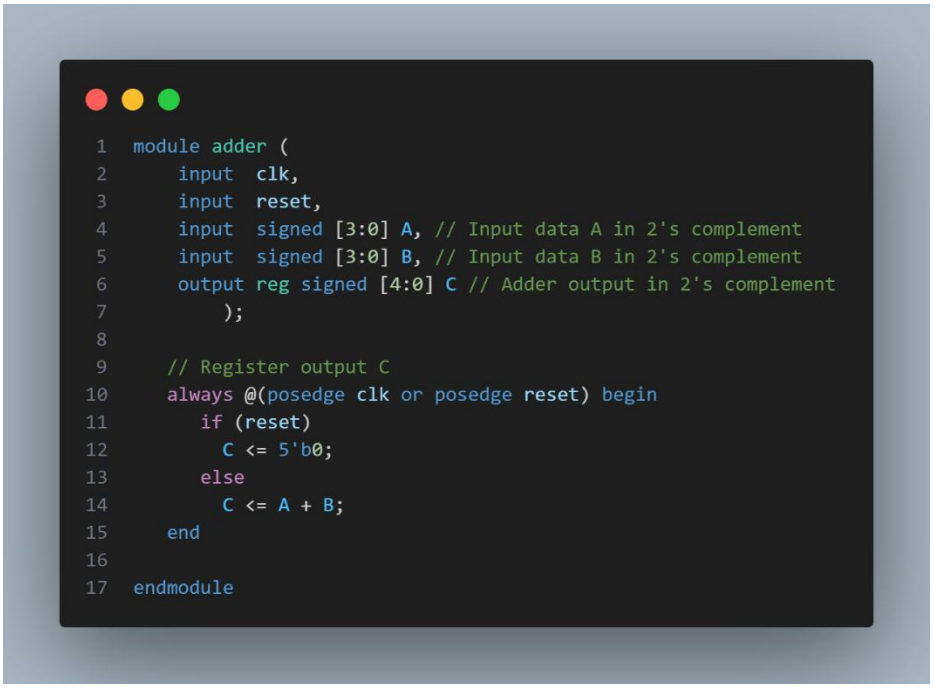
P1)

1. 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

endmodule
```

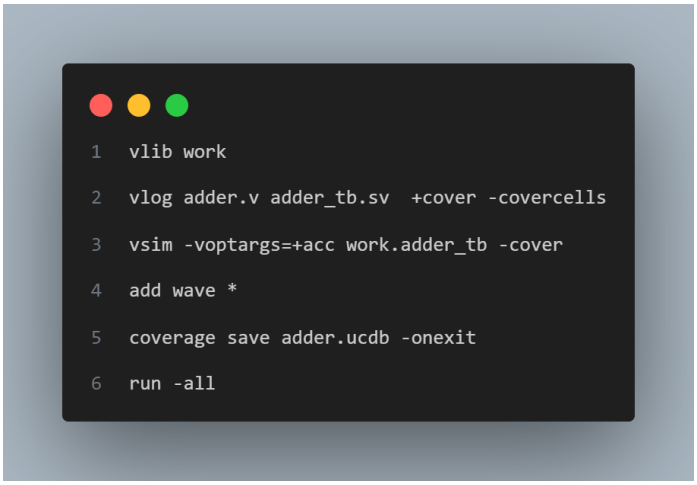
2. Verification Plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
TEST_2	Assign A with MAXPOS value (7) & B with zero , the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_3	Assign A with MAXPOS value (7) & B with MAXPOS , the output should be (14)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_4	Assign A with MAXPOS value (7) & B with MAXNEG , the output should be (-1)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_5	Assign A with MAXNEG value (-8) & B with zero , the output should be (-8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_6	Assign A with MAXNEG value (-8) & B with MAXPOS , the output should be (-1)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_7	Assign A with MAXNEG value (-8) & B with MAXNEG , the output should be (-16)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_8	Assign A with zero & B with zero , the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_9	Assign A with zero & B with MAXPOS , the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_10	Assign A with zero & B with MAXNEG , the output should be (-8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_11	Assign A with MAXPOS value (7) & B with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_12	Assign A with MAXNEG value (-8) & B with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_13	Assign A with zero & B with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_14	Assign B with MAXPOS value (7) & A with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_15	Assign B with MAXNEG value (-8) & A with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_16	Assign B with zero & A with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_17	Assign A with 10 random value & B with 10 random values , the output should be the sum of A & B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_18	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

3. Testbench

```
module adder_tb();
    reg  clk;
    reg  reset;
    reg  signed [3:0] A;
    reg  signed [3:0] B;
    wire signed [4:0] C;

    integer error_count , correct_count ;

    localparam MAXPOS = 7 ;
    localparam MAXNEG = -8 ;

    adder DUT (clk,reset,A,B,C) ;

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

    initial begin
        A = 0 ;
        B = 4 ;
        reset = 0 ;
        error_count = 0 ;
        correct_count = 0 ;
        // TEST_1
        assert_reset () ;
        // TEST_2
        A = MAXPOS ;
        B = 0 ;
        check_result (7) ;
        // TEST_3
        B = MAXPOS ;
        check_result (14) ;
        // TEST_4
        B = MAXNEG ;
        check_result (-1) ;
        // TEST_5
        A = MAXNEG;
        B = 0 ;
        check_result (-8) ;
        // TEST_6
        B = MAXPOS ;
        check_result (-1) ;
        // TEST_7
        B = MAXNEG ;
        check_result (-16) ;
        // TEST_8
        A = 0 ;
        B = 0 ;
        check_result (0) ;
        // TEST_9
        B = MAXPOS ;
        check_result (7) ;
        // TEST_10
        B = MAXNEG ;
        check_result (-8) ;
        // TEST_11
        repeat (10) begin
            A = MAXPOS ;
            B = $random;
            check_result (A+B) ;
        end
        // TEST_12
        repeat (10) begin
            A = MAXNEG ;
            B = $random;
            check_result (A+B) ;
        end
        // TEST_13
        repeat (10) begin
            A = 0 ;
            B = $random;
            check_result (A+B) ;
        end
        // TEST_14
        repeat (10) begin
            A = $random ;
            B = MAXPOS;
            check_result (A+B) ;
        end
        // TEST_15
        repeat (10) begin
            A = $random ;
            B = MAXNEG;
            check_result (A+B) ;
        end
        // TEST_16
        repeat (10) begin
            A = $random ;
            B = 0;
            check_result (A+B) ;
        end
        // TEST_17
        repeat (10) begin
            A = $random ;
            B = $random ;
            check_result (A+B) ;
        end
        // TEST_18
        assert_reset () ;

        $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
        $stop ;
    end

    task check_result (input signed [4:0] expected_out);
        @ (negedge clk) ;
        if (C == expected_out) begin
            $display ("test passes") ;
            correct_count = correct_count + 1 ;
        end
        else begin
           $display ("test fail ") ;
            error_count = error_count + 1 ;
        end
    endtask

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

endmodule
```

4. DO file



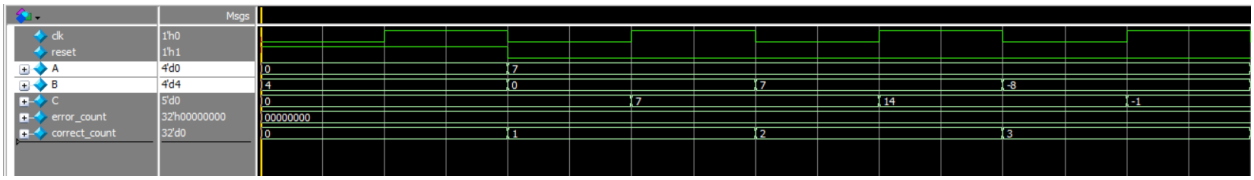
```
vlib work
vlog adder.v adder_tb.sv  +cover -covercells
vsim -voptargs=+acc work.adder_tb -cover
add wave *
coverage save adder.ucdb -onexit
run -all
```

5. Coverage report

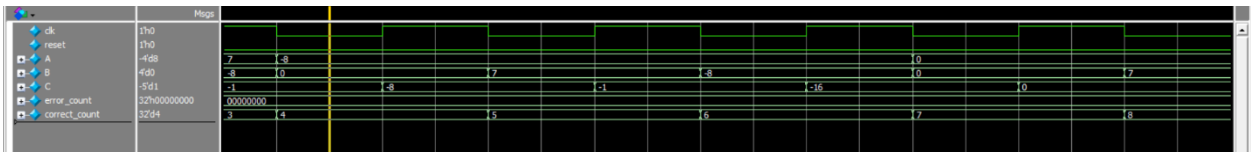
```
# ================================================================================
# === Instance: /\adder_tb#DUT
# === Design Unit: work.adder
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    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                                        83     Count coming in to IF
#     11              1                          4           if (reset)
#
#     13              1                         79           else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                       3         3         0   100.00%
#
```

```
#
# 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)
#
```

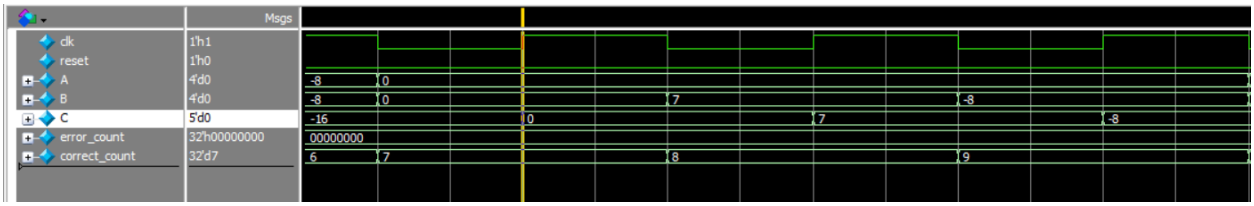
6. Questasim snippets
    - Test 1 && Test_2 && Test_3 && Test_4



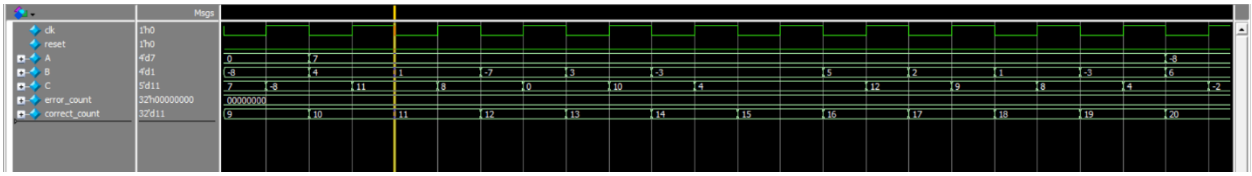
    - Test_5 && Test_6 && Test_7



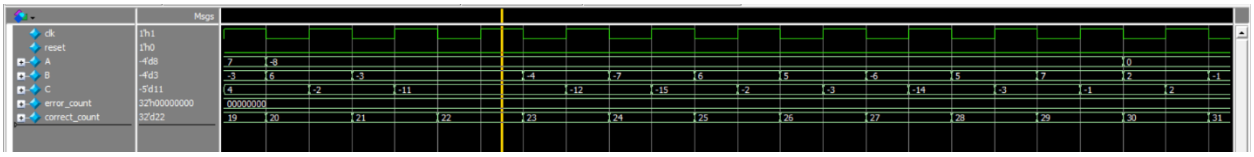
    - Test_8 && Test_9 && Test_10



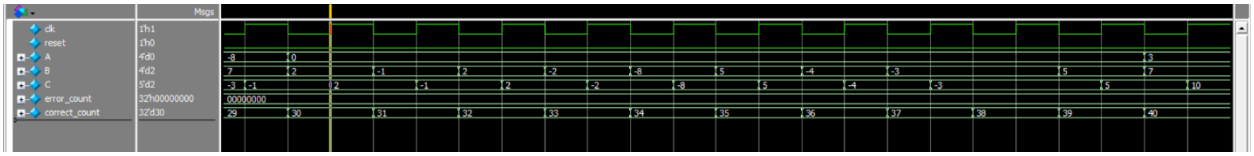
    - Test_11



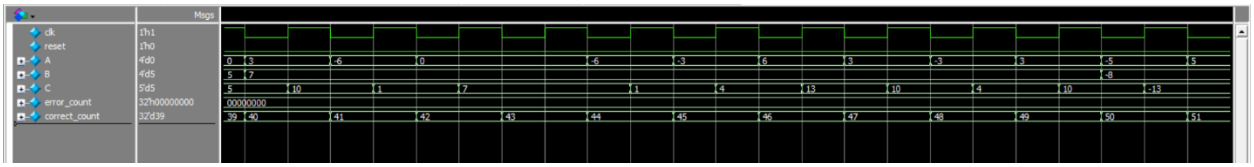
- Test_12



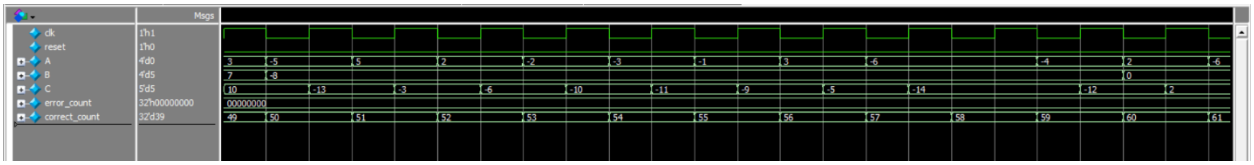
- Test_13



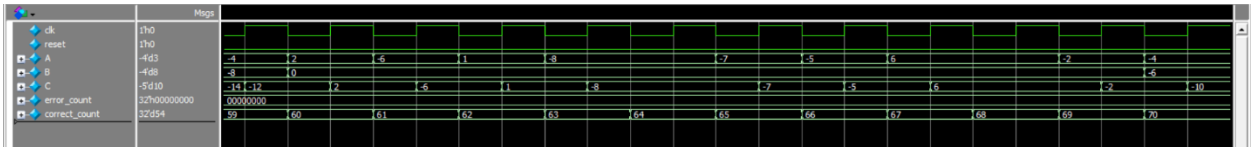
- Test_14



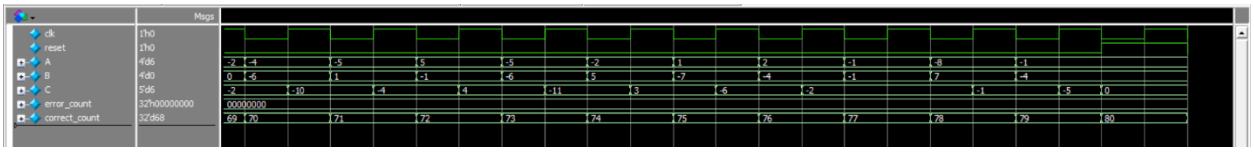
- Test_15



- Test_16



- Test_17 && Test_18

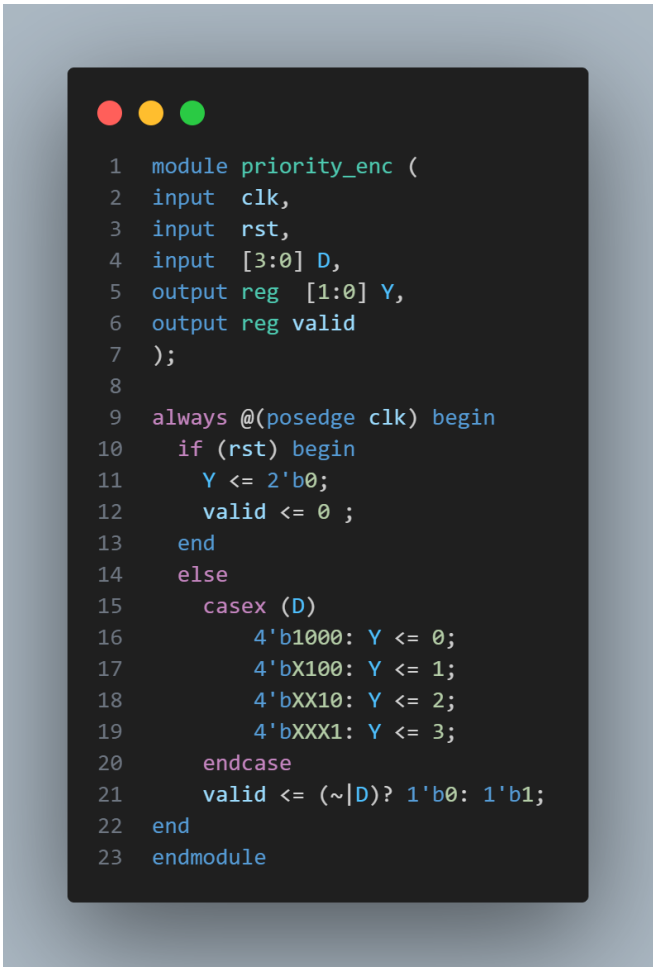


- Transcript

```
# test passes
# test passes
# test passes
# test passes
# correct counter =          81 , error counter =           0
# ** Note: $stop    : adder_tb.sv(104)
#    Time: 324 ns  Iteration: 1  Instance: /adder_tb
# Break in Module adder_tb at adder_tb.sv line 104
```

P2)

1. Design



```
module priority_enc (
input  clk,
input  rst,
input  [3:0] D,
output reg  [1:0] Y,
output reg valid
);

always @(posedge clk) begin
  if (rst) begin
    Y <= 2'b0;
    valid <= 0 ;
  end
  else
    casex (D)
        4'b1000: Y <= 0;
        4'bX100: Y <= 1;
        4'bXX10: Y <= 2;
        4'bXXX1: Y <= 3;
    endcase
    valid <= (~|D)? 1'b0: 1'b1;
end
endmodule
```

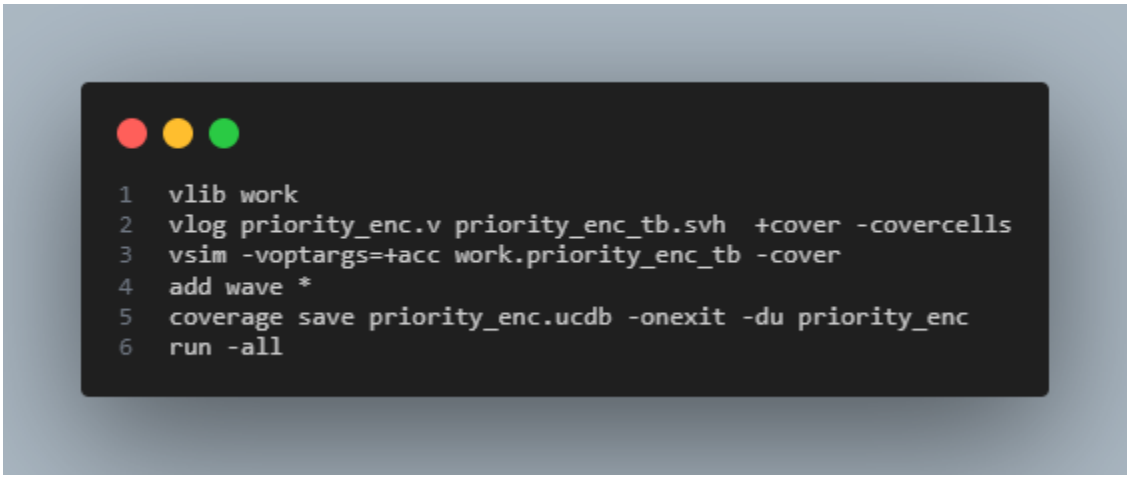
2. Verification plan

	A	B	C	D	E
1	**Label**	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
3	TEST_2	Assign D with all possible values (from 0 to 16)	Directed	-	A checker in the testbench to make sure the output is correct
4	TEST_3	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

3. Testbench

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

    integer error_count , correct_count ;
    integer D_count ;

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

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

    initial begin
        D = 4 ;
        rst = 0 ;
        error_count = 0 ;
        correct_count = 0 ;
        // TEST_1
        assert_reset () ;
        $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;
        // TEST_2
        for (D_count = 0 ; D_count < 17 ; D_count = D_count + 1 ) begin
            D = D_count ;
            check_result () ;
            $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;
        end
        // TEST_3
        assert_reset () ;
        $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;

        $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
        $stop ;

    end

    task check_reset ();
        @ (negedge clk) ;
        if (Y == 0 ) begin
           $display ("test reset passes") ;
           correct_count = correct_count + 1 ;
        end
        else begin
                $display ("test reset  fail ") ;
                error_count = error_count + 1 ;
            end
    endtask

    task check_result ();
        @ (negedge clk) ;
        if (D == 0 ) begin
            if ( valid == 0 ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else if (D[0] == 1) begin
            if (Y == 3 && valid == 1 ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else if (D[1] == 1) begin
            if (Y == 2 && valid == 1 ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else if (D[2] == 1) begin
            if (Y == 1 && valid == 1 ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else if (D[3] == 1) begin
            if (Y == 0 && valid == 1 ) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
    endtask

    task assert_reset ();
        rst = 1 ;
        // expected_out = 0 ;
        check_reset () ;
        rst = 0 ;
    endtask


endmodule
```

4. DO file



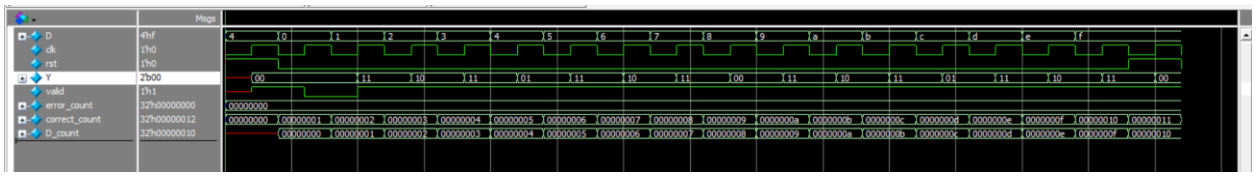
```
vlib work
vlog priority_enc.v priority_enc_tb.svh  +cover -covercells
vsim -voptargs=+acc work.priority_enc_tb -cover
add wave *
coverage save priority_enc.ucdb -onexit -du priority_enc
run -all
```

5. Coverage report

```
# --------------------------------------------------------------------------------
# === Instance: /\priority_enc_tb#DUT
# === Design Unit: work.priority_enc
# --------------------------------------------------------------------------------
# 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 priority_enc.v
# ------------------------------------IF Branch------------------------------------
#     10                                        19     Count coming in to IF
#     10              1                          2       if (rst) begin
#
#     14              1                         17       else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ----------------------------------CASE Branch----------------------------------
#     15                                        17     Count coming in to CASE
#     16              1                          1         4'b1000: Y <= 0;
#
#     17              1                          2         4'bX100: Y <= 1;
#
#     18              1                          4         4'bXX10: Y <= 2;
#
#     19              1                          8         4'bXXX1: Y <= 3;
#
#                                                2     All False Count
# Branch totals: 5 hits of 5 branches = 100.00%
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                       8         8         0   100.00%
#
```

```
#
# 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%
```

6. Questasim snippets



```
# test reset passes
# D = 0100 , Y = 00 , valid = 1
# test passes
# D = 0000 , Y = 00 , valid = 0
# test passes
# D = 0001 , Y = 11 , valid = 1
# test passes
# D = 0010 , Y = 10 , valid = 1
# test passes
# D = 0011 , Y = 11 , valid = 1
# test passes
# D = 0100 , Y = 01 , valid = 1
# test passes
# D = 0101 , Y = 11 , valid = 1
# test passes
# D = 0110 , Y = 10 , valid = 1
# test passes
# D = 0111 , Y = 11 , valid = 1
# test passes
# D = 1000 , Y = 00 , valid = 1
# test passes
# D = 1001 , Y = 11 , valid = 1
# test passes
# D = 1010 , Y = 10 , valid = 1
# test passes
# D = 1011 , Y = 11 , valid = 1
# test passes
# D = 1100 , Y = 01 , valid = 1
# test passes
# D = 1101 , Y = 11 , valid = 1
# test passes
# D = 1110 , Y = 10 , valid = 1
# test passes
# D = 1111 , Y = 11 , valid = 1
# test reset passes
# D = 1111 , Y = 00 , valid = 1
# correct counter =          18 , error counter =           0
```

P3)

1. Design

```
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

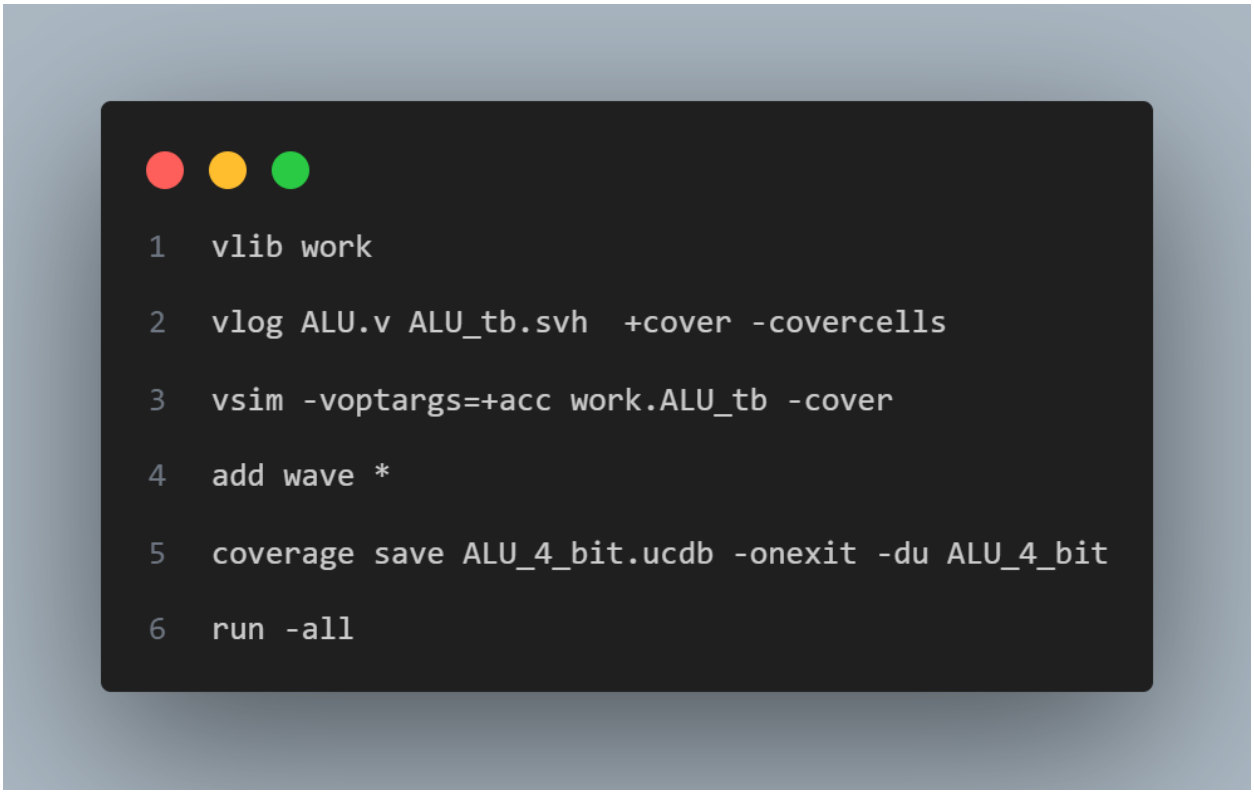
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
TEST_2	Assign A with MAXPOS value (7) & B with zero & opcode with 0 , the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_3	Assign A with MAXPOS value (7) & B with MAXPOS & opcode with 0 the output should be (14)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_4	Assign A with MAXPOS value (7) & B with MAXNEG & opcode with 0 the output should be (-1)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_5	Assign A with MAXNEG value (-8) & B with zero & opcode with 0 the output should be (-8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_6	Assign A with MAXNEG value (-8) & B with MAXPOS & opcode with 0 the output should be (-1)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_7	Assign A with MAXNEG value (-8) & B with MAXNEG & opcode with 0 the output should be (-16)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_8	Assign A with zero & B with zero & opcode with 0 the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_9	Assign A with zero & B with MAXPOS & opcode with 0 the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_10	Assign A with zero & B with MAXNEG & opcode with 0 the output should be (-8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_11	Assign A with MAXPOS value (7) & B with zero & opcode with 1 , the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_12	Assign A with MAXPOS value (7) & B with MAXPOS & opcode with 1 the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_13	Assign A with MAXPOS value (7) & B with MAXNEG & opcode with 1 the output should be (15)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_14	Assign A with MAXNEG value (-8) & B with zero & opcode with 1 the output should be (-8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_15	Assign A with MAXNEG value (-8) & B with MAXPOS & opcode with 1 the output should be (-15)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_16	Assign A with MAXNEG value (-8) & B with MAXNEG & opcode with 1 the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_17	Assign A with zero & B with zero & opcode with 1 the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_18	Assign A with zero & B with MAXPOS & opcode with 1 the output should be (-7)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_19	Assign A with zero & B with MAXNEG & opcode with 1 the output should be (8)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_20	Assign A with 10 random values & opcode with 2 the output should be the bit wise inverter of A	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_21	Assign B with 10 random values & opcode with 3 the output should be the reduction OR of B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_22	Assign A with random value & Assign B with random value & opcode with 0 the output should be the sum of A and B	Randomized	-	A checker in the testbench to make sure the output is correct
TEST_23	Assign opcode with 1 the output should be the difference between A and B	Directed	-	A checker in the testbench to make sure the output is correct
TEST_24	Assign opcode with 2 the output should be the bit wise inverter of A	Directed	-	A checker in the testbench to make sure the output is correct
TEST_25	Assign opcode with 3 the output should be the reduction OR of B	Directed	-	A checker in the testbench to make sure the output is correct
TEST_26	Assign opcode with (2'bxx) to test the default case the output should be (0)	Directed	-	A checker in the testbench to make sure the output is correct
TEST_27	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

3. Testbench

```
module ALU_tb ();
    reg signed [3:0] A ,B ;
    reg clk ;
    reg rst ;
    reg [1:0] opcode ;
    wire [4:0] C ;

    localparam MAXPOS = 7 ;
    localparam MAXNEG = -8 ;

    integer error_count , correct_count ;

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

    ALU_4_bit DUT (clk,rst,opcode,A,B,C) ;

    initial begin
        rst = 0 ;
        A = 0 ;
        B = 0 ;
        correct_count = 0 ;
        error_count = 0 ;
        opcode = 0 ;
        // TEST_1
        assert_reset () ;
        // TEST_2
        A = MAXPOS ;
        B = 0 ;
        check_result (7) ;
        // TEST_3
        B = MAXPOS ;
        check_result (14) ;
        // TEST_4
        B = MAXNEG ;
        check_result (-1) ;
        // TEST_5
        A = MAXNEG;
        B = 0 ;
        check_result (-8) ;
        // TEST_6
        B = MAXPOS ;
        check_result (-1) ;
        // TEST_7
        B = MAXNEG ;
        check_result (-16) ;
        // TEST_8
        A = 0 ;
        B = 0 ;
        check_result (0) ;
        // TEST_9
        B = MAXPOS ;
        check_result (7) ;
        // TEST_10
        B = MAXNEG ;
        check_result (-8) ;
        // TEST_11
        opcode = 1 ;
        A = MAXPOS ;
        B = 0 ;
        check_result (7) ;
        // TEST_12
        B = MAXPOS ;
        check_result (0) ;
        // TEST_13
        B = MAXNEG ;
        check_result (15) ;
        // TEST_14
        A = MAXNEG;
        B = 0 ;
        check_result (-8) ;
        // TEST_15
        B = MAXPOS ;
        check_result (-15) ;
        // TEST_16
        B = MAXNEG ;
        check_result (0) ;
        // TEST_17
        A = 0 ;
        B = 0 ;
        check_result (0) ;
        // TEST_18
        B = MAXPOS ;
        check_result (-7) ;
        // TEST_19
        B = MAXNEG ;
        check_result (8) ;
        // TEST_20
        opcode = 2 ;
        repeat (10)  begin
            A = $random ;
            check_result (~A) ;
        end
        // TEST_21
        opcode = 3 ;
        repeat (10)  begin
            B = $random ;
            check_result (|B) ;
        end
        // TEST_22
        A = $random ;
        B = $random ;
        opcode = 0 ;
        check_result (A+B) ;
        // TEST_23
        opcode = 1 ;
        check_result (A-B) ;
        // TEST_24
        opcode = 2 ;
        check_result (~A) ;
        // TEST_25
        opcode = 3 ;
        check_result (|B) ;
        // TEST_26
        opcode = 2'bxx ;
        check_result (0);
        // TEST_27
        assert_reset();

        $display ("correct counter = %d , error counter = %d " ,
correct_count , error_count ) ;
        $stop ;
    end
```

```
task check_result (input signed [4:0] expected_out);
        @ (negedge clk) ;
        if (opcode == 0 || opcode == 1) begin
            if (C == expected_out) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else if (opcode == 2) begin
            if (C == expected_out) begin
            $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
        else begin
            if (C == expected_out) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
            else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        end
    endtask

    task assert_reset ();
        rst = 1 ;
        // expected_out = 0 ;
        check_result (0) ;
        rst = 0 ;
    endtask

endmodule
```

4. DO file



```
1  vlib work
2  vlog ALU.v ALU_tb.svh  +cover -covercells
3  vsim -voptargs=+acc work.ALU_tb -cover
4  add wave *
5  coverage save ALU_4_bit.ucdb -onexit -du ALU_4_bit
6  run -all
```

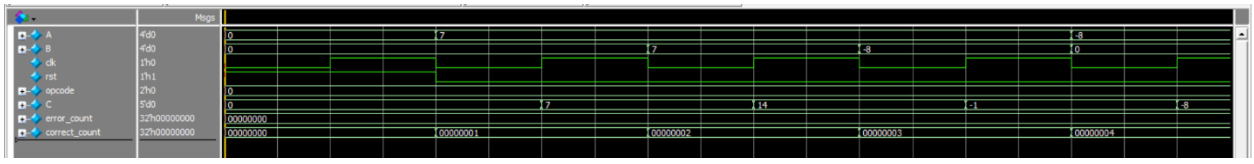
5. Coverage report

```
# ================================================================================
# === Instance: /\ALU_tb#DUT
# === Design Unit: work.ALU_4_bit
# ================================================================================
# 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------------------------------------
#     21                                       42     Count coming in to CASE
#     22              1                        11             Add:            Alu_out = A + B;
#
#     23              1                        10             Sub:            Alu_out = A - B;
#
#     24              1                        10             Not_A:          Alu_out = ~A;
#
#     25              1                        10             ReductionOR_B:  Alu_out = |B;
#
#     26              1                         1               default:  Alu_out = 5'b0;
#
# Branch totals: 5 hits of 5 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     32                                       38     Count coming in to IF
#     32              1                         4           if (reset)
#
#     34              1                        34           else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                       9         9         0   100.00%
#
```

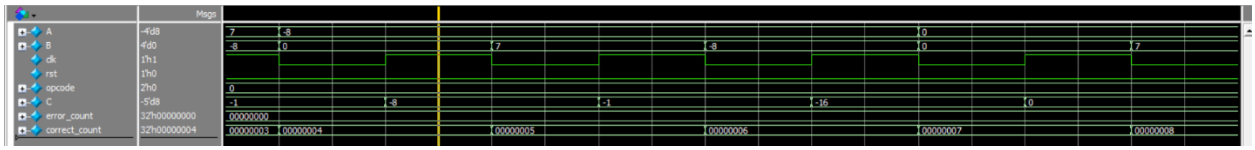
```
#
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                         44        44         0   100.00%
#
# ================================Toggle Details================================
#
# 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)
#
```

6. Questasim snippets

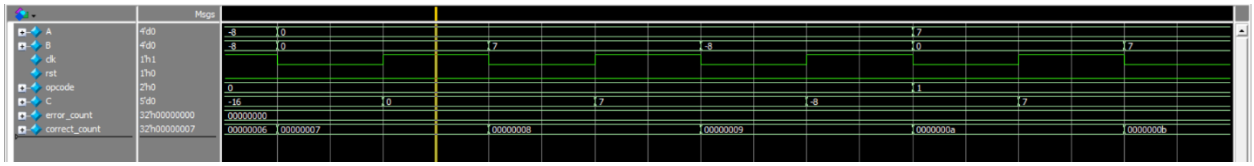
- Test 1 && Test_2 && Test_3 && Test_4



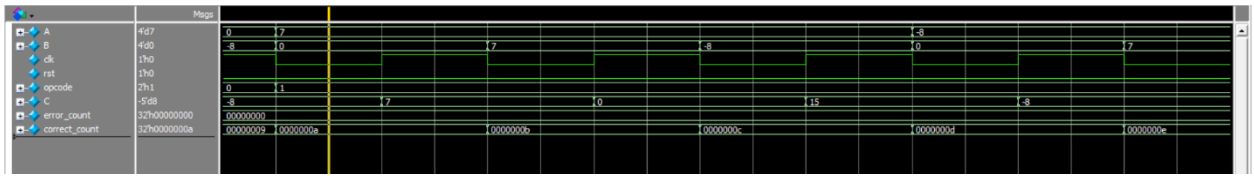
- Test_5 && Test_6 && Test_7



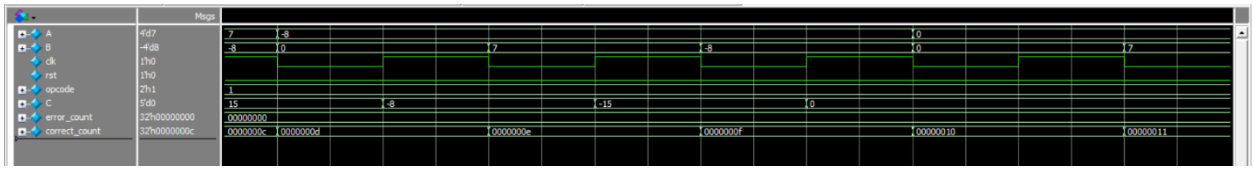
- Test_8 && Test_9 && Test_10



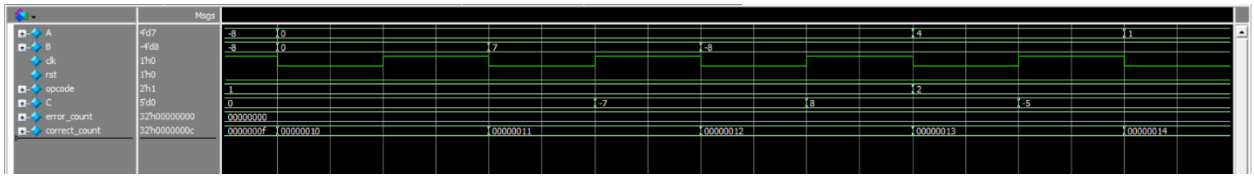
- Test_11 && Test_12 && Test_13



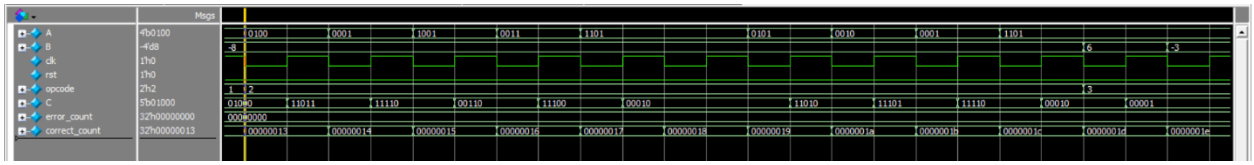
- Test_14 && Test_15 && Test_16



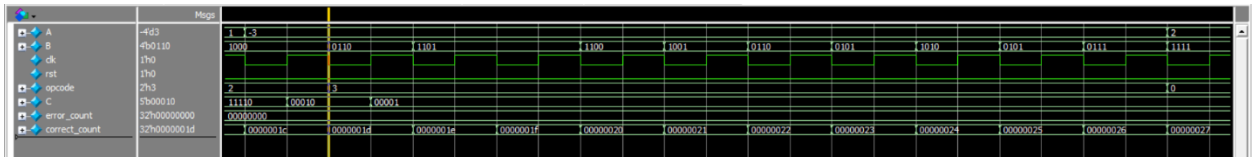
- Test_17 && Test_18 && Test_19



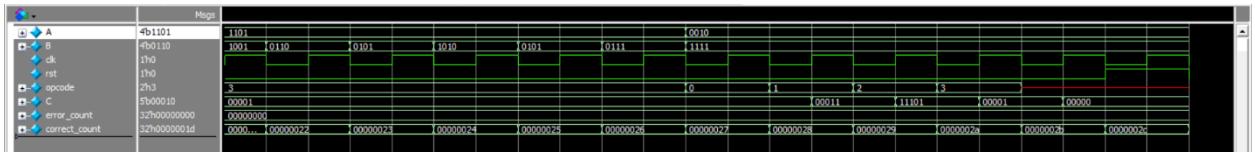
- TEST_20



- TEST_21



- Test_22 && Test_23 && Test_24 && Test_25 && TEST_26 && TEST_27



- Transcript

```
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# test passes
# correct counter =          45 , error counter =           0
# ** Note: $stop    : ALU_tb.svh(125)
#    Time: 180 ns  Iteration: 1  Instance: /ALU_tb
# Break in Module ALU_tb at ALU_tb.svh line 125
```

P4)

1. 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;
reg  [47:0] C_reg;
reg  [36:0] mult_out;
reg  [18:0] adder_out_stg1 ,adder_out_stg2 ;

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;
         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;
        adder_out_stg2 <= adder_out_stg1;
        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_stg2;
    end
end

endmodule
```

- Change **adder_out_stg1 & adder_out_stg2** from **18 bit width** to **19 bit width** to cover the overflow cases
- Change **mul_out** from **48 bit width** to **37 bit width** only as it take the value of multplication of two 18 bits width variables

2. Verification plan

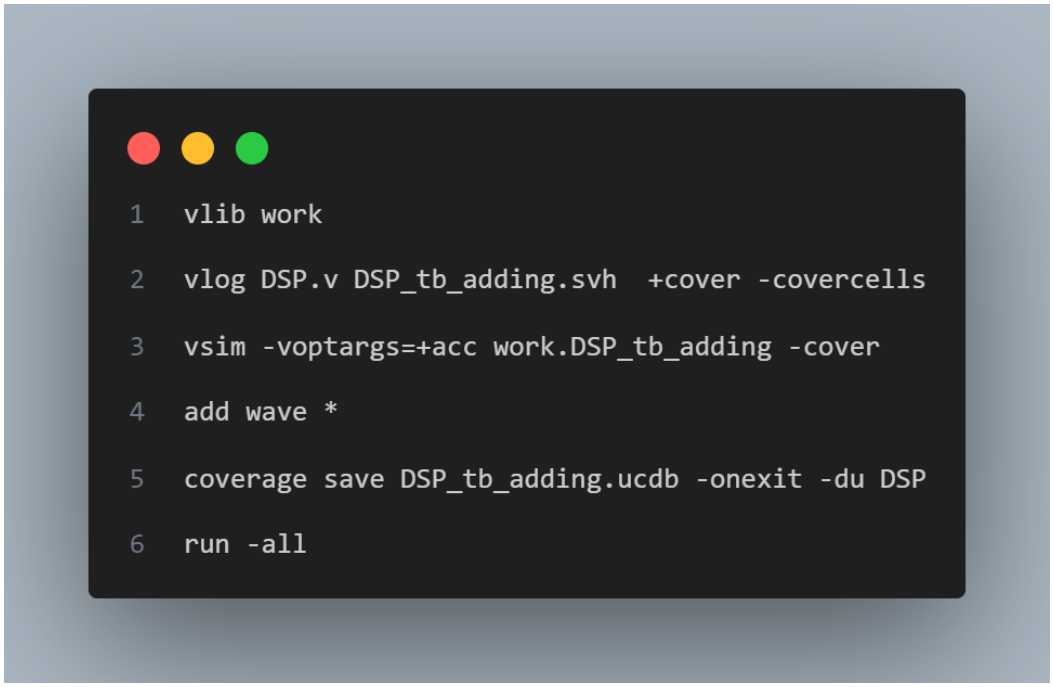
	A	B	C	D	E
1	**Label**	**Design Requirement Description**	**Stimulus Generation**	**Functional Coverage**	**Functionality Check**
2	TEST_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
3	TEST_2	Assign A,B,C&D with 1000 random values and check the output of design with the output of the golden module	Randomized	-	A checker in the testbench to make sure the output is correct
4	TEST_3	Assign A,B,C&D with extrem values and check the output of design with the output of the golden module	Directed	-	A checker in the testbench to make sure the output is correct
5	TEST_4	When the reset is asserted, the output value should be low	Directed at the end of the simulation	-	A checker in the testbench to make sure the output is correct

3. Testbench

```
module DSP_tb_adding();
    reg [17:0] A ;
    reg [17:0] B ;
    reg [47:0] C ;
    reg [17:0] D ;
    reg [47:0] P_expected;
    reg clk ;
    reg rst_n ;
    wire [47:0] P;

    integer error_count , correct_count ;

    DSP #(.OPERATION("ADD")) DUT (A, B, C, D, clk, rst_n, P);

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

    initial begin
        rst_n = 1 ;
        A = 0 ;
        B = 0 ;
        C = 0 ;
        D = 0 ;
        error_count = 0 ;
        correct_count = 0 ;
        assert_reset () ;
        repeat (1000) begin
            A = $random ;
            B = $random ;
            C = $random ;
            D = $random ;
            golden_model () ;
            check_result () ;
        end

        assert_reset () ;

        $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
        $stop ;
    end
```

```
task check_result ();
    repeat (5) @(negedge clk) ;
    // golden_model () ;
        if (P == P_expected) begin
                $display ("test passes") ;
                correct_count = correct_count + 1 ;
            end
        else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
    endtask

    task assert_reset ();
        rst_n = 0 ;
        @(negedge clk) ;
        if (P == 0 ) begin
            $display ("test passes") ;
            correct_count = correct_count + 1 ;
        end
        else begin
                $display ("test fail ") ;
                error_count = error_count + 1 ;
            end
        rst_n = 1 ;
    endtask

    task golden_model ();
        P_expected = ((D+B)*A)+C ;
    endtask


endmodule
```

4. DO file



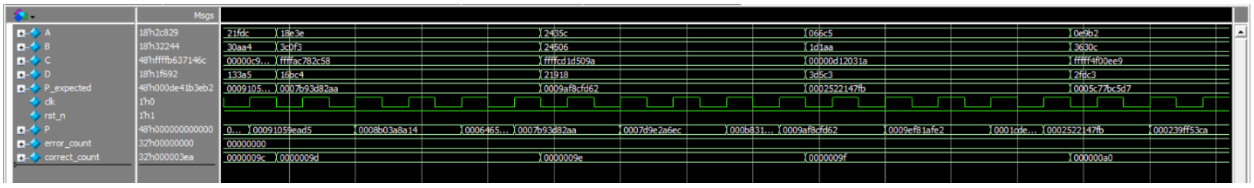
```
vlib work
vlog DSP.v DSP_tb_adding.svh  +cover -covercells
vsim -voptargs=+acc work.DSP_tb_adding -cover
add wave *
coverage save DSP_tb_adding.ucdb -onexit -du DSP
run -all
```

5. Coverage report

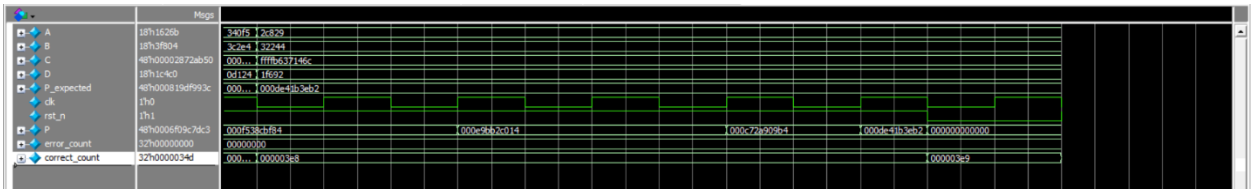
```
# ================================================================================
# === Instance: /\DSP_tb_adding#DUT
# === Design Unit: work.DSP
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                         2         2         0   100.00%
#
# ================================Branch Details================================
#
# Branch Coverage for instance /\DSP_tb_adding#DUT
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File DSP.v
# ------------------------------------IF Branch------------------------------------
#     14                                     5004     Count coming in to IF
#     14              1                         4      if (!rst_n) begin
#
#     24              1                      5000      else begin
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                      17        17         0   100.00%
#
```

```
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                        694       694         0   100.00%
#
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\DSP_tb_adding#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[18-0]        1           1      100.00
#                                adder_out_stg2[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     =        347
# Toggled Node Count   =        347
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (694 of 694 bins)
#
```

6. Questasim snippets
   - TEST_1 && TEST_2



   - TEST_3 && TEST_4



   - Transcript

```
# test passes
# test passes
# test passes
# correct counter =        1002 , error counter =           0
# ** Note: $stop    : DSP_tb_adding.svh(43)
#    Time: 20008 ns  Iteration: 1  Instance: /DSP_tb_adding
# Break in Module DSP_tb_adding at DSP_tb_adding.svh line 43
```