

P1)

1. Design

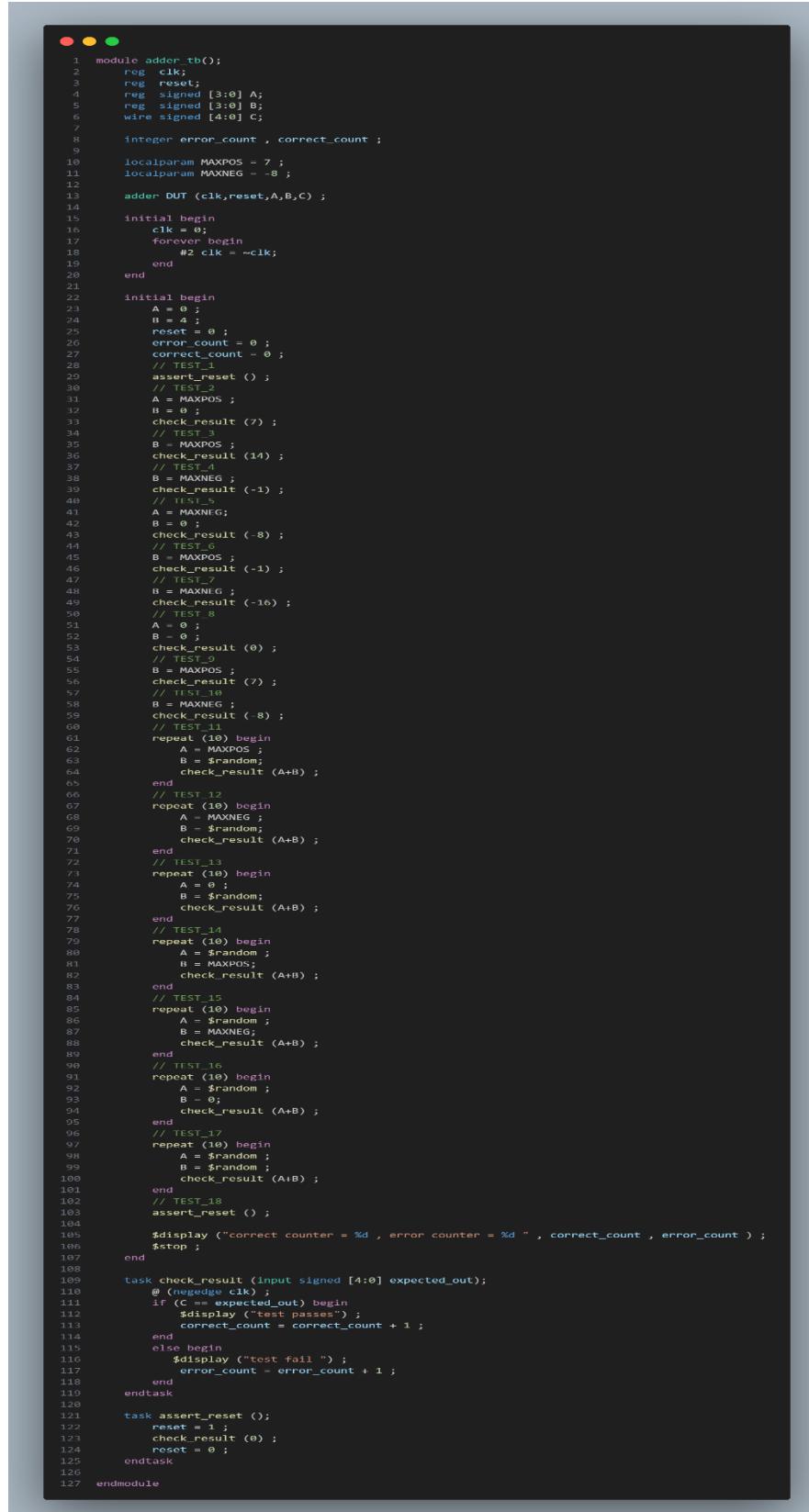


```
1  module adder (
2      input  clk,
3      input  reset,
4      input  signed [3:0] A, // Input data A in 2's complement
5      input  signed [3:0] B, // Input data B in 2's complement
6      output reg signed [4:0] C // Adder output in 2's complement
7  );
8
9  // Register output C
10 always @(posedge 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

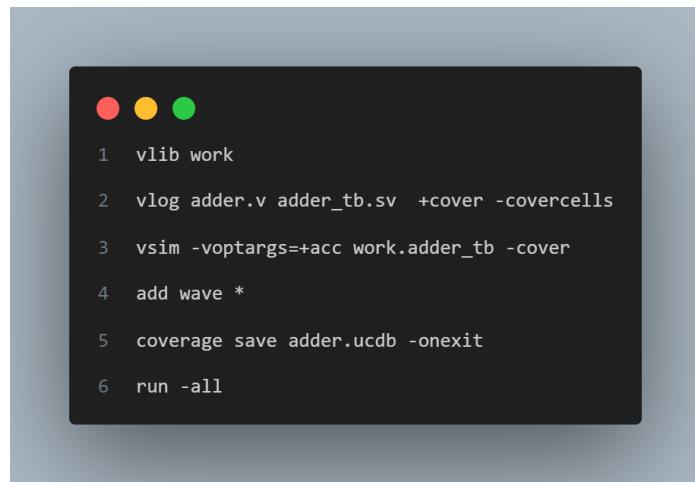
A	B	C	D	E
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
1	TEST_1 When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
2	TEST_2 Assign A with MAXPOS value (7) & B with zero , the output should be (7)	Directed	-	A checker in the testbench to make sure the output is correct
3	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
4	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
5	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
6	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
7	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
8	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
9	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
10	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
11	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
12	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
13	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
14				
15	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
16	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
17	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
18	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
19	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



```
1  module adder_tb();
2    reg clk;
3    reg reset;
4    reg signed [3:0] A;
5    reg signed [3:0] B;
6    wire signed [4:0] C;
7
8    integer error_count , correct_count ;
9
10   localparam MAXPOS = 7 ;
11   localparam MAXNEG = -8 ;
12
13   adder DUT (clk,reset,A,B,C) ;
14
15   initial begin
16     clk = 0;
17     forever begin
18       #2 clk = ~clk;
19     end
20   end
21
22   initial begin
23     A = 0 ;
24     B = 4 ;
25     reset = 0 ;
26     error_count = 0 ;
27     correct_count = 0 ;
28     // TEST_1
29     assert_reset () ;
30     // TEST_2
31     A = MAXPOS ;
32     B = 0 ;
33     check_result (7) ;
34     // TEST_3
35     B = MAXPOS ;
36     check_result (14) ;
37     // TEST_4
38     B = MAXNEG ;
39     check_result (-1) ;
40     // TEST_5
41     A = MAXNEG;
42     B = 0 ;
43     check_result (-8) ;
44     // TEST_6
45     B = MAXPOS ;
46     check_result (-1) ;
47     // TEST_7
48     B = MAXNEG ;
49     check_result (-16) ;
50     // TEST_8
51     A = 0 ;
52     B = 0 ;
53     check_result (0) ;
54     // TEST_9
55     B = MAXPOS ;
56     check_result (7) ;
57     // TEST_10
58     B = MAXNEG ;
59     check_result (-8) ;
60     // TEST_11
61     repeat (10) begin
62       A = MAXPOS ;
63       B = $random;
64       check_result (A+B) ;
65     end
66     // TEST_12
67     repeat (10) begin
68       A = MAXNEG ;
69       B = $random;
70       check_result (A+B) ;
71     end
72     // TEST_13
73     repeat (10) begin
74       A = 0 ;
75       B = $random;
76       check_result (A+B) ;
77     end
78     // TEST_14
79     repeat (10) begin
80       A = $random ;
81       B = MAXPOS;
82       check_result (A+B) ;
83     end
84     // TEST_15
85     repeat (10) begin
86       A = $random ;
87       B = MAXNEG;
88       check_result (A+B) ;
89     end
90     // TEST_16
91     repeat (10) begin
92       A = $random ;
93       B = 0 ;
94       check_result (A+B) ;
95     end
96     // TEST_17
97     repeat (10) begin
98       A = $random ;
99       B = $random ;
100      check_result (A+B) ;
101    end
102    // TEST_18
103    assert_reset () ;
104
105    $display ("correct counter = %d , error counter = %d ", correct_count , error_count ) ;
106    $stop ;
107  end
108
109  task check_result (input signed [4:0] expected_out);
110    @ (negedge clk) ;
111    if (C == expected_out) begin
112      $display ("test passes");
113      correct_count = correct_count + 1 ;
114    end
115    else begin
116      $display ("test fail ");
117      error_count = error_count + 1 ;
118    end
119  endtask
120
121  task assert_reset ();
122    reset = 1 ;
123    check_result (0) ;
124    reset = 0 ;
125  endtask
126
127 endmodule
```

4. DO file



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

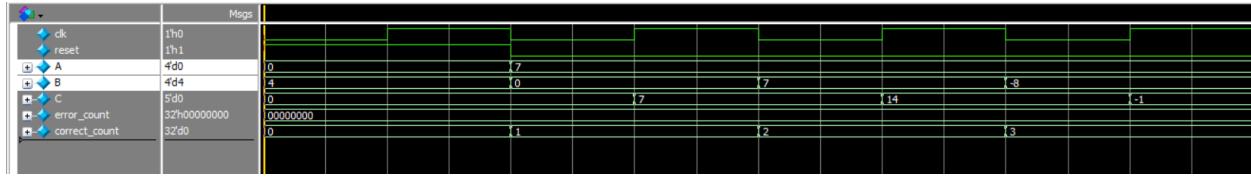
```

#
# 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)
#
#

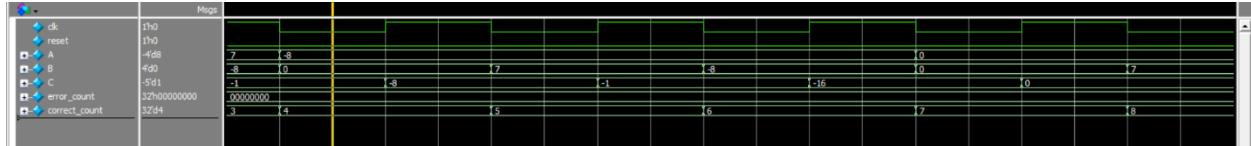
```

6. Questasim snippets

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



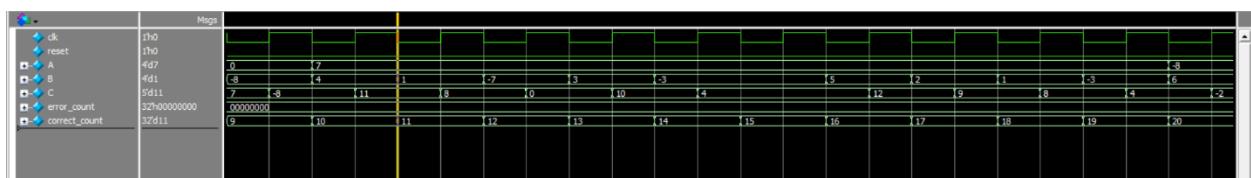
- Test_5 && Test_6 && Test_7



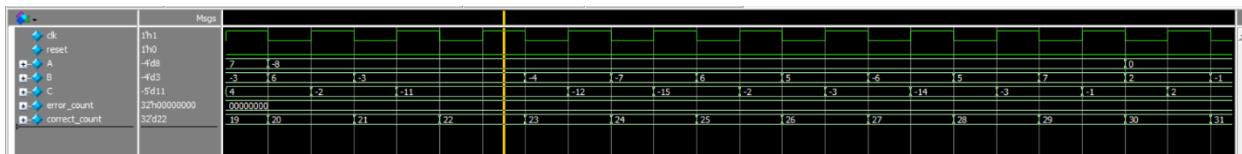
- Test_8 && Test_9 && Test_10



- Test_11



- Test_12



- Test_13



- Test_14



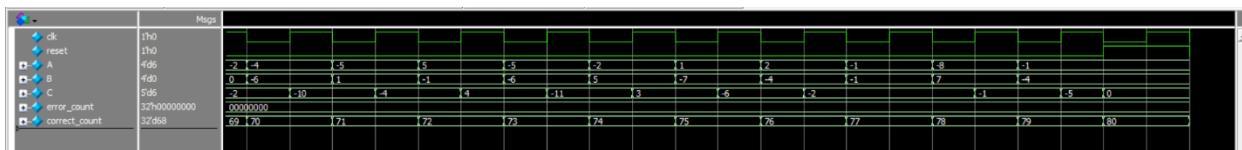
- Test_15



- Test_16



- Test_17 && Test_18

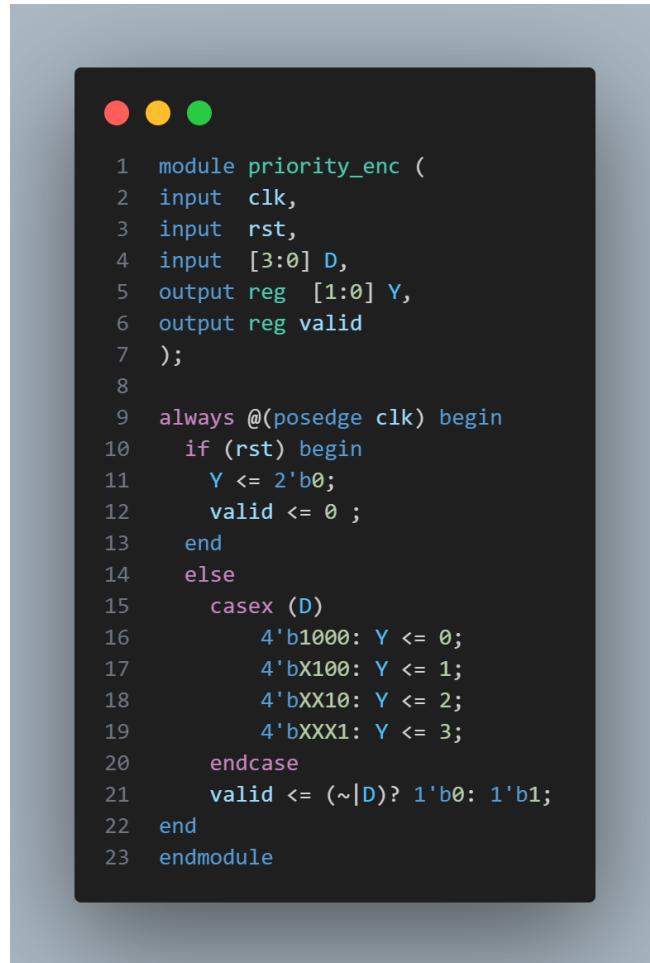


- Transcript

```
# 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



```
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
15       casex (D)
16         4'b1000: Y <= 0;
17         4'bX100: Y <= 1;
18         4'bXX10: Y <= 2;
19         4'bXXX1: Y <= 3;
20       endcase
21     valid <= (~|D)? 1'b0: 1'b1;
22   end
23 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

```
● ● ●
1  module priority_enc_tb();
2    reg [3:0] D ;
3    reg clk ;
4    reg rst ;
5    wire [1:0] Y ;
6    wire valid ;
7
8    integer error_count , correct_count ;
9    integer D_count ;
10
11   priority_enc DUT (clk,rst,D,Y,valid);
12
13  initial begin
14    clk = 0;
15    forever begin
16      #2 clk = ~clk;
17    end
18  end
19
20  initial begin
21    D = 4 ;
22    rst = 0 ;
23    error_count = 0 ;
24    correct_count = 0 ;
25    // TEST_1
26    assert_reset () ;
27    $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;
28    // TEST_2
29    for (D_count = 0 ; D_count < 17 ; D_count = D_count + 1 ) begin
30      D = D_count ;
31      check_result () ;
32      $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;
33    end
34    // TEST_3
35    assert_reset () ;
36    $display ("D = %b , Y = %b , valid = %b" , D , Y , valid) ;
37
38    $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
39    $stop ;
40  end
41
42
43  task check_reset ();
44    @ (negedge clk) ;
45    if (Y == 0 ) begin
46      $display ("test reset passes") ;
47      correct_count = correct_count + 1 ;
48    end
49    else begin
50      $display ("test reset fail ") ;
51      error_count = error_count + 1 ;
52    end
53  endtask
54
55  task check_result ();
56    @ (negedge clk) ;
57    if (D == 0 )begin
58      if ( valid == 0 ) begin
59        $display ("test passes") ;
60        correct_count = correct_count + 1 ;
61      end
62      else begin
63        $display ("test fail ") ;
64        error_count = error_count + 1 ;
65      end
66    end
67    else if (D[0] == 1) begin
68      if (Y == 3 && valid == 1 ) begin
69        $display ("test passes") ;
70        correct_count = correct_count + 1 ;
71      end
72      else begin
73        $display ("test fail ") ;
74        error_count = error_count + 1 ;
75      end
76    end
77    else if (D[1] == 1) begin
78      if (Y == 2 && valid == 1 ) begin
79        $display ("test passes") ;
80        correct_count = correct_count + 1 ;
81      end
82      else begin
83        $display ("test fail ") ;
84        error_count = error_count + 1 ;
85      end
86    end
87    else if (D[2] == 1) begin
88      if (Y == 1 && valid == 1 ) begin
89        $display ("test passes") ;
90        correct_count = correct_count + 1 ;
91      end
92      else begin
93        $display ("test fail ") ;
94        error_count = error_count + 1 ;
95      end
96    end
97    else if (D[3] == 1) begin
98      if (Y == 0 && valid == 1 ) begin
99        $display ("test passes") ;
100       correct_count = correct_count + 1 ;
101     end
102     else begin
103       $display ("test fail ") ;
104       error_count = error_count + 1 ;
105     end
106   end
107 endtask
108
109 task assert_reset ();
110   rst = 1 ;
111   // expected_out = 0 ;
112   check_reset () ;
113   rst = 0 ;
114 endtask
115
116
117 endmodule
```

4. DO file

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

5. Coverage report

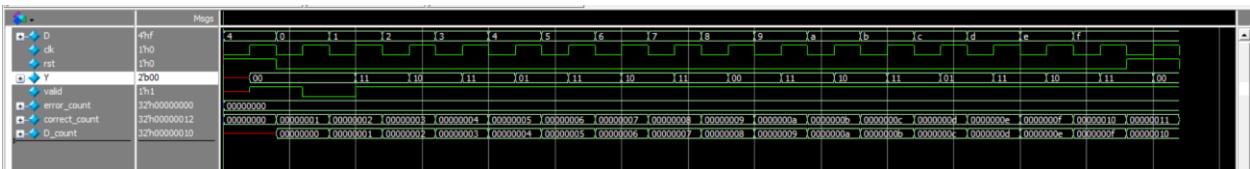
```
# -----
# === Instance: /\priority_enc_tb#DUT
# === Design Unit: work.priority_enc
# -----
# Branch Coverage:
#   Enabled Coverage      Bins    Hits    Misses  Coverage
#   -----      -----  -----  -----
#   Branches          7       7       0  100.00%
#
# =====Branch Details=====
#
# Branch Coverage 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

```
● ● ●

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    always @* begin
21        case (Opcode)
22            Add:          Alu_out = A + B;
23            Sub:          Alu_out = A - B;
24            Not_A:        Alu_out = ~A;
25            ReductionOR_B: Alu_out = |B;
26            default:      Alu_out = 5'b0;
27        endcase
28    end // always @ *
29
30    // Register output C
31    always @(posedge clk or posedge reset) begin
32        if (reset)
33            C <= 5'b0;
34        else
35            C<= Alu_out;
36    end
37
38 endmodule
39
```

2. Verification plan

A	B	C	D	E	
	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
1	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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
10	A	B	C	D	E
11	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
12	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
13	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
14	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
15	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
16	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
17	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
18	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
19	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
20	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
21	A	B	C	D	E
22	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
23	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
24	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
25	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
26	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
27	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
28	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
28	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 (?) ;
    // 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 (?) ;
    // TEST_10
    B = MAXNEG ;
    check_result (-8) ;
    // TEST_11
    opcode = 1 ;
    A = MAXPOS ;
    B = 0 ;
    check_result (?) ;
    // 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           11  Add: Alu_out = A + B;
#   23           10  Sub: Alu_out = A - B;
#   24           10  Not_A: Alu_out = ~A;
#   25           10  ReductionOR_B: Alu_out = |B;
#   26           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           4   if (reset)
#   34           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%
```

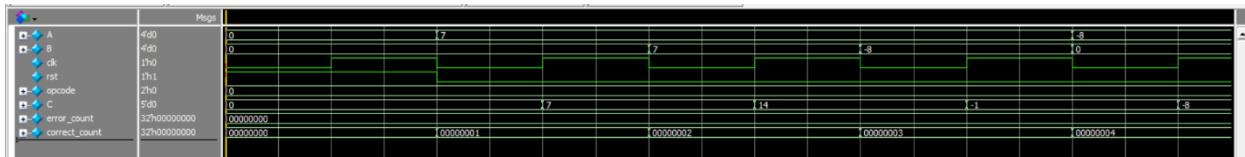
```

#
# 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)
#

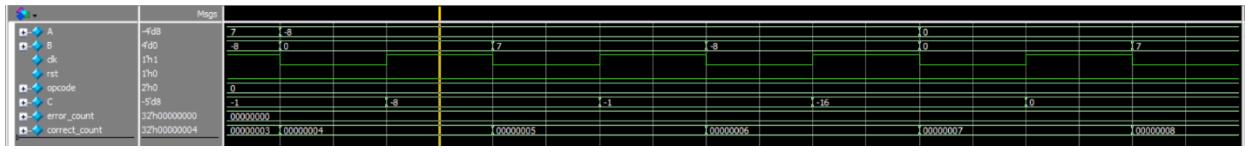
```

6. Questasim snippets

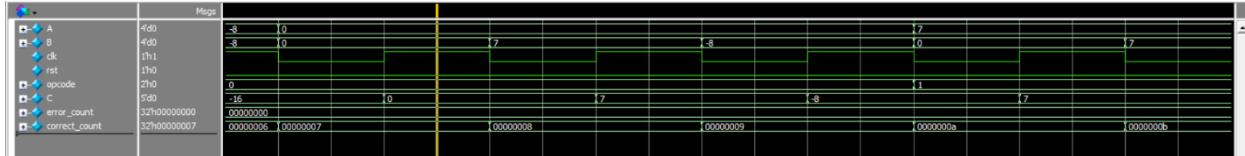
- Test 1 && Test_2 && Test_3 && Test_4



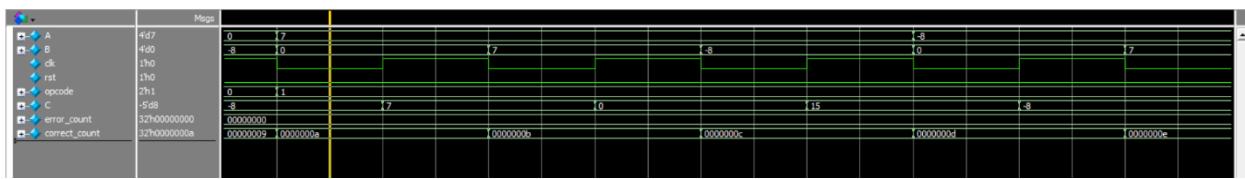
- Test_5 && Test_6 && Test_7



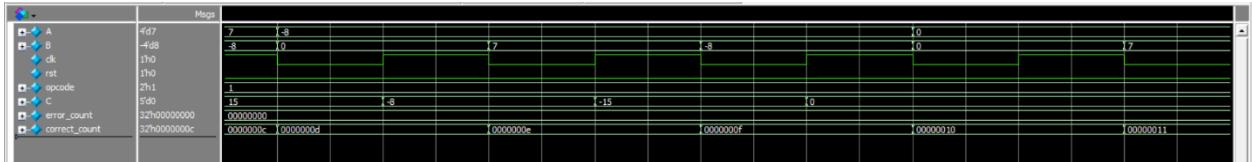
- Test_8 && Test_9 && Test_10



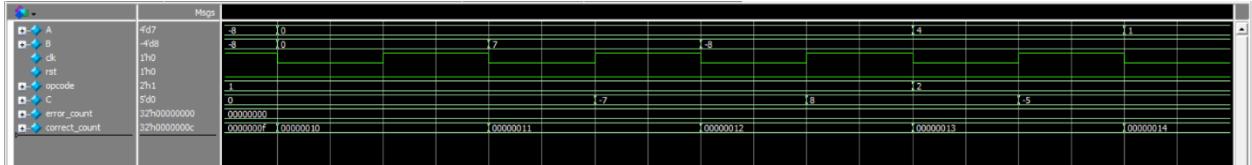
- Test_11 && Test_12 && Test_13



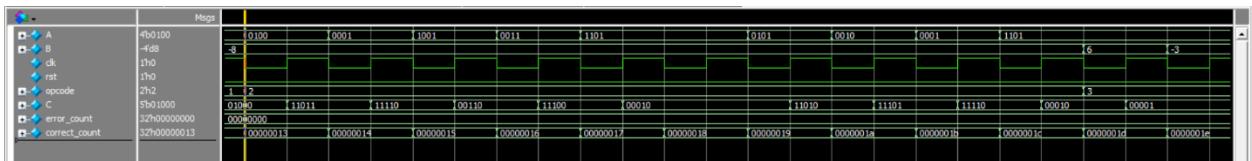
- Test_14 && Test_15 && Test_16



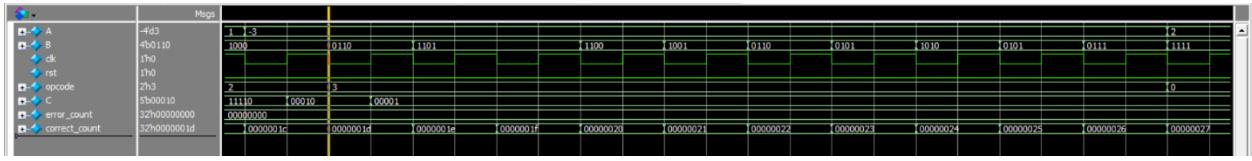
- Test_17 && Test_18 && Test_19



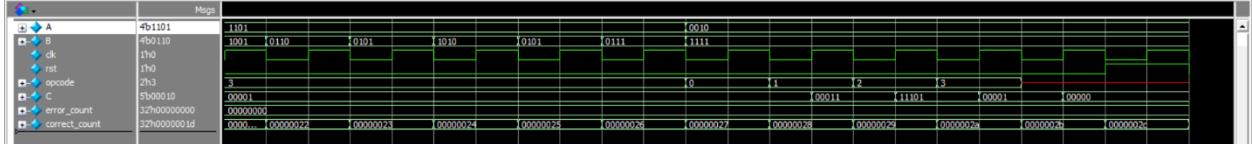
- TEST_20



- TEST_21



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



- Transcript

```
# test passes
# 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

```
● ● ●

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 [47:0] C_reg;
10 reg [36:0] mult_out;
11 reg [18:0] adder_out_stg1 ,adder_out_stg2 ;
12
13 always @(posedge clk or negedge rst_n) begin
14     if (!rst_n) begin
15         // reset
16         A_reg_stg1 <= 0;
17         A_reg_stg2 <= 0;
18         B_reg <= 0;
19         D_reg <= 0;
20         adder_out_stg1 <= 0;
21         mult_out <= 0;
22         P <= 0;
23     end
24     else begin
25         A_reg_stg1 <= A;
26         A_reg_stg2 <= A_reg_stg1;
27         B_reg <= B;
28         C_reg <= C;
29         D_reg <= D;
30         adder_out_stg2 <= adder_out_stg1;
31         if (OPERATION == "ADD") begin
32             adder_out_stg1 <= D_reg + B_reg;
33             P <= mult_out + C_reg;
34         end
35         else if (OPERATION == "SUBTRACT") begin
36             adder_out_stg1 <= D_reg - B_reg;
37             P <= mult_out - C_reg;
38         end
39         mult_out <= A_reg_stg2 * adder_out_stg2;
40     end
41 end
42
43 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 multiplication of two 18 bits width variables

2. Verification plan

	A	B	C	D	E
	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
1	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
2	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
3	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
4	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
5					

3. Testbench

```

1  module DSP_tb_adding();
2    reg [17:0] A ;
3    reg [17:0] B ;
4    reg [47:0] C ;
5    reg [17:0] D ;
6    reg [47:0] P_expected;
7    reg clk ;
8    reg rst_n ;
9    wire [47:0] P;
10
11   integer error_count , correct_count ;
12
13   DSP #( .OPERATION("ADD") ) DUT (A, B, C, D, clk, rst_n, P);
14
15   initial begin
16     clk = 0;
17     forever begin
18       #2 clk = ~clk;
19     end
20   end
21
22   initial begin
23     rst_n = 1 ;
24     A = 0 ;
25     B = 0 ;
26     C = 0 ;
27     D = 0 ;
28     error_count = 0 ;
29     correct_count = 0 ;
30     assert_reset () ;
31     repeat (1000) begin
32       A = $random ;
33       B = $random ;
34       C = $random ;
35       D = $random ;
36       golden_model () ;
37       check_result () ;
38     end
39
40     assert_reset () ;
41
42     $display ("correct counter = %d , error counter = %d " , correct_count , error_count ) ;
43     $stop ;
44   end

```



```

1  task check_result ();
2    repeat (5) @ (negedge clk) ;
3    // golden_model () ;
4    if (P == P_expected) begin
5      $display ("test passes") ;
6      correct_count = correct_count + 1 ;
7    end
8    else begin
9      $display ("test fail ") ;
10     error_count = error_count + 1 ;
11   end
12 endtask
13
14 task assert_reset ();
15   rst_n = 0 ;
16   @(negedge clk) ;
17   if (P == 0 ) begin
18     $display ("test passes") ;
19     correct_count = correct_count + 1 ;
20   end
21   else begin
22     $display ("test fail ") ;
23     error_count = error_count + 1 ;
24   end
25   rst_n = 1 ;
26 endtask
27
28 task golden_model ();
29   P_expected = ((D+B)*A)+C ;
30 endtask
31
32
33 endmodule

```

4. DO file

```
● ● ●

1 vlib work

2 vlog DSP.v DSP_tb_adding.svh +cover -covercells

3 vsim -voptargs=+acc work.DSP_tb_adding -cover

4 add wave *

5 coverage save DSP_tb_adding.ucdb -onexit -du DSP

6 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%
```

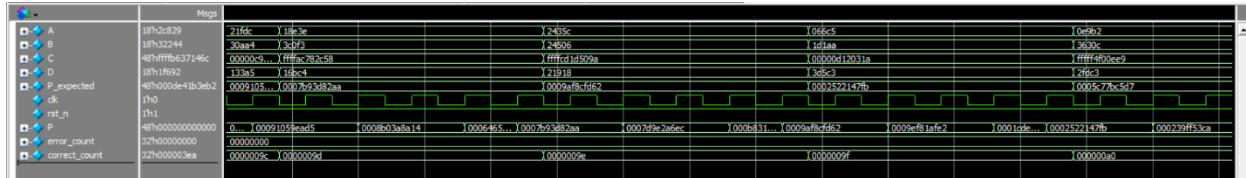
```

# 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)
#

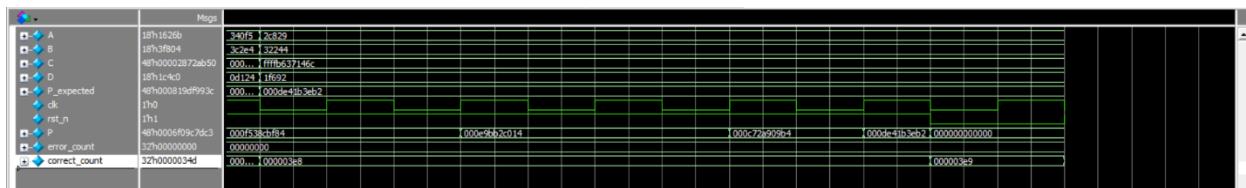
```

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

```