يوسف أحمد محمد ابراهيم

Assignment 1

Question 1:

1. Design :

```
1   module adder (
2       input  clk,
3       input  reset,
4       input  signed [3:0] A, // Input data A in 2's complement
5       input  signed [3:0] B, // Input data B in 2's complement
6       output reg signed [4:0] C // Adder output in 2's complement
7           );
8
9      // Register output C
10     always @(posedge clk or posedge reset) begin
11        if (reset)
12          C <= 5'b0;
13        else
14          C <= A + B;
15     end
16
17  endmodule
```

2. verification plan :

A	B	C	D	E
Label	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
RESET_1	When the reset is asserted, output **C** should be low regardless value of **A, B.**	Directed at the start and the end of the simulation	-	**assert_reset()** task checks that output is 0 during reset.
ADDER_1	The sum of two maximum positive values sould be correct **(A = 7, B = 7, C = 14).**	Directed test	-	**check_result(14)** verifies correct addition.
ADDER_2	The sum maximum positive value and maximum negative value should be correct **(A = 7, B = -8, C = -1).**	Directed test	-	**check_result(-1)** verifies correct addition.
ADDER_3	The sum maximum negative value and maximum positive value should be correct **(A = -8, B = 7, C = -1).**	Directed test	-	**check_result(-1)** verifies correct addition.
ADDER_4	The sum of 2 maximum negative values should be correct **(A = -8, B = -8, C = -16).**	Directed test	-	**check_result(-16)** verifies correct addition.
ADDER_5	The sum of 0 and maximum positive value should be correct **(A = 0, B = 7, C = 7).**	Directed test	-	**check_result(7)** verifies correct addition.
ADDER_6	The sum of maximum positive value and 0 should be correct **(A = 7, B = 0, C = 7).**	Directed test	-	**check_result(7)** verifies correct addition.
ADDER_7	The sum of 2 **zeroes** should be correct **(A = 0, B = 0, C = 0).**	Directed test	-	**check_result(0)** verifies correct addition.
ADDER_8	The sum of 0 and maximum negative value should be correct **(A = 0, B = -8, C = -8).**	Directed test	-	**check_result(-8)** verifies correct addition.
ADDER_9	The sum of maximum negative value and 0 should be correct **(A = -8, B = 0, C = -8).**	Directed test	-	**check_result(-8)** verifies correct addition.

3. Testbench :

```
module adder_tb();
    bit signed [3:0]   A;
    bit signed [3:0]   B;
    bit signed [4:0]   C;
    logic       clk;
    logic       rst;

    integer error_count, correct_count;

    localparam MAXPOS = 4'b0111;
    localparam MAXNEG = 4'b1000;

    adder dut(clk, rst, A, B, C);

    initial begin
        clk = 0;
        forever
            #1 clk = ~clk;
    end

    initial begin
        // reset test
        rst = 1;
        A = 0;
        B = 0;
        error_count = 0;
        correct_count = 0;
        assert_reset();

        //TEST1
        A = MAXPOS;
        B = MAXPOS;
        check_result(14);
```

3. Testbench :

```
35          //TEST2
36          A = MAXPOS;
37          B = MAXNEG;
38          check_result(-1);
39
40          //TEST3
41          A = MAXNEG;
42          B = MAXPOS;
43          check_result(-1);
44
45          //TEST4
46          A = MAXNEG;
47          B = MAXNEG;
48          check_result(-16);
49
50          //TEST5
51          A = 0;
52          B = MAXPOS;
53          check_result(7);
54
55          //TEST6
56          A = MAXPOS;
57          B = 0;
58          check_result(7);
59
60          //TEST7
61          A = 0;
62          B = 0;
63          check_result(0);
64
65          //TEST8
66          A = 0;
67          B = MAXNEG;
68          check_result(-8);
```

3. Testbench :

```
70          //TEST9
71          A = MAXNEG;
72          B = 0;
73          check_result(-8);
74
75          assert_reset();
76
77          $display("*** errors: %d, success: %d ***", error_count, correct_count);
78          $stop;
79      end
80
81
82      task assert_reset();
83          rst = 1;
84          check_result(0);
85          rst = 0;
86      endtask
87
88      task check_result(logic signed [4:0] C_exp);
89          @(negedge clk);
90          if(C != C_exp) begin
91              $display ("*** ERROR, A = %d, B = %d, C = %d", A, B, C, );
92              error_count = error_count + 1;
93          end
94          else
95              correct_count = correct_count + 1;
96      endtask
97  endmodule
```

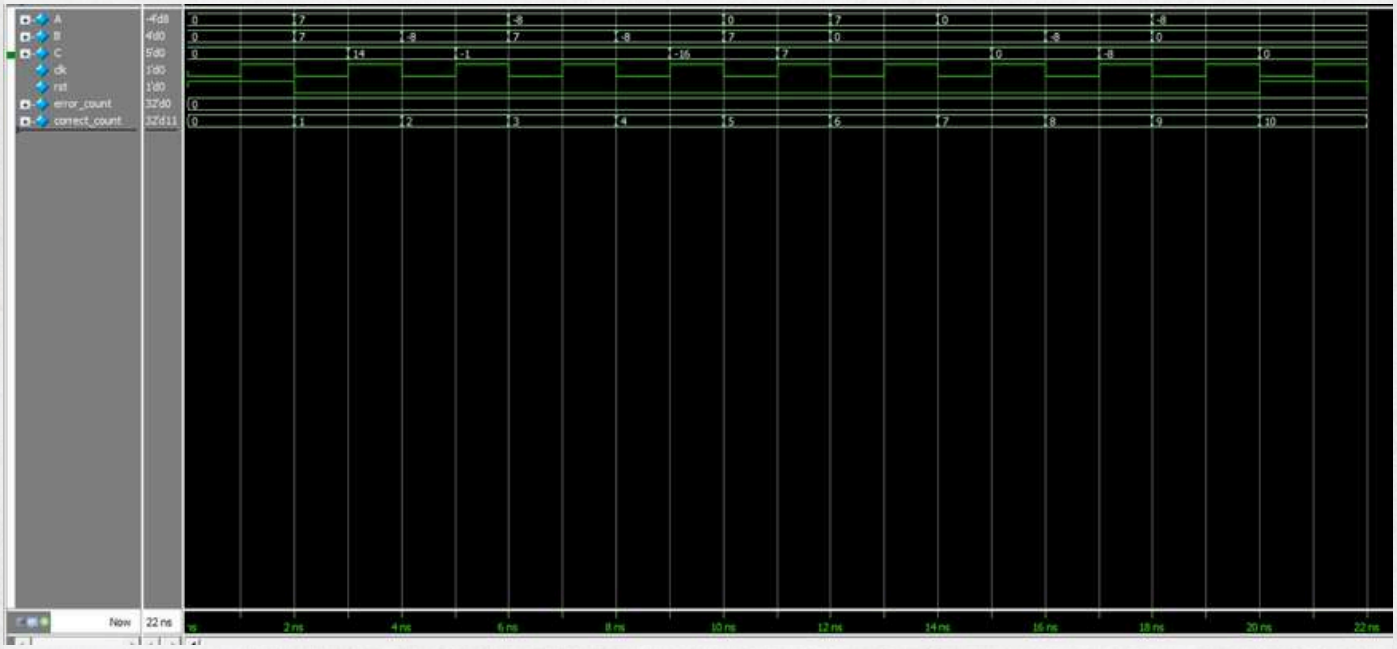
4. Do file :

```
1  vlib work
2  vlog adder.v 1_adder_tb.sv  +cover -covercells
3  vsim -voptargs=+acc work.adder_tb -cover
4  add wave *
5  coverage save adder_tb.ucdb -onexit -du work.adder
6  run -all
```

4. Do file :

```
1  vlib work
2  vlog adder.v 1_adder_tb.sv  +cover -covercells
3  vsim -voptargs=+acc work.adder_tb -cover
4  add wave *
5  coverage save adder_tb.ucdb -onexit -du work.adder
6  run -all
```

5. Qesta sim wave snippets (decimal radix) :




```
# *** errors: 0, success: 11 ***
# ** Note: $stop    : 1_adder_tb.sv(78)
#    Time: 22 ns  Iteration: 1  Instance: /adder_tb
# Break in Module adder_tb at 1_adder_tb.sv line 78
```

6. Statement coverage report :

```
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
    1                                               module adder (

    2                                                   input  clk,

    3                                                   input  reset,

    4                                                   input  signed [3:0] A, // Input data A in 2's complement

    5                                                   input  signed [3:0] B, // Input data B in 2's complement

    6                                                   output reg signed [4:0] C // Adder output in 2's complement

    7                                                    );

    8

    9                                                  // Register output C

    10             1                        13         always @(posedge clk or posedge reset) begin

    11                                                    if (reset)

    12             1                         4            C <= 5'b0;

    13                                                    else

    14             1                         9            C <= A + B;


Toggle Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Toggles                        30        30         0   100.00%
```

7. Branch 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           if (reset)

    13              1                          9           else

Branch totals: 2 hits of 2 branches = 100.00%
```

7. Toggle coverage report :

```
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:

1. RTL code (after correction) :

```
1   module priority_enc (
2   input              clk,
3   input              rst,
4   input       [3:0] D,
5   output reg  [1:0] Y,
6   output reg        valid
7   );
8
9   always @(posedge clk) begin
10    if (rst) begin
11      Y       <= 2'b0;
12      valid   <= 0;
13    end
14    else begin
15      casex (D)
16          4'b1000: Y <= 0;
17          4'bX100: Y <= 1;
18          4'bXX10: Y <= 2;
19          4'bXXX1: Y <= 3;
20          4'b0000: Y <= 2'bxx;
21      endcase
22      valid <= (~|D)? 1'b0: 1'b1;
23    end
24  end
25  endmodule
```

- **Errors fixed:**
  a. Missing reg datatype for the ouputs and using them in always block. Cased a combilation error.
  b. Uninitialized **valid** at reset assertion. Caused the **valid** signal to remain at 1'bx.

2. Verification plan :

	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
2	RESET	When the reset is asserted, outputs **Y** and **valid** should be low regardless value of **D.**	**assert_reset()** task that asserts rst to1, wait then deassert rst to 0 at the begining and end of the simulation.	-	**assert_reset()** task checks that output is 0 during reset.
3	TEST_1	output is correct at any value of **D.**	Exaustive test to check all possible input combinations.	-	**check_result()** verifies correct output.

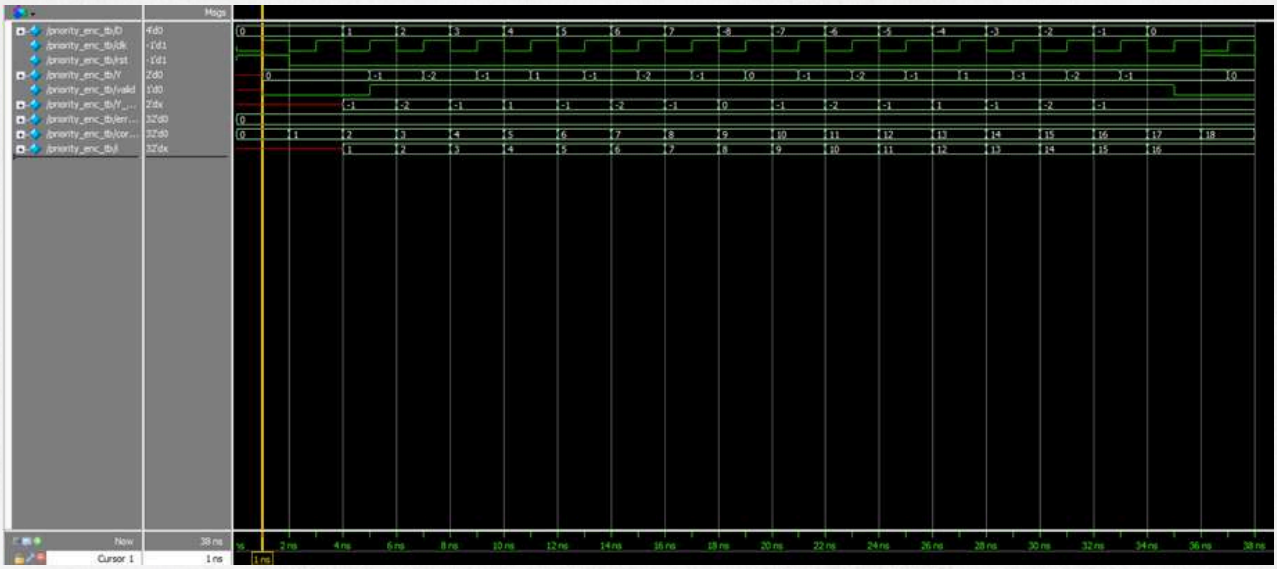
3. Testbench :

```
module priority_enc_tb();
    logic [3:0] D;
    logic       clk;
    logic       rst;
    logic [1:0] Y;
    logic       valid;


    logic [1:0] Y_exp;
    integer error_count, correct_count;

    priority_enc dut(clk, rst, D, Y, valid);

    initial begin
        clk = 0;
        forever
            #1 clk = ~clk;
    end

    integer i;
    initial begin
        //RESET test
        D = 0;
        error_count = 0;
        correct_count = 0;
        assert_reset();

        //TESET_1
        check_result(2'bxx, 0);
        for(i = 1; i < 16; i = i+ 1) begin
            D = i;
            casex (D)
                4'b1000: Y_exp = 0;
                4'bX100: Y_exp = 1;
                4'bXX10: Y_exp = 2;
                4'bXXX1: Y_exp = 3;
            endcase
            check_result(Y_exp, 1);
        end
        D = 0;
        check_result(2'bxx, 0);
        assert_reset();

        $display("*** errors: %0d, success: %0d ***", error_count, correct_count);
        $stop;
    end

    task assert_reset();
        rst = 1;
        check_result(0, 0);
        rst = 0;
    endtask

    task check_result(logic [1:0]Y_exp, logic valid_exp);
        @(negedge clk);
        if(Y != Y_exp || valid != valid_exp) begin
            error_count = error_count +1;
            $display ("*** ERROR! D = %0d, rst = %0d, Y = %0d, valid = %0d ***", D, rst, Y, valid);
        end
        else
            correct_count = correct_count +1;
    endtask

endmodule
```

4. Do file :

```
vlib work
vlog 2_priority_enc.v 2_priority_enc_tb.sv  +cover -covercells
vsim -voptargs=+acc work.priority_enc_tb -cover
add wave *
coverage save priority_enc_tb.ucdb -onexit -du work.priority_enc
run -all
```

5. Questasim snippets (decimal radix) :



```
# ** errors: 0, success: 19 ***
# ** Note: $stop    : 2_priority_enc_tb.sv(45)
#    Time: 38 ns  Iteration: 1  Instance: /priority_enc_tb
# Break in Module priority_enc_tb at 2_priority_enc_tb.sv line 45
```

6. Statement coverage report :

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       8         8         0   100.00%

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

Statement Coverage for instance /\priority_enc_tb#dut  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File 2_priority_enc.v
    9               1                         19
    11              1                          2
    12              1                          2
    16              1                          1
    17              1                          2
    18              1                          4
    19              1                          8
    21              1                         17
```

7. Branch coverage report :

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

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

Branch Coverage for instance /\priority_enc_tb#dut

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File 2_priority_enc.v
------------------------------------IF Branch------------------------------------
    10                                       19     Count coming in to IF
    10              1                         2
    14              1                        17
Branch totals: 2 hits of 2 branches = 100.00%

----------------------------------CASE Branch-----------------------------------
    15                                       17     Count coming in to CASE
    16              1                         1
    17              1                         2
    18              1                         4
    19              1                         8
                                              2     All False Count
Branch totals: 5 hits of 5 branches = 100.00%
```

8. Toggle coverage report :

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         18        18         0   100.00%

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

Toggle Coverage for instance /\priority_enc_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:

1. RTL code :

```
module ALU_4_bit (
    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
```

2. Verification plan :

1	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
2	RESET_TEST	When the reset is asserted, outputs **C** should be low.	**assert_reset()** task that asserts rst to1, wait then deassert rst to 0 at the begining and end of the simulation.	-	**check_result()** task checks that output is 0 during reset.
3	ADD_TEST	Output **C** should be equal to the sum of inputs **A** and **B** at the extreme boundary values of the 4-bit signed range.	Directed test, drive all combinations of inputs at extreme values (MAXPOS, MAXNEG) and 0;	-	**check_result()** verifies correct output.
4	SUB_TEST	Output **C** should be equal to the substraction of inputs **A** and **B** at the extreme boundary values values of the 4-bit signed range.	Directed test, drive all combinations of inputs at extreme values (MAXPOS, MAXNEG) and 0;		**check_result()** verifies correct output.
5	INVERT_TEST	Output **C** should be equal to the bitwise inversion of the input **A** at extreme boundary values and one-hot combinations.	Directed test, drive the input **A** at extreme values (4'b1111, 4'b0000) and all one-hot values.		**check_result()** verifies correct output.
6	RED_OR_TEST	Output C should be equal to the reduction or of the input **B** at extreme boundary values and one-hot combinations.	Directed test, drive the input **B** at extreme values (4'b1111, 4'b0000) and all one-hot values.		**check_result()** verifies correct output.

3. Testbench :

```
module ALU_tb();
    logic                clk;
    logic                reset;
    logic        [1:0]   Opcode;
    logic signed [3:0]   A;
    logic signed [3:0]   B;
    logic signed [4:0] C;

    integer correct_count, error_count;

    localparam MAXPOS = 4'b0111;
    localparam MAXNEG = 4'b1000;

    ALU_4_bit dut(clk, reset, Opcode, A, B, C);

    initial begin
        clk = 0;
        forever
            #1 clk = ~clk;
    end

    integer i;
    initial begin
        correct_count = 0;
        error_count = 0;
        Opcode  = 0;
        A       = 0;
        B       = 0;
        //reset test
        assert_reset();

```

```
//addition test
Opcode = 0;

A = MAXPOS;
B = MAXPOS;
check_result(14);

A = MAXPOS;
B = MAXNEG;
check_result(-1);

A = MAXPOS;
B = 0;
check_result(7);

A = MAXNEG;
B = MAXPOS;
check_result(-1);

A = MAXNEG;
B = MAXNEG;
check_result(-16);

A = MAXNEG;
B = 0;
check_result(-8);

A = 0;
B = MAXPOS;
check_result(7);

A = 0;
B = MAXNEG;
check_result(-8);

A = 0;
B = 0;
check_result(0);
```

```
//substraction test
Opcode = 1;

A = MAXPOS;
B = MAXPOS;
check_result(0);

A = MAXPOS;
B = MAXNEG;
check_result(15);

A = MAXPOS;
B = 0;
check_result(7);

A = MAXNEG;
B = MAXPOS;
check_result(-15);

A = MAXNEG;
B = MAXNEG;
check_result(0);

A = MAXNEG;
B = 0;
check_result(-8);

A = 0;
B = MAXPOS;
check_result(-7);

A = 0;
B = MAXNEG;
check_result(8);

A = 0;
B = 0;
check_result(0);
```

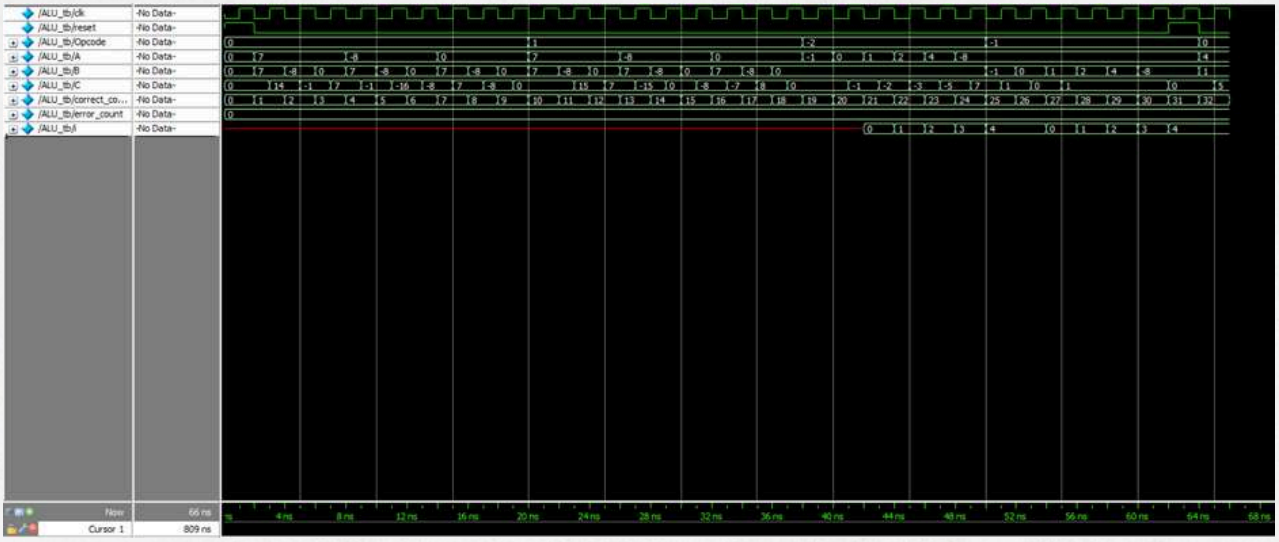
```
    //bitwise invertion test
    Opcode = 2;

    A = 4'b1111;
    check_result(0);

    A = 4'b0000;
    check_result(-1);

    for(i = 0; i < 4; i = i + 1) begin
        A    = 0;
        A[i] = 1;
        check_result(~A);
    end


    //red_OR test
    Opcode = 3;

    B = 4'b1111;
    check_result(1);

    B = 4'b0000;
    check_result(0);

    for(i = 0; i < 4; i = i + 1) begin
        B    = 0;
        B[i] = 1;
        check_result(|B);
    end


    //reset test
    assert_reset();

    //set_opcodee back to 0
    Opcode = 0;
    A = $random;
    B = $random;
    check_result($signed(A + B));

    $display("*** errors: %0d, success: %0d ***", error_count, correct_count);
    $stop;
end
```

```
    task assert_reset();
        reset = 1;
        check_result(0);
        reset = 0;
    endtask

    task check_result(logic signed [4:0] C_exp);
        @(negedge clk);
        if(C != C_exp) begin
            $display ("*** ERROR, A = %d, B = %d, C = %d, OPCODE = %d ***", A, B, C, Opcode);
            error_count = error_count + 1;
        end
        else
            correct_count = correct_count + 1;
    endtask
endmodule
```

4. Do file :

```
1   vlib work
2   vlog 3_ALU.v 3_ALU_tb.sv  +cover -covercells
3   vsim -voptargs=+acc work.ALU_tb -cover
4   add wave *
5   coverage save 3_ALU_tb.ucdb -onexit -du work.ALU_4_bit
6   coverage exclude -src 3_ALU.v -scope /ALU_tb/dut -line 26 -code b
7   coverage exclude -src 3_ALU.v -scope /ALU_tb/dut -line 26 -code s
8   run -all
```

5. Questasim snippets (decimal radix) :




```
# *** errors: 0, success: 33 ***
# ** Note: $stop    : 3_ALU_tb.sv(154)
#    Time: 66 ns  Iteration: 1  Instance: /ALU_tb
# Break in Module ALU_tb at 3_ALU_tb.sv line 154
```

I have Excluded the defalt case from branch and statement coverage as it will never be reached becaue all possible cases are writen.

6. Statement coverage report :

```
Statement Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Statements                      8         8         0   100.00%

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

Statement Coverage for instance /\ALU_tb#dut  --

    Line         Item                    Count     Source
    ----         ----                    -----     ------
  File 3_ALU.v
    1                                              module ALU_4_bit (
    2                                                  input  clk,
    3                                                  input  reset,
    4                                                  input  [1:0] Opcode,   // The opcode
    5                                                  input  signed [3:0] A, // Input data A in 2's complement
    6                                                  input  signed [3:0] B, // Input data B in 2's complement
    7
    8                                                  output reg signed [4:0] C // ALU output in 2's complement
    9
    10                                                     );
    11
    12                                             reg signed [4:0]       Alu_out; // ALU output in 2's complement
    13
    14                                             localparam         Add           = 2'b00; // A + B
    15                                             localparam         Sub           = 2'b01; // A - B
    16                                             localparam         Not_A           = 2'b10; // ~A
    17                                             localparam         ReductionOR_B  = 2'b11; // |B
    18
    19                                             // Do the operation
    20              1                         32       always @* begin
    21                                                 case (Opcode)
    22              1                         11           Add:            Alu_out = A + B;
    23              1                          9           Sub:            Alu_out = A - B;
    24              1                          6           Not_A:          Alu_out = ~A;
    25              1                          6           ReductionOR_B:  Alu_out = |B;
    26                                                      default:  Alu_out = 5'b0;
    27                                                 endcase
    28                                             end // always @ *
    29
    30                                             // Register output C
    31              1                         33       always @(posedge clk or posedge reset) begin
    32                                                 if (reset)
    33              1                          4           C <= 5'b0;
    34                                                 else
    35              1                         29           C<= Alu_out;
```

7. Branch coverage report :

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         6         6         0   100.00%

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

Branch Coverage for instance /\ALU_tb#dut

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File 3_ALU.v
------------------------------------CASE Branch------------------------------------
    21                                       32     Count coming in to CASE
    22              1                        11            Add:            Alu_out = A + B;

    23              1                         9            Sub:            Alu_out = A - B;

    24              1                         6            Not_A:          Alu_out = ~A;

    25              1                         6            ReductionOR_B:  Alu_out = |B;

Branch totals: 4 hits of 4 branches = 100.00%

-----------------------------------IF Branch------------------------------------
    32                                       33     Count coming in to IF
    32              1                         4           if (reset)

    34              1                        29           else

Branch totals: 2 hits of 2 branches = 100.00%
```

8. Toggle coverage report :

```
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:

1. RTL code :

```
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;
reg  [18:0] adder_out_stg1;
reg  [47:0] C_reg, 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
```

- **Errors fixed:**
    a. **adder_out2** register has no benefit, the correct design should have one pipeline stage between adder and mult,
    b. **C_reg** wasnt't initialized when reset is on.
    c. **adder_out1** signal is 18 bit size. It should be 19 bit width to avoid overflow case.

2. Verification plan :

	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
2	RESET_TEST	When the reset is asserted, outputs **C** should be low.	**assert_reset()** task that asserts rst to1, wait then deassert rst to 0 at the begining and end of the simulation.	-	**check_result()** task checks that output is 0 during reset.
3	ADD_TEST	Output **P** should be equal to **(((B+D)*A)+C)**.	randomized values for **A, B, C, D** every 4 cycle.	-	**check_result()** checks the value of the output every 4cycle using a golden model.

3. Testbench :

```
1   module DSP_tb();
2
3       logic [17:0] A, B, D;
4       logic [47:0] C;
5       logic        clk;
6       logic        rst_n;
7       logic [47:0] P;
8       logic [47:0] P_exp;
9
10      integer correct_count, error_count;
11
12      DSP dut(A, B, C, D, clk, rst_n, P);
13
14      initial begin
15          clk = 0;
16          forever
17              #1 clk = ~clk;
18      end
19
20      initial begin
21          A = 0;
22          B = 0;
23          D = 0;
24          C = 0;
25          correct_count   = 0;
26          error_count     = 0;
27
28          //Reset check
29          rst_n = 0;
30          check_result();
31          rst_n = 1;
32
33          //ADD test
34          repeat(20) begin
35              A = $random;
36              B = $random;
37              D = $random;
38              C = {$random, $random};
39              check_result();
40          end
```

```
        //Reset check
        rst_n = 0;
        check_result();
        rst_n = 1;

        $display("*** errors: %0d, success: %0d ***", error_count, correct_count);
        $stop;
    end

    task check_result();
        repeat(4) @(negedge clk);
        if(!rst_n)
            P_exp = 0;
        else
            P_exp = (((B+D)*A)+C);

        if(P != P_exp) begin
            $display("*** ERROR! B = %d D = %d A = %d C = %d P = %d, Expected: %d ***", B, D, A, C, P, P_exp);
            error_count = error_count + 1;
        end
        else
            correct_count = correct_count + 1;
    endtask
endmodule
```

4. Do file :

```
vlib work
vlog 4_DSP.v 4_DSP_tb.sv  +cover -covercells
vsim -voptargs=+acc work.DSP_tb -cover
add wave *
coverage save 4_DSP_tb.ucdb -onexit -du work.DSP
coverage exclude -du DSP -togglenode C
coverage exclude -du DSP -togglenode {mult_out[37]}
coverage exclude -du DSP -togglenode {mult_out[38]}
coverage exclude -du DSP -togglenode {mult_out[39]}
coverage exclude -du DSP -togglenode {mult_out[40]}
coverage exclude -du DSP -togglenode {mult_out[41]}
coverage exclude -du DSP -togglenode {mult_out[42]}
coverage exclude -du DSP -togglenode {mult_out[43]}
coverage exclude -du DSP -togglenode {mult_out[44]}
coverage exclude -du DSP -togglenode {mult_out[45]}
coverage exclude -du DSP -togglenode {mult_out[46]}
coverage exclude -du DSP -togglenode {mult_out[47]}
run -all
```

5. Questasim snippets (hex radix) :

```
#           Using alternate file: ./wlfti58bde
# *** errors: 0, success: 22 ***
# ** Note: $stop    : 4_DSP_tb.sv(48)
#    Time: 176 ns  Iteration: 1  Instance: /DSP_tb
# Break in Module DSP_tb at 4_DSP_tb.sv line 48
```

*I excluded **mult_out[36:47]** from the coverage report becaue they area always equal 0;*

6. Statement coverage report :

```
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 4_DSP.v
    1                                                module DSP(A, B, C, D, clk, rst_n, P);

    2                                                parameter OPERATION = "ADD";

    3                                                input  [17:0] A, B, D;

    4                                                input  [47:0] C;

    5                                                input clk, rst_n;

    6                                                output reg  [47:0] P;

    7

    8                                                reg  [17:0] A_reg_stg1, A_reg_stg2, B_reg, D_reg;

    9                                                reg  [18:0] adder_out_stg1;

    10                                               reg  [47:0] C_reg, mult_out;

    11

    12             1                          84     always @(posedge clk or negedge rst_n) begin

    13                                                   if (!rst_n) begin

    14                                                       // reset

    15             1                           4            A_reg_stg1 <= 0;

    16             1                           4            A_reg_stg2 <= 0;

    17             1                           4            B_reg <= 0;

    18             1                           4            D_reg <= 0;

    19             1                           4            C_reg <= 0;
```

```
77
78      20          1                   4           adder_out_stg1 <= 0;
79
80      21          1                   4           mult_out <= 0;
81
82      22          1                   4           P <= 0;
83
84      23                                      end
85
86      24                                      else begin
87
88      25          1                   80          A_reg_stg1 <= A;
89
90      26          1                   80          A_reg_stg2 <= A_reg_stg1;
91
92      27          1                   80          B_reg <= B;
93
94      28          1                   80          C_reg <= C;
95
96      29          1                   80          D_reg <= D;
97
98      30                                          if (OPERATION == "ADD") begin
99
100     31          1                   80              adder_out_stg1 <= D_reg + B_reg;
101
102     32          1                   80              P <= mult_out + C_reg;
103
104     33                                          end
105
106     34                                          else if (OPERATION == "SUBTRACT") begin
107
108     35                                              adder_out_stg1 <= D_reg - B_reg;
109
110     36                                              P <= mult_out - C_reg;
111
112     37                                          end
113
114     38          1                   80          mult_out <= A_reg_stg2 * adder_out_stg1;
115
```

7. Branch 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 4_DSP.v
------------------------------------IF Branch------------------------------------
    13                                        84     Count coming in to IF
    13              1                          4        if (!rst_n) begin

    24              1                         80        else begin

Branch totals: 2 hits of 2 branches = 100.00%
```

8. Toggle coverage report :

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        560       560         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_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[18-0]     1           1      100.00
                                                     clk          1           1      100.00
                                            mult_out[36-0]        1           1      100.00
                                                   rst_n          1           1      100.00

Total Node Count     =        280
Toggled Node Count   =        280
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (560 of 560 bins)


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