1.

```
// ADDER_11
repeat(50) begin
    rand_A = $random;
    rand_B = $random;
    drive_inputs(rand_A, rand_B);
    check_result(adder_fun(rand_A, rand_B));
end

// ADDER_12
assert_reset;
```

```
you got correct cases = 61, and wrong cases = 0

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

2.

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
ENCODER1	When reset is asserted, the output Y should be 0 and valid should be 0.	Directed at the start of the simulation	-	A checker in the testbench to verify that Y and valid are 0.
ENCODER2	When D has a valid input, Y should reflect the highest priority bit position, and valid should be 1.	Randomization	-	A checker in the testbench to ensure Y matches the highest priority bit and valid is 1.

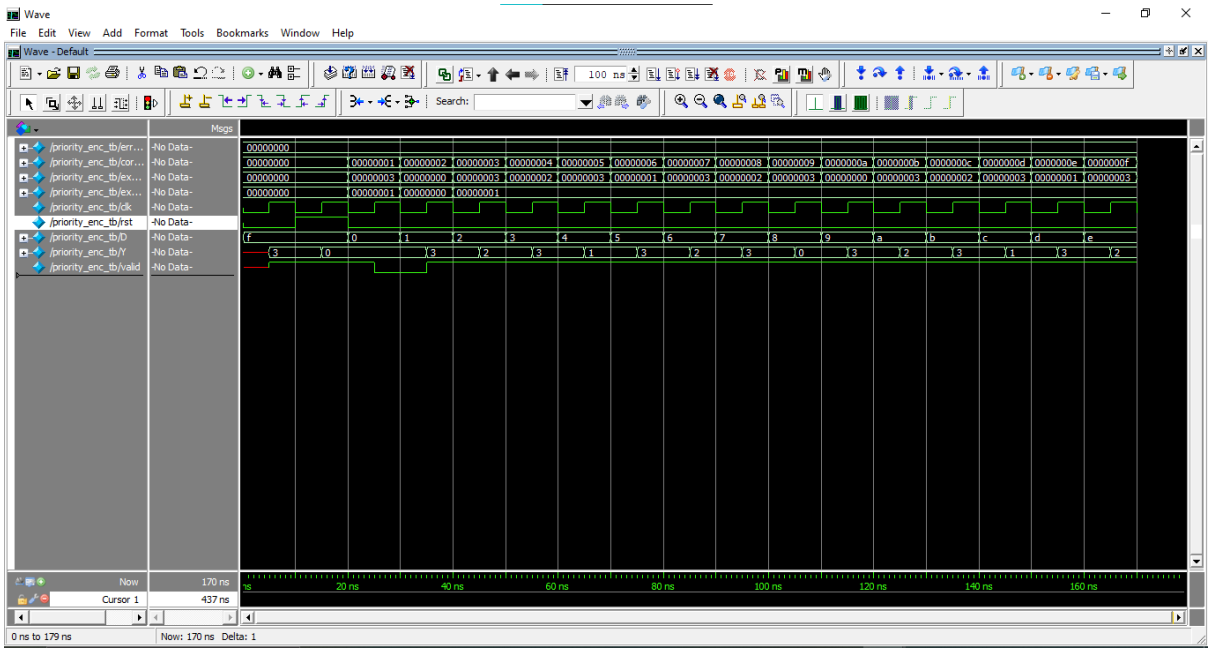
```
module priority_enc_tb;
    parameter CLK_PERIOD = 10;
    int error_count, correct_count;
    int expected_Y, expected_valid;

    // Signals
    logic clk, rst;
    logic [3:0] D;
    logic [1:0] Y;
    logic valid;

    // Instantiate the DUT
    priority_enc DUT (.*);

    // Clock Generation
    always #(CLK_PERIOD/2) clk = ~clk;

    // Task to apply inputs
    task drive_input (
        input [3:0] data
    );
        D = data;
        @(negedge clk);
    endtask

    // Expected priority encoding function
    function void expected_output(
        input  [3:0] din,
        output [1:0] expected_Y,
        output expected_valid
```

```
    );
        begin
            casex (din)
                4'b1000: begin
                    expected_Y = 2'b00;
                    expected_valid = 1'b1;
                end
                4'bx100: begin
                    expected_Y = 2'b01;
                    expected_valid = 1'b1;
                end
                4'bxx10: begin
                    expected_Y = 2'b10;
                    expected_valid = 1'b1;
                end
                4'bxxx1: begin
                    expected_Y = 2'b11;
                    expected_valid = 1'b1;
                end
                default: begin
                    expected_Y = 2'bxx;
                    expected_valid = 1'b0;
                end
            endcase
        end
    endfunction

    task check_result (
        input [1:0] expected_Y,
        input expected_valid
    );
        if (Y !== expected_Y || valid !== expected_valid) begin
            $display("[%0t] ERROR: Expected Y = %b, valid = %b | Got Y = %b,
valid = %b | D = %b",
                    $time, expected_Y, expected_valid, Y, valid, D);
            error_count++;
        end else begin
            correct_count++;
        end
    endtask


    task assert_reset;
        rst = 1'b1;         // Assert reset
        @(negedge clk);
        rst = 1'b0;
    endtask

    task wait_cycles (
```

```
        input integer num_cycles
    );
        repeat(num_cycles) @(negedge clk);
    endtask

    // Test process
    initial begin
        initialize_inputs;
        wait_cycles(1);

        // ENCODER1
        assert_reset;
        expected_output(D, expected_Y, expected_valid);
        check_result(2'b00, expected_valid);

        // ENCODER2
        for (int i = 0; i < 15; i++) begin
            drive_input(i);
            expected_output(i, expected_Y, expected_valid);
            check_result(expected_Y, expected_valid);
        end

        // Finish Simulation
        $display("you got correct cases = %0d, and wrong cases = %0d",
correct_count, error_count);
        $stop;
    end

    task initialize_inputs;
        clk = 0;
        rst = 0;
        D = 4'b1111;
    endtask

endmodule
```

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

```
> do run.do
you got correct cases = 16, and wrong cases = 0
```




```
> vcover report priority_enc_tb.ucdb -details -annotate -all -output
coverage

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

================================================================================
Statement Coverage:
    Enabled Coverage                 Bins      Hits    Misses  Coverage
    ----------------                 ----      ----    ------  --------
    Statements                          7         7         0   100.00%

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

3.

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
ALU1	When reset is asserted, C should be 0.	Assert reset = 1 and check C.	-	Ensure C == 0 after reset.
ALU2	ALU performs **addition** with boundary cases.	Directed test cases: A = MAXNEG, ZERO, MAXNEG B = MAXNEG, ZERO, MAXNEG	-	Ensure C is correct and check overflow conditions.
ALU3	ALU performs **subtraction** with boundary cases.	Directed test cases: A = MAXNEG, ZERO, MAXNEG B = MAXNEG, ZERO, MAXNEG	-	Ensure C is correct and check underflow conditions.
ALU4	ALU performs **bitwise NOT**	Directed test cases for all possible values of A.	-	Ensure C == ~A for all cases.
ALU5	ALU performs **reduction OR** on	Directed test cases for all possible values of B.	-	Ensure C == |B for all cases.

```
module ALU_tb;
    parameter CLK_PERIOD = 10;

    localparam MAXPOS = 7;
    localparam ZERO   = 0;
    localparam MAXNEG = -8;

    localparam ADD           = 2'b00;
    localparam SUB           = 2'b01;
    localparam NOT_A         = 2'b10;
    localparam ReductionOR_B = 2'b11;

    int error_count, correct_count;
    int A_temp, B_temp, opcode_temp;

    // Signals
```

```
    logic clk, reset;
    logic [1:0] opcode;
    logic signed [3:0] A, B;
    logic signed [4:0] C;

    // Instantiate the DUT
    ALU DUT (.*);

    // Clock Generation
    always #(CLK_PERIOD/2) clk = ~clk;

    // Task to apply inputs
    task drive_input (
        input [1:0] in_opcode,
        input signed [3:0] in1,
        input signed [3:0] in2
    );
        opcode = in_opcode;
        A = in1; B = in2;
        @(negedge clk);
    endtask

    function signed [4:0] alu_function(
        input [1:0] opcode,
        input signed [3:0] A,
        input signed [3:0] B
    );
        case (opcode)
            2'b00: alu_function = A + B;  // Addition
            2'b01: alu_function = A - B;  // Subtraction
            2'b10: alu_function = ~A;     // Bitwise NOT of A
            2'b11: alu_function = |B;     // Reduction OR of B
            default: alu_function = 5'b0; // Default case
        endcase
    endfunction


    task check_result(
        input signed [4:0] expected_result
    );
        if (C == expected_result) begin
            correct_count++;
        end else begin
            $display("Error: expected output = %d, got output = %d | A = %d, B
= %d, Opcode = %b",
                     expected_result, C, A, B, opcode);
            error_count++;
        end
    endtask
```

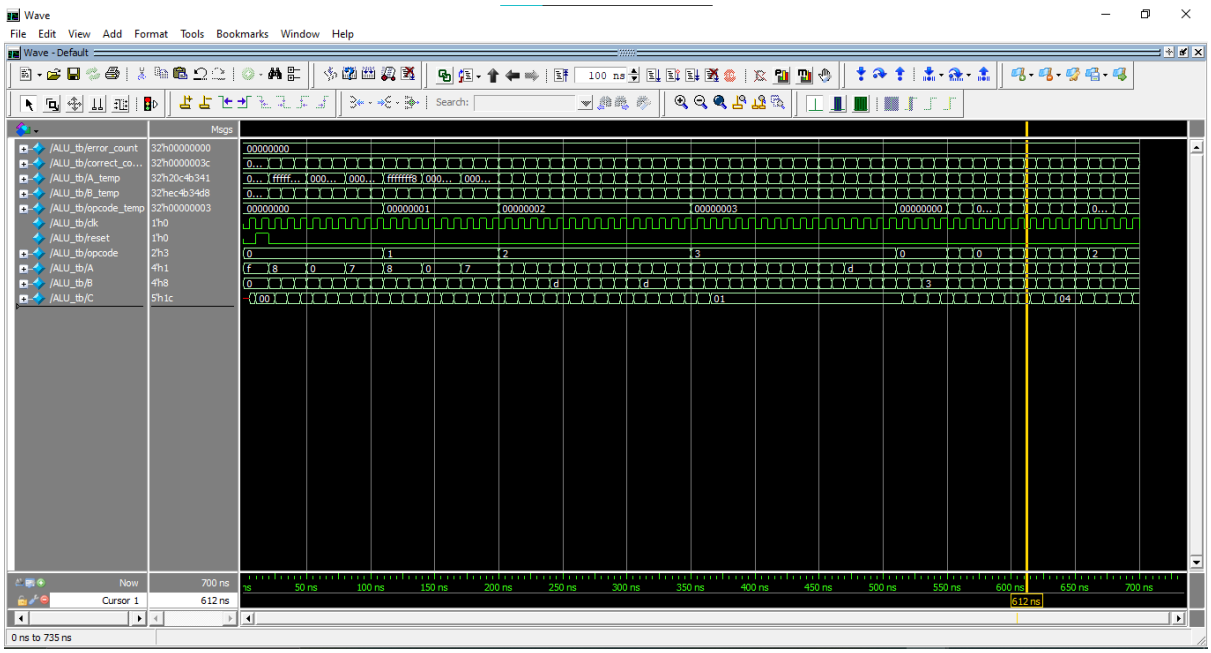
```
task assert_reset;
    reset = 1'b1;        // Assert reset
    @(negedge clk);
    reset = 1'b0;
endtask

task wait_cycles (
    input integer num_cycles
);
    repeat(num_cycles) @(negedge clk);
endtask

initial begin
    initialize_inputs;
    wait_cycles(1);

    // ALU1
    assert_reset;
    check_result(ZERO);

    // ALU2
    opcode_temp = ADD;
    A_temp = MAXNEG; B_temp = MAXNEG;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = MAXNEG; B_temp = ZERO;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = MAXNEG; B_temp = MAXPOS;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = ZERO; B_temp = MAXNEG;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = ZERO; B_temp = ZERO;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = ZERO; B_temp = MAXPOS;
    drive_input(opcode_temp, A_temp, B_temp);
    check_result(alu_function(opcode_temp, A_temp, B_temp));

    A_temp = MAXPOS; B_temp = MAXNEG;
```

```
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXPOS; B_temp = ZERO;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXPOS; B_temp = MAXPOS;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

// ALU3
opcode_temp = SUB;
A_temp = MAXNEG; B_temp = MAXNEG;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXNEG; B_temp = ZERO;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXNEG; B_temp = MAXPOS;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = ZERO; B_temp = MAXNEG;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = ZERO; B_temp = ZERO;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = ZERO; B_temp = MAXPOS;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXPOS; B_temp = MAXNEG;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXPOS; B_temp = ZERO;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));

A_temp = MAXPOS; B_temp = MAXPOS;
drive_input(opcode_temp, A_temp, B_temp);
check_result(alu_function(opcode_temp, A_temp, B_temp));
```

```
        // ALU3
        opcode_temp = NOT_A;
        for (int i = 0; i < 15; i++) begin
            A_temp = i; B_temp = $random;
            drive_input(opcode_temp, A_temp, B_temp);
            check_result(alu_function(opcode_temp, A_temp, B_temp));
        end

        // ALU4
        opcode_temp = ReductionOR_B;
        for (int i = 0; i < 15; i++) begin
            A_temp = $random; B_temp = i;
            drive_input(opcode_temp, A_temp, B_temp);
            check_result(alu_function(opcode_temp, A_temp, B_temp));
        end

        // ALU5
        for (int i = 0; i < 20; i++) begin
            opcode_temp = $urandom_range(0, 3);
            A_temp = $random;
            B_temp = $random;
            drive_input(opcode_temp, A_temp, B_temp);
            check_result(alu_function(opcode_temp, A_temp, B_temp));
        end

        $display("you got correct cases = %0d, and wrong cases = %0d",
correct_count, error_count);
        $stop;
    end

    task initialize_inputs;
        clk = 0;
        reset = 0;
        opcode = 2'b00;
        A = 4'b1111;
        B = 4'b0000;
    endtask

endmodule


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

```
coverage exclude -src ALU.v -line 26 -code b
coverage exclude -src ALU.v -line 26 -code s
```

```
> do run.do
you got correct cases = 69, and wrong cases = 0
```




```
> vcover report ALU_tb.ucdb -details -annotate -all -output coverage_rpt.txt
```

```
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         6         6         0   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                         44        44         0   100.00%
```

4.

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
DSP1	When reset is asserted, the output P should be 0.	Directed at the start of the simulation	-	A checker in the testbench to verify that P is 0.
DSP2	Assign random values to A, B, C, D and verify output.	Randomization	-	A checker in the testbench to ensure P matches with the expected

```
module DSP_tb;
    parameter CLK_PERIOD = 10;

    int error_count, correct_count;
    logic [17:0] A_temp, B_temp, D_temp;
    logic [47:0] C_temp;

    // Signals
    logic clk, rst_n;
    logic [17:0] A, B, D;
    logic [47:0] C;
    logic [47:0] P;  // Output from DSP

    // Instantiate the DUT (DSP module)
    DSP #(.OPERATION("ADD")) DUT (.*);

    // Clock Generation
    always #(CLK_PERIOD/2) clk = ~clk;

    // Task to drive inputs
    task drive_input (
        input [17:0] in_A,
        input [17:0] in_B,
        input [47:0] in_C,
        input [17:0] in_D
    );
        A = in_A; B = in_B; C = in_C; D = in_D;
        @(negedge clk);
    endtask

    // Function to compute expected result
    function [47:0] dsp_function (
```

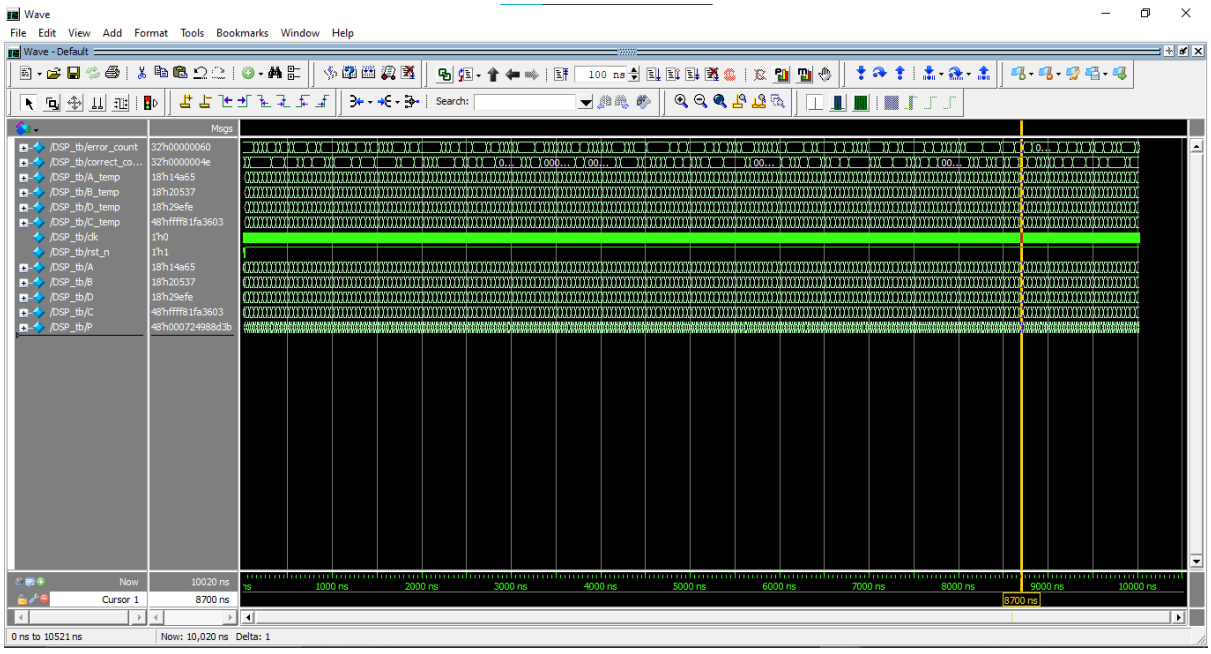
```
        input [17:0] A,
        input [17:0] B,
        input [47:0] C,
        input [17:0] D
    );
        dsp_function = (DUT.OPERATION == "ADD") ? (A * (D + B) + C) :
                       (DUT.OPERATION == "SUBTRACT") ? (A * (D - B) - C) : 0;
    endfunction


    // Task to check the result
    task check_result (
        input [47:0] expected_result
    );
        if (P == expected_result) begin
            correct_count++;
        end else begin
            $display("Error: Expected = %d, Got = %d | A = %d, B = %d, C = %d,
D = %d",
                     expected_result, P, A, B, C, D);
            error_count++;
        end
    endtask

    // Task for reset assertion
    task assert_reset;
        rst_n = 1'b0; // Active-low reset
        @(negedge clk);
        rst_n = 1'b1;
    endtask

    // Task to wait for cycles
    task wait_cycles (
        input integer num_cycles
    );
        repeat(num_cycles) @(negedge clk);
    endtask

    initial begin
        initialize_inputs;
        wait_cycles(1);

        // DSP1
        assert_reset;
        check_result(48'b0);

        // DSP2
        repeat(200) begin
            A_temp = $random;
```

```
            B_temp = $random;
            C_temp = $random;
            D_temp = $random;
            drive_input(A_temp, B_temp, C_temp, D_temp);
            wait_cycles(4);
            check_result(dsp_function(A_temp, B_temp, C_temp, D_temp));
        end

        $display("you got correct cases = %0d, and wrong cases = %0d",
correct_count, error_count);
        $stop;
    end

    task initialize_inputs;
        clk = 0;
        rst_n = 0;
        A = 18'b0;
        B = 18'b0;
        C = 48'b0;
        D = 18'b0;
    endtask

endmodule
```

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

coverage exclude -du DSP -togglenode {mult_out[36]}
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]}

> do run.do
```

```
# Error: Expected =     22604205900, Got =      4782608204 | A =  67984, B = 175093, C = 281473468895308, D = 179579
# Error: Expected =    101861773921, Got =     33600262753 | A = 260397, B = 186409, C =       1579504060, D = 198704
# Error: Expected =     76462717281, Got =     21852879201 | A = 208320, B = 158032, C = 281474184997537, D = 212813
# Error: Expected =    110216146956, Got =     43620823052 | A = 254041, B = 183846, C = 281473706456680, D = 255006
# Error: Expected =     49094171303, Got =     12609231527 | A = 139179, B = 186832, C =        934897007, D = 159192
# Error: Expected =     38929033941, Got =      2431249109 | A = 139228, B = 199332, C =        211258649, D =  78757
# Error: Expected =     33172587178, Got =      6447006378 | A = 101950, B = 246003, C = 281473575300184, D =  93124
# Error: Expected =     41599958370, Got =      2719808866 | A = 148316, B = 148742, C = 281474122993818, D = 137496
# Error: Expected =      9967847419, Got =      3071100923 | A =  26309, B = 119210, C =        219284250, D = 251331
# Error: Expected =     24821614039, Got =      9138587095 | A =  59826, B = 221964, C = 281474791116521, D = 196035
# Error: Expected =     35754474077, Got =     10795481693 | A =  95211, B = 113144, C =       2032927730, D = 241033
.
.
# Error: Expected =     89711369899, Got =     27075993259 | A = 238935, B = 166710, C =        426378034, D = 206969
# Error: Expected =     54416358133, Got =       600554229 | A = 205291, B =  80821, C = 281473986670217, D = 189071
# Error: Expected =     49383301441, Got =      5490434369 | A = 167438, B = 201969, C =       1720065485, D =  82693
# Error: Expected =     23613624556, Got =      1441222892 | A =  84581, B = 132407, C = 281472862402051, D = 171774
# Error: Expected =     32156415666, Got =      5736494770 | A = 100784, B = 121862, C =        152456466, D = 195688
# Error: Expected =     56158975426, Got =      3090281922 | A = 202441, B = 233056, C = 281473422996038, D =  52028
# Error: Expected =     39242501904, Got =     19656150800 | A =  74716, B = 246576, C =       1749415376, D = 255232
# Error: Expected =     14877376095, Got =      5240962655 | A =  36760, B = 212319, C = 281473965789831, D = 219898
# Error: Expected =     39562684344, Got =      8345266104 | A = 119085, B = 131696, C = 281473730275435, D = 210993
```

```
# you got correct cases = 91, and wrong cases = 110
```




```
> vcover report DSP_tb.ucdb -details -annotate -all -output
coverage_rpt.txt
```

```
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   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                        688       688         0    100.00%
```