    25             1                           1
    30             1                          51
    32             1                           4
    34             1                          47
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         44        44         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /\ALU_TB#DUT  --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                          A[0-3]           1           1      100.00
                                       Alu_out[4-0]        1           1      100.00
                                          B[0-3]           1           1      100.00
                                          C[4-0]           1           1      100.00
                                       Opcode[0-1]         1           1      100.00
                                          clk              1           1      100.00
                                        reset              1           1      100.00

Total Node Count     =         22
Toggled Node Count   =         22
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (44 of 44 bins)
```

Question 4

4) Verify the functionality of the following simplified version of the DSP block DSP48A1. All the registers are positive-edge triggered with the clock and have an async active low reset. Assume that we will only use the DSP in the addition mode only. For simplicity, you can drive the DSP inputs every 4 clock cycles to check the P output. Use a for loop and randomize the inputs after the reset checking.

Note: it is an unsigned DSP block.



Port Type	Width
A Input	18
B Input	18
C Input	48
D Input	18
Clk Input	1
rst_n (async active low) Input	1
P Output	48

Parameters:

1. OPERATION: take 2 values either “ADD” or “SUBTRACT”, Default value “ADD” ○ When subtracting use “D - B” and “multiplier_out – C”. multiplier_out is an internal signal

Question 4

I. Verification plan

Label	Description	Stimulus Generation	Functionality check
Testbench Setup	Initialize testbench signals and clock generation.	clk = 0; forever #1 clk = ~clk;	Verify that the clock signal is toggling correctly.
Reset Assertion	Assert and de-assert reset signal.	rst_n = 0; rst_n = 1;	Check that output P is zero when reset is asserted.
Error Count Init	Initialize error count.	error_count = 0;	Ensure the error count starts from 0.
Correct Count Init	Initialize correct count.	correct_count = 0;	Ensure the correct count starts from 0.
Test 1: ZERO Inputs	Test DSP operation with all inputs set to zero.	A = 0; B = 0; C = 0; D = 0;	Verify that P = 0.
Test 2: A = MAX, Others = 0	Test DSP operation with A at maximum and others at zero.	A = {18{1'b1}}; B = 0; C = 0; D = 0;	Verify that P = 0.
Test 3: B = MAX, Others = 0	Test DSP operation with B at maximum and others at zero.	A = 0; B = {18{1'b1}}; C = 0; D = 0;	Verify that P = 0.
Test 4: D = MAX, Others = 0	Test DSP operation with D at maximum and others at zero.	A = 0; B = 0; C = 0; D = {18{1'b1}};	Verify that P = 0.
Test 5: A, D = MAX, Others = 0	Test DSP operation with A and D at maximum, others at zero.	A = {18{1'b1}}; B = 0; C = 0; D = {18{1'b1}};	Verify that P = D * A.
Test 6: C = MAX, Others = 0	Test DSP operation with C at maximum and others at zero.	A = 0; B = 0; C = {48{1'b1}}; D = 0;	Verify that P = C.
Test 7: A, B = MAX, C, D = 0	Test DSP operation with A, B at maximum, C, D at zero.	A = {18{1'b1}}; B = {18{1'b1}}; C = 0; D = 0;	Verify that P = B * A.
Test 8: Specific Values 1	Test DSP operation with specific inputs A, B, C, D.	A = 18'h1234; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h1234;	Verify that P = ((D + B) * A) + C.
Test 9: Specific Values 2	Test DSP operation with specific inputs A, B, C, D.	A = 18'h1234; B = 18'h1234; C = 48'h0; D = 18'h1234;	Verify that P = ((D + B) * A) + C.
Test 10: Specific Values 3	Test DSP operation with specific inputs A, B, C, D.	A = 18'h0; B = 18'h0; C = 48'hABCDE1234567; D = 18'h0;	Verify that P = ((D + B) * A) + C.
Test 11: Specific Values 4	Test DSP operation with specific inputs A, B, C, D.	A = 18'h1234; B = 18'h0; C = 48'hABCDE1234567; D = 18'h1234;	Verify that P = ((D + B) * A) + C.
Test 12: Specific Values 5	Test DSP operation with specific inputs A, B, C, D.	A = 18'h0; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h1234;	Verify that P = ((D + B) * A) + C.
Test 13: Specific Values 6	Test DSP operation with specific inputs A, B, C, D.	A = 18'h1234; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h0;	Verify that P = ((D + B) * A) + C.
Randomized Test	Randomize inputs A, B, C, and D for multiple cycles.	Randomized A, B, C, and D values over 100 iterations.	Verify that P = ((D + B) * A) + C for each randomized input combination.
Error Reporting	Display the total number of errors at the end of the test.	$display("Error count: %0d", error_count);	Ensure all errors are logged and displayed.
Correct Reporting	Display the total number of correct results at the end of the test.	$display("Correct count: %0d", correct_count);	Ensure all correct results are logged and displayed.
Stop Simulation	End the simulation and display results.	$stop;	Ensure simulation halts and results are displayed.

II. RTL Design

```
module DSP(A, B, C, D, clk, rst_n, P);
parameter OPERATION = "ADD";
input  [17:0] A, B, D;
input  [47:0] C;
input clk, rst_n;
output reg  [47:0] P;

reg  [17:0] A_reg_stg1, A_reg_stg2, B_reg, D_reg, adder_out_stg1;
reg  [47:0] C_reg;
reg  [35:0] mult_out;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        // reset
         A_reg_stg1 <= 0;
         A_reg_stg2 <= 0;
         B_reg <= 0;
         D_reg <= 0;
                C_reg <= 0;
         adder_out_stg1 <= 0;
         mult_out <= 0;
         P <= 0;
    end
    else begin
        A_reg_stg1 <= A;
        A_reg_stg2 <= A_reg_stg1;
        B_reg <= B;
        C_reg <= C;
        D_reg <= D;
        if (OPERATION == "ADD") begin
            adder_out_stg1 <= D_reg + B_reg;
            P <= mult_out + C_reg;
        end
        else if (OPERATION == "SUBTRACT") begin
            adder_out_stg1 <= D_reg - B_reg;
            P <= mult_out - C_reg;
        end
        mult_out <= A_reg_stg2 * adder_out_stg1;
    end
end
endmodule
```

III. TestBench

```
module DSP_TB;
//PARAMETER
parameter OPERATION = "ADD";
localparam ZERO=0;
//SIGNAL DECLARATION
logic   [17:0] A, B, D;
logic   [47:0] C;
logic   clk, rst_n;
logic   [47:0] P;
//INSTATIATION
 DSP DUT(.A(A),.B(B),.C(C),.D(D),.clk(clk),.rst_n(rst_n),.P(P));
//COUNTER
 integer error_count;
 integer correct_count;
//CLOCK GENERATION
 initial begin
clk=0;
forever
#1 clk=~clk;
 end
//TEST
initial begin
error_count=0;
correct_count=0;
assert_reset;
A = ZERO; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
A = {18{1'b1}}; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
A = ZERO; B = {18{1'b1}}; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
A = ZERO; B = ZERO; C = ZERO; D = {18{1'b1}}; check_result(((D+ B) * A) + C);
A = {18{1'b1}}; B = ZERO; C = ZERO; D = {18{1'b1}}; check_result(((D + B) * A) + C);
A = ZERO; B = ZERO; C = {48{1'b1}}; D = ZERO; check_result(((D + B) * A) + C);
A = {18{1'b1}}; B = {18{1'b1}}; C = ZERO; D = ZERO; check_result(((D+ B) * A) + C);
A = ZERO; B = ZERO; C = ZERO; D = ZERO; check_result(((D + B) * A) + C);
// Test cases with specific numbers
A = 18'h1234; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h1234; check_result(((D + B) * A) + C);
A = 18'h1234; B = 18'h1234; C = 48'h0; D = 18'h1234; check_result(((D + B) * A) + C);
A = 18'h0; B = 18'h0; C = 48'hABCDE1234567; D = 18'h0; check_result(((D + B) * A) + C);
A = 18'h1234; B = 18'h0; C = 48'hABCDE1234567; D = 18'h1234; check_result(((D + B) * A) + C);
A = 18'h0; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h1234; check_result(((D + B) * A) + C);
A = 18'h1234; B = 18'h1234; C = 48'hABCDE1234567; D = 18'h0; check_result(((D + B) * A) + C);



for (int i = 0; i < 100; i++) begin
 // Randomize inputs
   A = $urandom_range(ZERO, 18'h1234);
   B = $urandom_range(ZERO, 18'h1234);
   C = $urandom_range(ZERO, 48'hABCDE1234567);
   D = $urandom_range(ZERO, 18'h1234);
check_result(((D+B) *A) +C);
end
assert_reset;
$display("correct Count:
$sEopgr Count:
@gdcorrect_count, error_count);

//TASK CHECHK
task check_result(input  [47:0] expected_result );
repeat (4) @(negedge clk);
if(P!== expected_result) begin
error_count++;
$display("INCORRECT P should equal 0d
ebdt is 0d
@]expeotedcteoult$P);
endtask
task assert_reset();
rst_n=0;
@(negedge clk );
if(P !== 0)begin
error_count++;
$display("0%t: Error: RESET VALUE IS ON AND OUTPUT IS NOT LOW",$time);
end
else
correct_count++;
rst_n=1;
endtask
endmodule

```

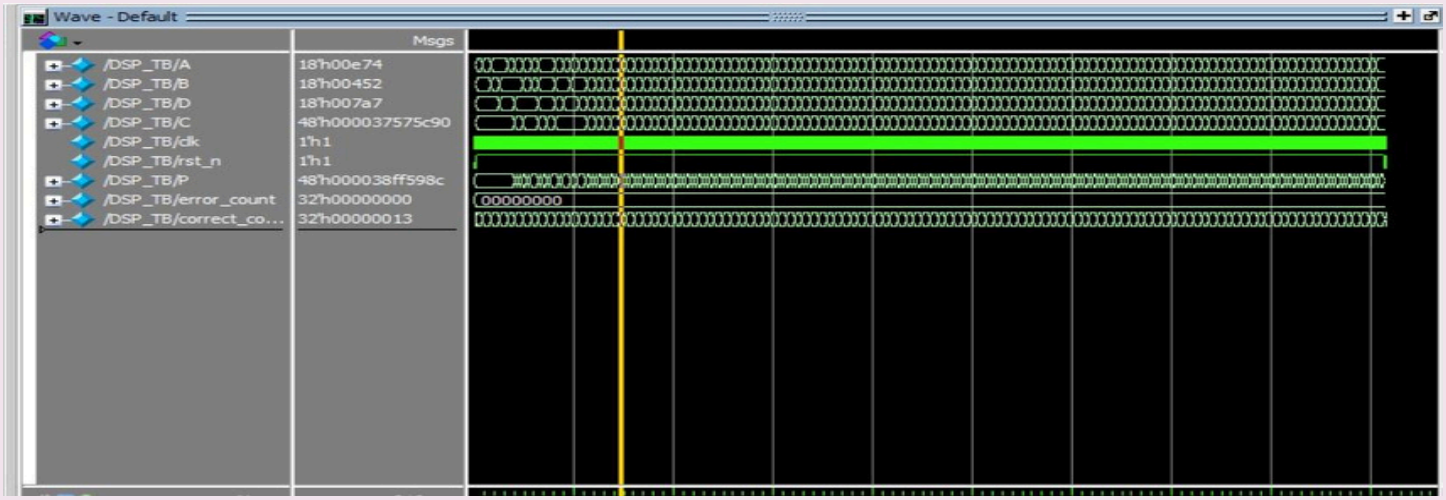
IV. Counters

```
# correct Count: 116, Error Count: 0
```

V. Do File

```
vlib work
vlog DSP.v DSP_TB.sv  +cover -covercells
vsim -voptargs=+acc work.DSP_TB -cover
add wave *
coverage save DSP_TB.ucdb -onexit -du work.DSP
run -all
```

VI. Waveform



VII. Coverage Report

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   100.00%
================================Branch Details================================

Branch Coverage for instance /\DSP_TB#DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File DSP.v
------------------------------------IF Branch------------------------------------
    13                                       452     Count coming in to IF
    13              1                          4
    24              1                        448
Branch totals: 2 hits of 2 branches = 100.00%


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      17        17         0   100.00%
============================Statement Details=============================
Statement Coverage for instance /\DSP_TB#DUT  --
    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File DSP.v
    12              1                        452
    15              1                          4
    16              1                          4
    17              1                          4
    18              1                          4
    19              1                          4
    20              1                          4
    21              1                          4
    22              1                          4
    25              1                        448
    26              1                        448
    27              1                        448
    28              1                        448
    29              1                        448
    31              1                        448
    32              1                        448
    38              1                        448
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        652       652         0   100.00%
================================Toggle Details================================

Toggle Coverage for instance /\DSP_TB#DUT  --
                                           Node      1H->0L      0L->1H  "Coverage"
                                           ----------------------------------------
                                           A[0-17]          1           1      100.00
                                  A_reg_stg1[17-0]          1           1      100.00
                                  A_reg_stg2[17-0]          1           1      100.00
                                           B[0-17]          1           1      100.00
                                       B_reg[17-0]          1           1      100.00
                                           C[0-47]          1           1      100.00
                                       C_reg[47-0]          1           1      100.00
                                           D[0-17]          1           1      100.00
                                       D_reg[17-0]          1           1      100.00
                                           P[47-0]          1           1      100.00
                             adder_out_stg1[17-0]          1           1      100.00
                                             clk          1           1      100.00
                                    mult_out[35-0]          1           1      100.00
                                           rst_n          1           1      100.00

Total Node Count     =        326
Toggled Node Count   =        326
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (652 of 652 bins)

Total Coverage By Instance (filtered view): 100.00%
```