SV Randomization & Functional Coverage

Assignment3

By: Mohamed Sayed Mohamed Soliman

Question 1

➢ Counter

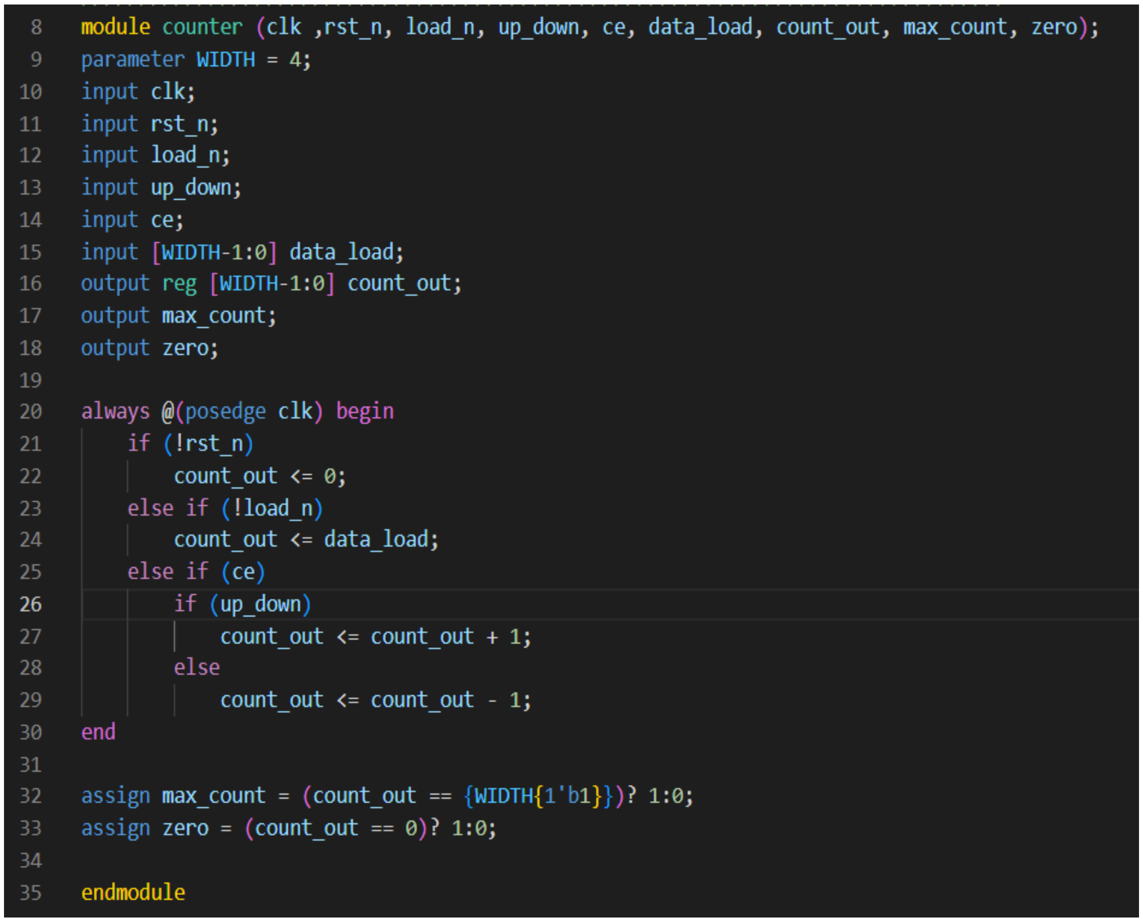
Note: Add the functional coverage in the functional coverage column of your verification plan.

Functional Coverage – to be sampled with the positive edge of the clock

1. Coverpoint for load data when load is asserted
2. Coverpoint for count out if the reset is deasserted, enable is active and up_down is high
    a. Autogenerate bins for all values
3. Coverpoint for count out if the reset is deasserted, enable is active and up_down is high
    a) Transition bin to check when overflow occurs (maximum value to zero)
4. Coverpoint for count out if the reset is deasserted, enable is active and up_down is low
    a. Autogenerate bins for all values
5. Coverpoint for count out if the reset is deasserted, enable is active and up_down is low
    a. Transition bin to check when underflow occurs (zero to maximum value)

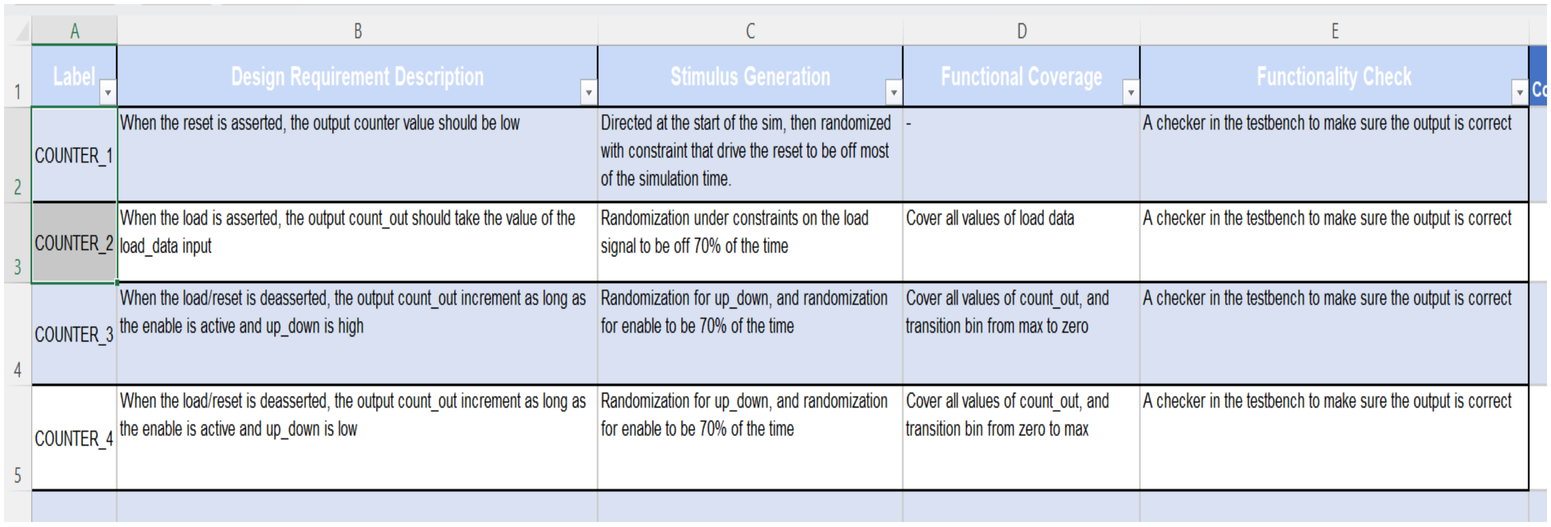
You are free to add more coverpoints to enrich your verification to reach 100% functional coverage. Use a do file to compile the package, design and testbench then simulate and save the coverage. Finally generate the code and functional coverage report.

1. Code Design



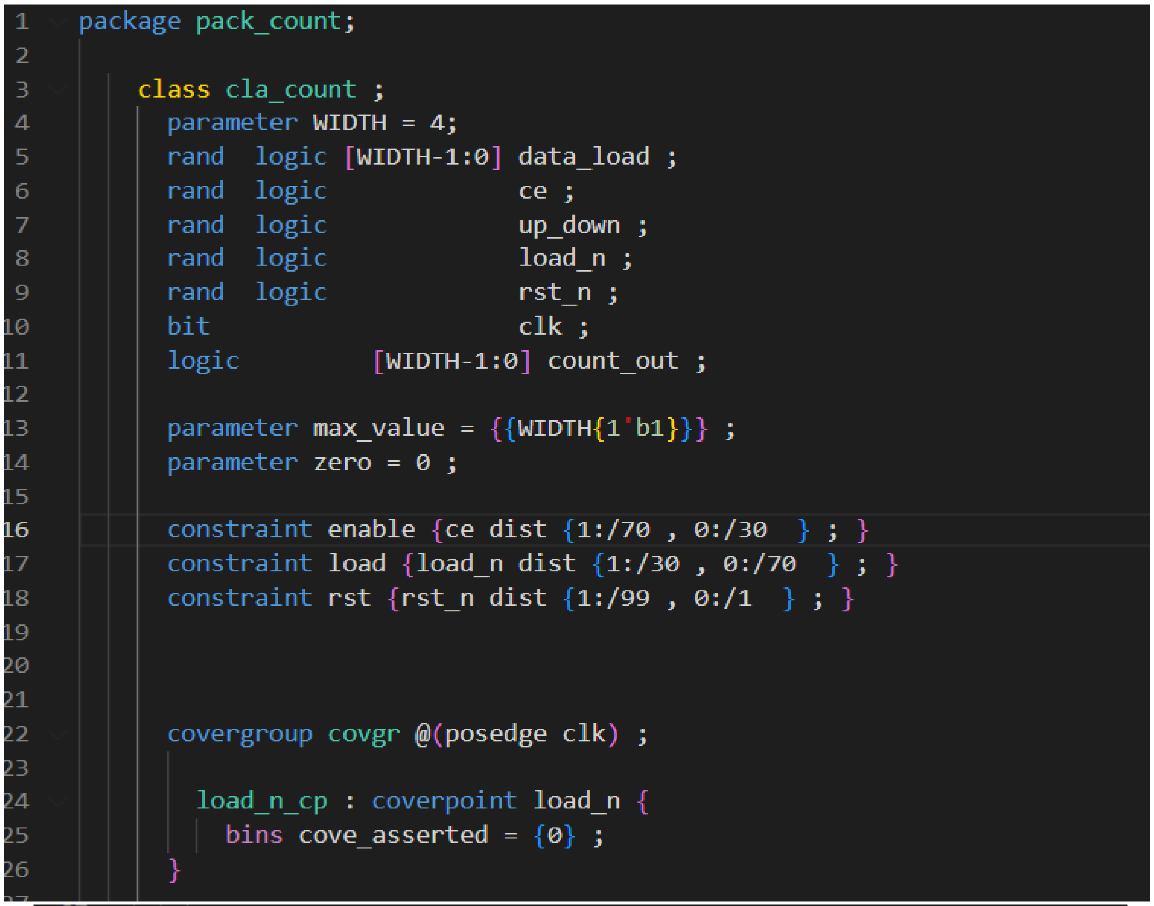
```verilog
module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
parameter WIDTH = 4;
input clk;
input rst_n;
input load_n;
input up_down;
input ce;
input [WIDTH-1:0] data_load;
output reg [WIDTH-1:0] count_out;
output max_count;
output zero;

always @(posedge clk) begin
    if (!rst_n)
        count_out <= 0;
    else if (!load_n)
        count_out <= data_load;
    else if (ce)
        if (up_down)
            count_out <= count_out + 1;
        else
            count_out <= count_out - 1;
end

assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
assign zero = (count_out == 0)? 1:0;

endmodule
```

2. Verification plan

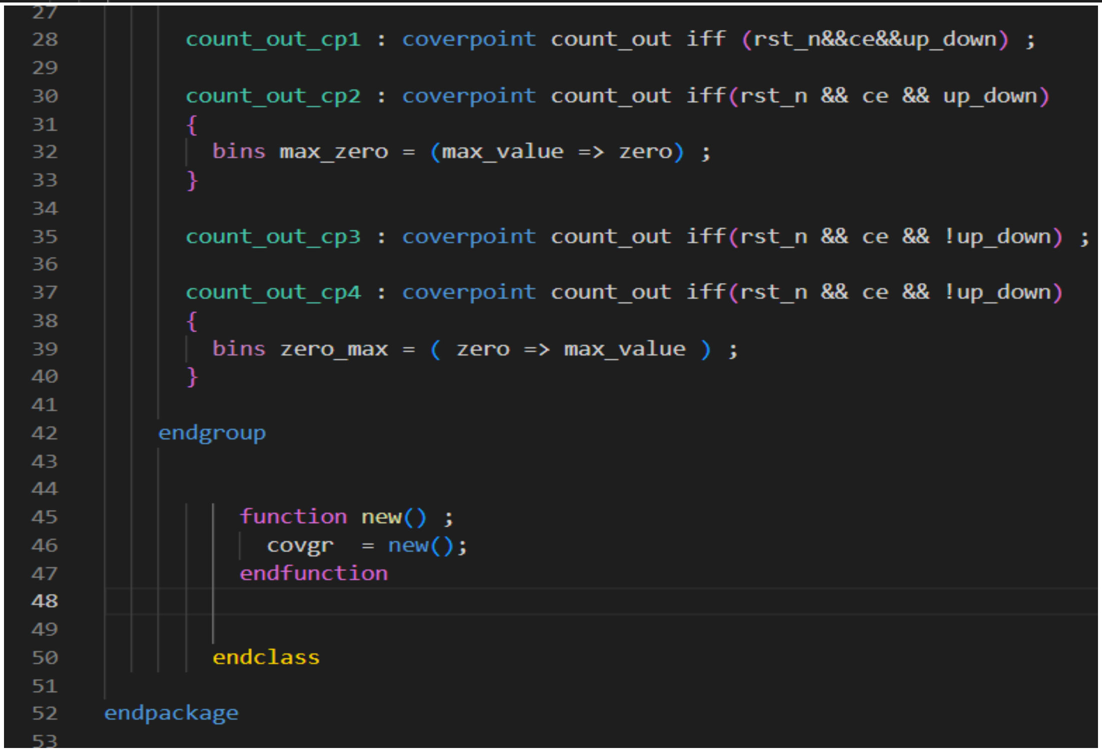


Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check	Co
COUNTER_1	When the reset is asserted, the output counter value should be low	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the testbench to make sure the output is correct	
COUNTER_2	When the load is asserted, the output count_out should take the value of the load_data input	Randomization under constraints on the load signal to be off 70% of the time	Cover all values of load data	A checker in the testbench to make sure the output is correct	
COUNTER_3	When the load/reset is deasserted, the output count_out increment as long as the enable is active and up_down is high	Randomization for up_down, and randomization for enable to be 70% of the time	Cover all values of count_out, and transition bin from max to zero	A checker in the testbench to make sure the output is correct	
COUNTER_4	When the load/reset is deasserted, the output count_out increment as long as the enable is active and up_down is low	Randomization for up_down, and randomization for enable to be 70% of the time	Cover all values of count_out, and transition bin from zero to max	A checker in the testbench to make sure the output is correct	

3. Counter Package

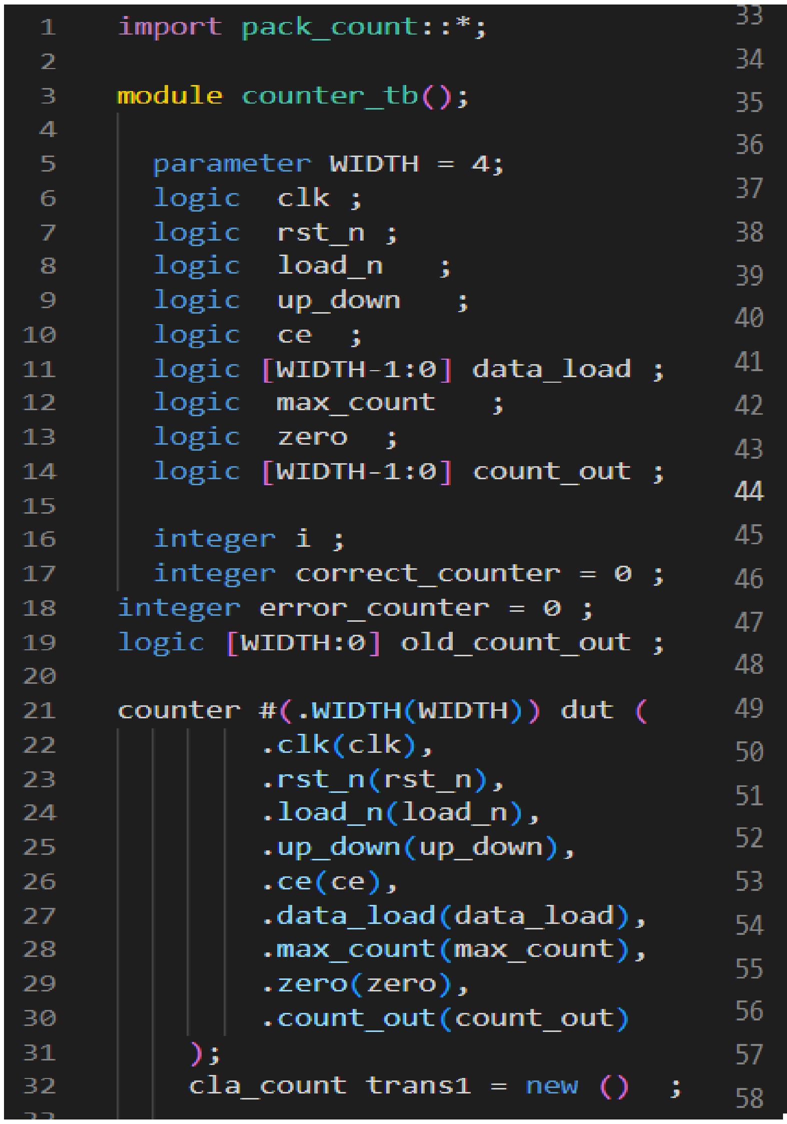


```
package pack_count;

    class cla_count ;
        parameter WIDTH = 4;
        rand  logic [WIDTH-1:0] data_load ;
        rand  logic             ce ;
        rand  logic             up_down ;
        rand  logic             load_n ;
        rand  logic             rst_n ;
        bit                     clk ;
        logic         [WIDTH-1:0] count_out ;

        parameter max_value = {{WIDTH{1'b1}}} ;
        parameter zero = 0 ;

        constraint enable {ce dist {1:/70 , 0:/30  } ; }
        constraint load {load_n dist {1:/30 , 0:/70  } ; }
        constraint rst {rst_n dist {1:/99 , 0:/1  } ; }



        covergroup covgr @(posedge clk) ;

          load_n_cp : coverpoint load_n {
            bins cove_asserted = {0} ;
        }
```

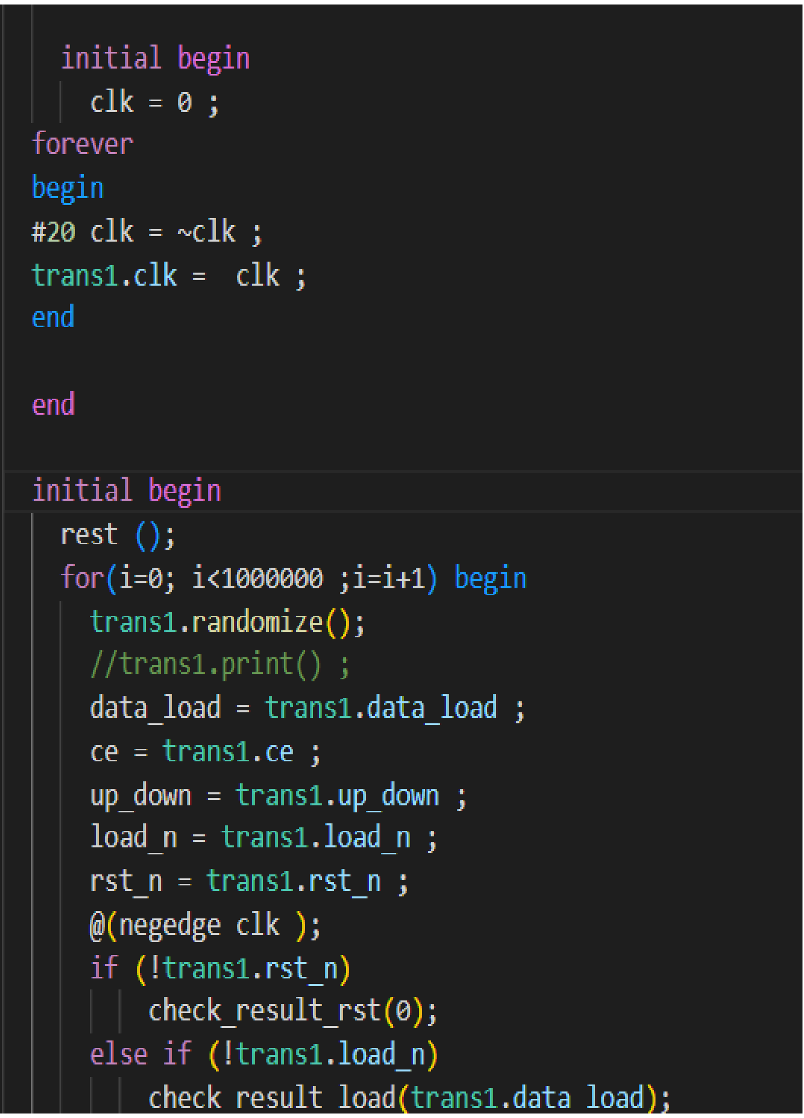


```

            count_out_cp1 : coverpoint count_out iff (rst_n&&ce&&up_down) ;

            count_out_cp2 : coverpoint count_out iff(rst_n && ce && up_down)
            {
              bins max_zero = (max_value => zero) ;
            }

            count_out_cp3 : coverpoint count_out iff(rst_n && ce && !up_down) ;

            count_out_cp4 : coverpoint count_out iff(rst_n && ce && !up_down)
            {
              bins zero_max = ( zero => max_value ) ;
            }

          endgroup


              function new() ;
                covgr  = new();
              endfunction


            endclass

    endpackage

```

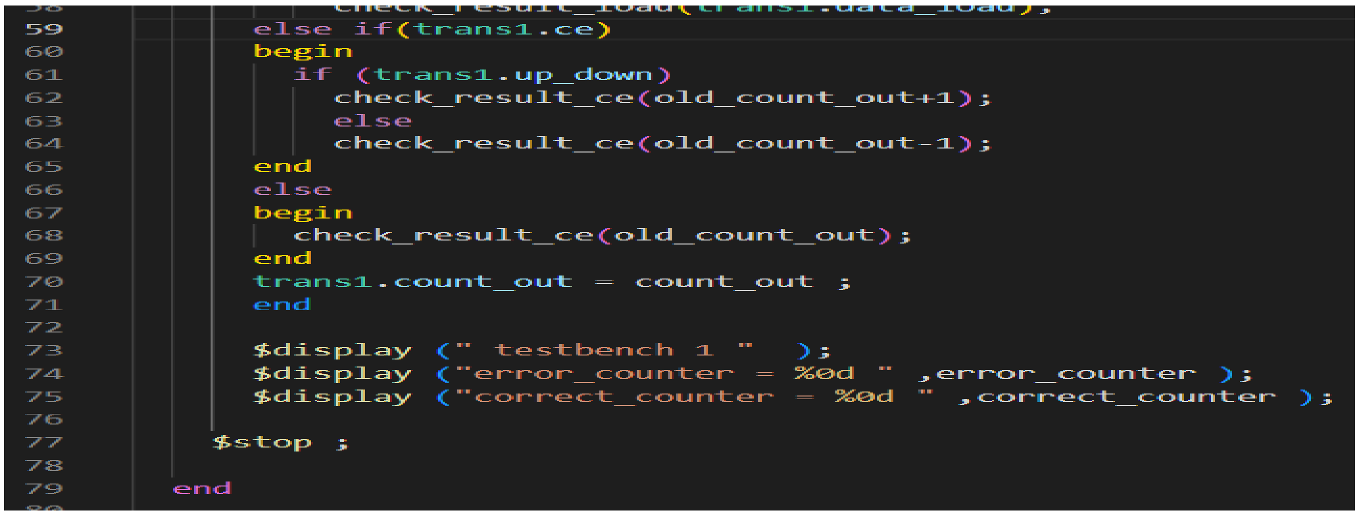
4. Counter Testbench



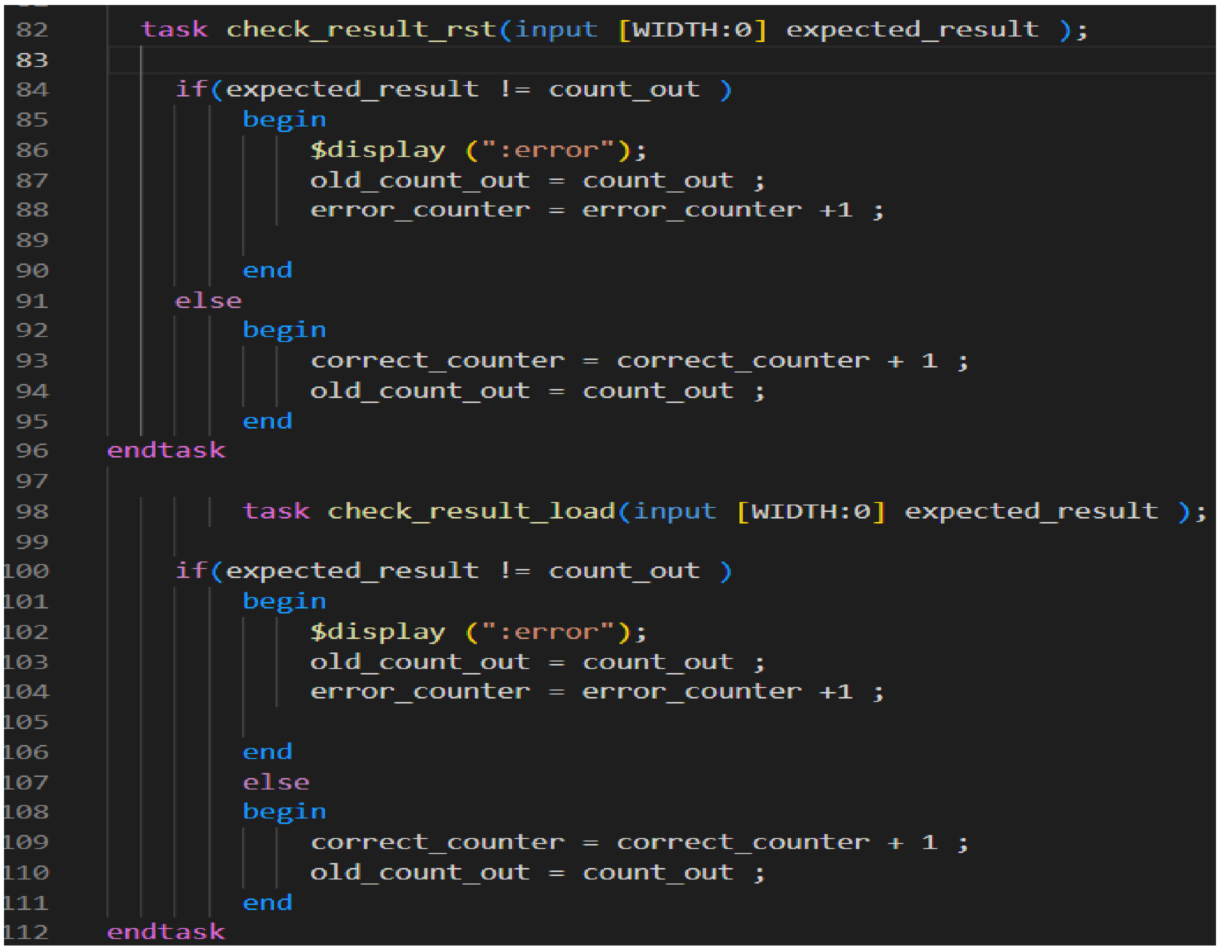
```
import pack_count::*;

module counter_tb();

  parameter WIDTH = 4;
  logic  clk ;
  logic  rst_n ;
  logic  load_n    ;
  logic  up_down    ;
  logic  ce   ;
  logic [WIDTH-1:0] data_load ;
  logic  max_count    ;
  logic  zero   ;
  logic [WIDTH-1:0] count_out ;

  integer i ;
  integer correct_counter = 0 ;
integer error_counter = 0 ;
logic [WIDTH:0] old_count_out ;

counter #(.WIDTH(WIDTH)) dut (
        .clk(clk),
        .rst_n(rst_n),
        .load_n(load_n),
        .up_down(up_down),
        .ce(ce),
        .data_load(data_load),
        .max_count(max_count),
        .zero(zero),
        .count_out(count_out)
    );
    cla_count trans1 = new ()   ;
```



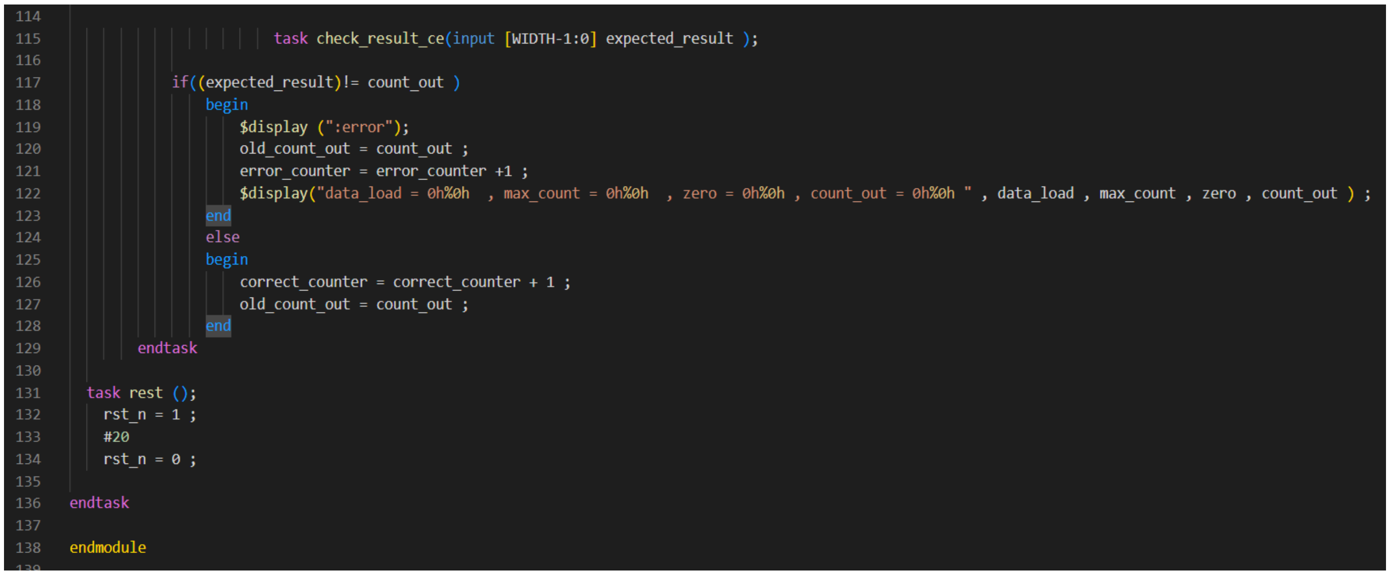
```

  initial begin
    clk = 0 ;
forever
begin
#20 clk = ~clk ;
trans1.clk =  clk ;
end

end

initial begin
  rest ();
  for(i=0; i<1000000 ;i=i+1) begin
    trans1.randomize();
    //trans1.print() ;
    data_load = trans1.data_load ;
    ce = trans1.ce ;
    up_down = trans1.up_down ;
    load_n = trans1.load_n ;
    rst_n = trans1.rst_n ;
    @(negedge clk );
    if (!trans1.rst_n)
      check_result_rst(0);
    else if (!trans1.load_n)
      check_result_load(trans1.data_load);
```



```
    else if(trans1.ce)
    begin
      if (trans1.up_down)
        check_result_ce(old_count_out+1);
        else
        check_result_ce(old_count_out-1);
    end
    else
    begin
      check_result_ce(old_count_out);
    end
    trans1.count_out = count_out ;
    end

    $display (" testbench 1 "   );
    $display ("error_counter = %0d "  ,error_counter );
    $display ("correct_counter = %0d "  ,correct_counter );

  $stop ;

end
```

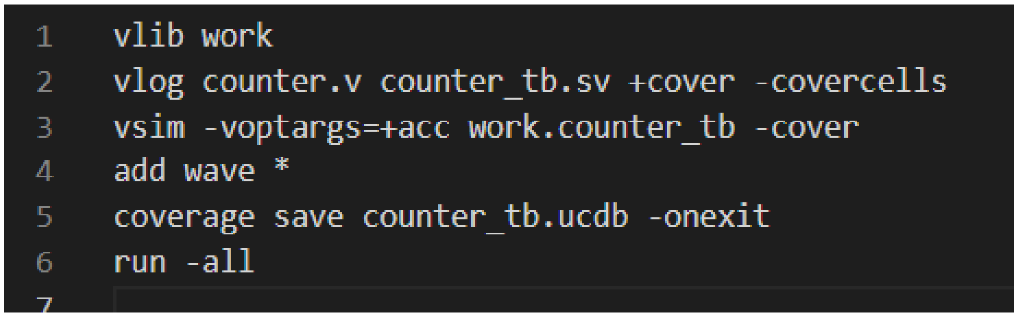


```verilog
 task check_result_rst(input [WIDTH:0] expected_result );

   if(expected_result != count_out )
       begin
           $display (":error");
           old_count_out = count_out ;
           error_counter = error_counter +1 ;

       end
   else
       begin
           correct_counter = correct_counter + 1 ;
           old_count_out = count_out ;
       end
endtask

             task check_result_load(input [WIDTH:0] expected_result );

   if(expected_result != count_out )
       begin
           $display (":error");
           old_count_out = count_out ;
           error_counter = error_counter +1 ;

       end
       else
       begin
           correct_counter = correct_counter + 1 ;
           old_count_out = count_out ;
       end
endtask
```



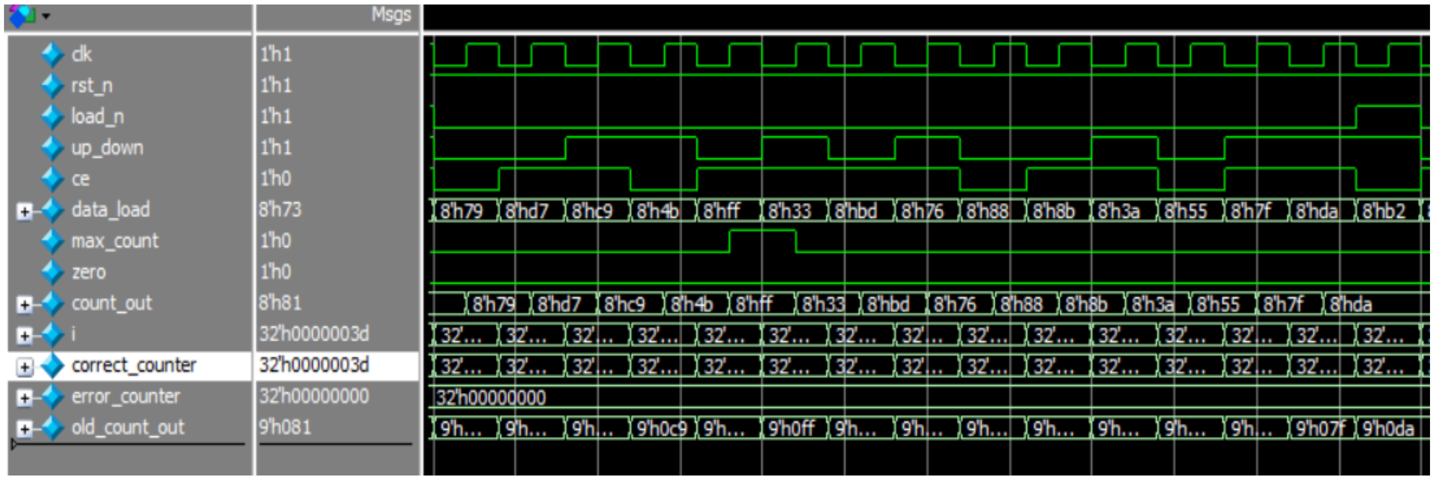
```verilog

                      task check_result_ce(input [WIDTH-1:0] expected_result );

            if((expected_result)!= count_out )
                begin
                    $display (":error");
                    old_count_out = count_out ;
                    error_counter = error_counter +1 ;
                    $display("data_load = 0h%0h  , max_count = 0h%0h  , zero = 0h%0h , count_out = 0h%0h " , data_load , max_count , zero , count_out ) ;
                end
                else
                begin
                    correct_counter = correct_counter + 1 ;
                    old_count_out = count_out ;
                end
        endtask

 task rest ();
  rst_n = 1 ;
  #20
  rst_n = 0 ;

endtask

endmodule
```

5. Do File

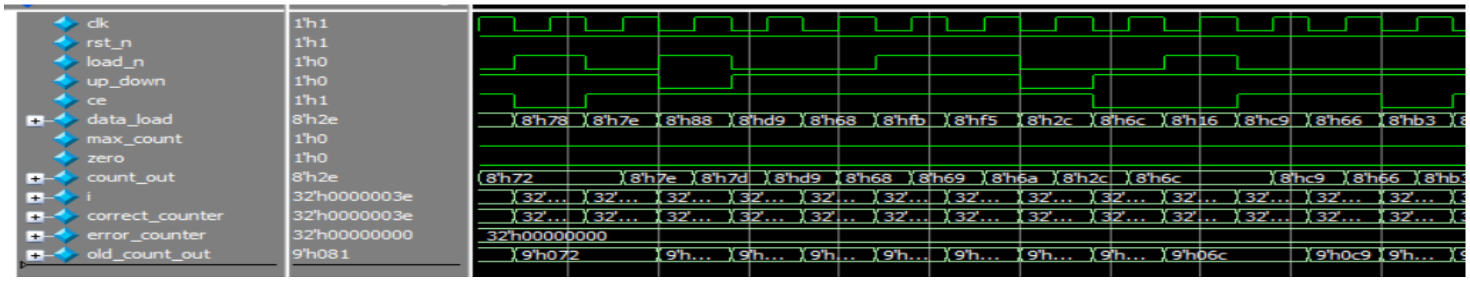


```
vlib work
vlog counter.v counter_tb.sv +cover -covercells
vsim -voptargs=+acc work.counter_tb -cover
add wave *
coverage save counter_tb.ucdb -onexit
run -all
```

6. Waveform

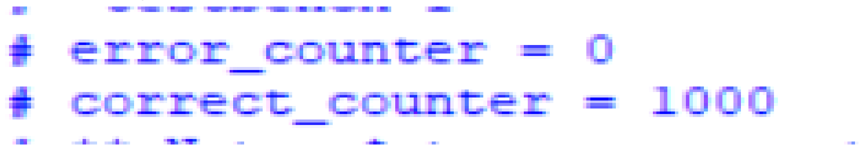


```
clk              1'h1
rst_n            1'h1
load_n           1'h1
up_down          1'h1
ce               1'h0
data_load        8'h73
max_count        1'h0
zero             1'h0
count_out        8'h81
i                32'h0000003d
correct_counter  32'h0000003d
error_counter    32'h00000000
old_count_out    9'h081
```



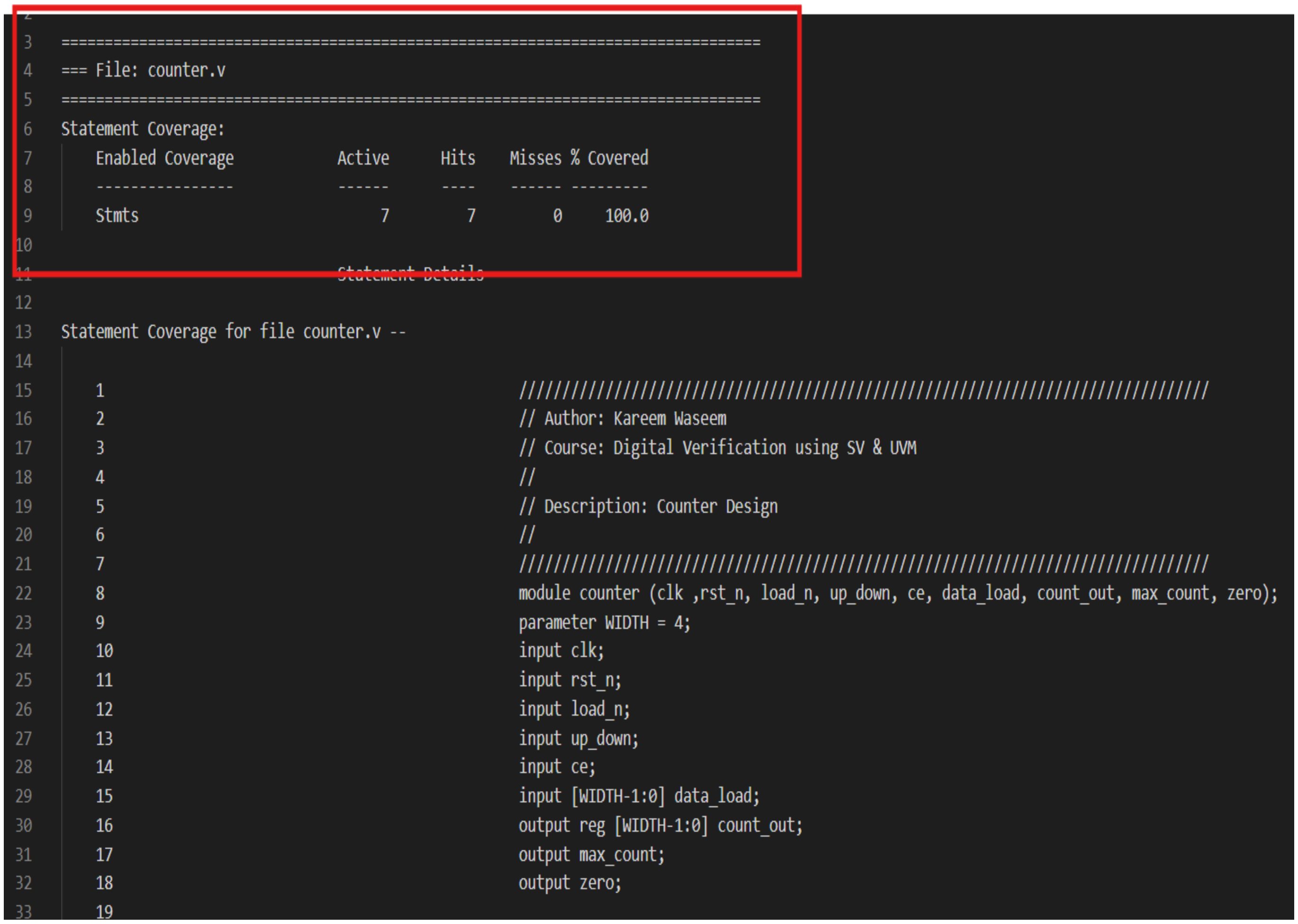
```
clk              1'h1
rst_n            1'h1
load_n           1'h0
up_down          1'h0
ce               1'h1
data_load        8'h2e
max_count        1'h0
zero             1'h0
count_out        8'h2e
i                32'h0000003e
correct_counter  32'h0000003e
error_counter    32'h00000000
old_count_out    9'h081
```

7. Transcript

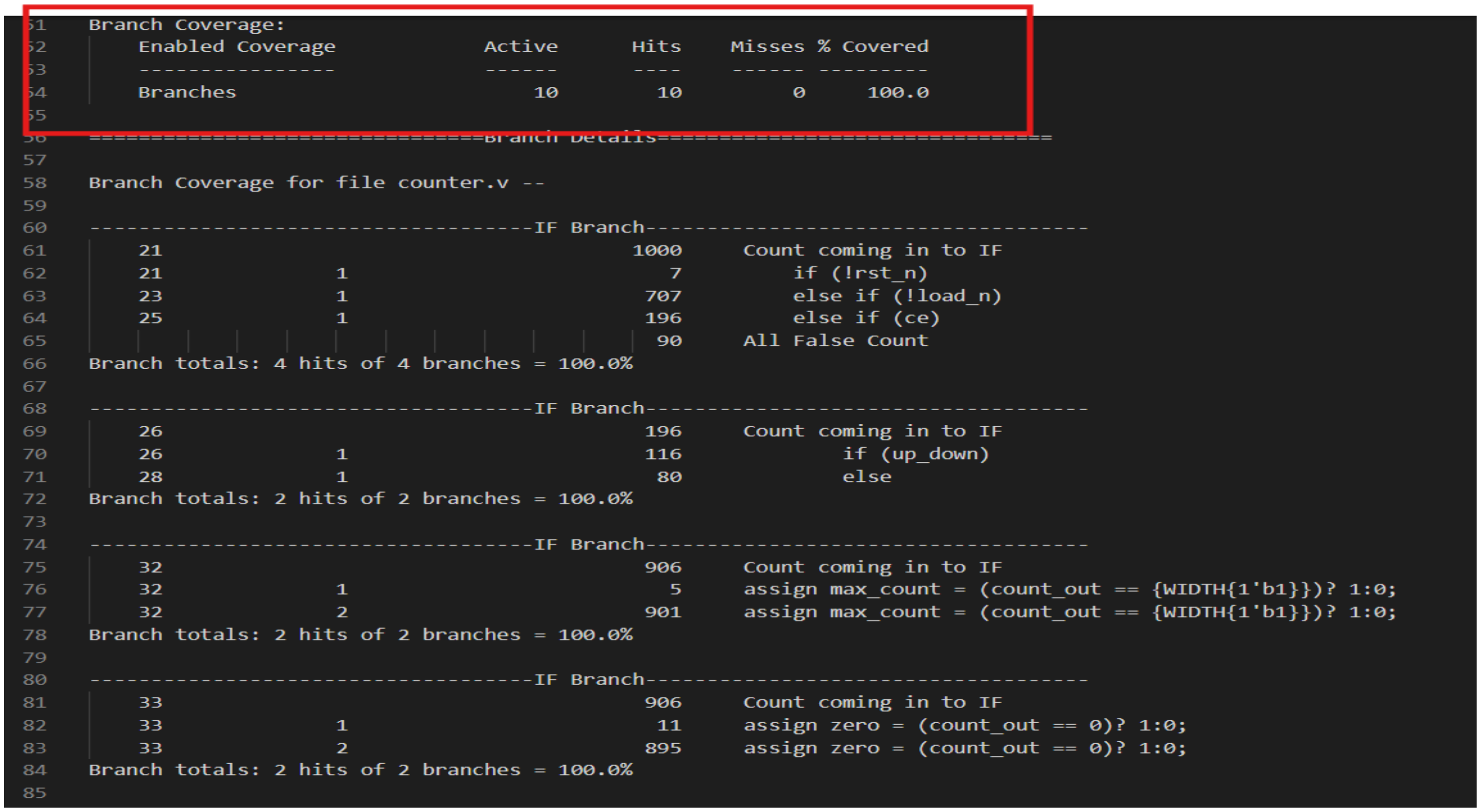


```
# error_counter = 0
# correct_counter = 1000
```

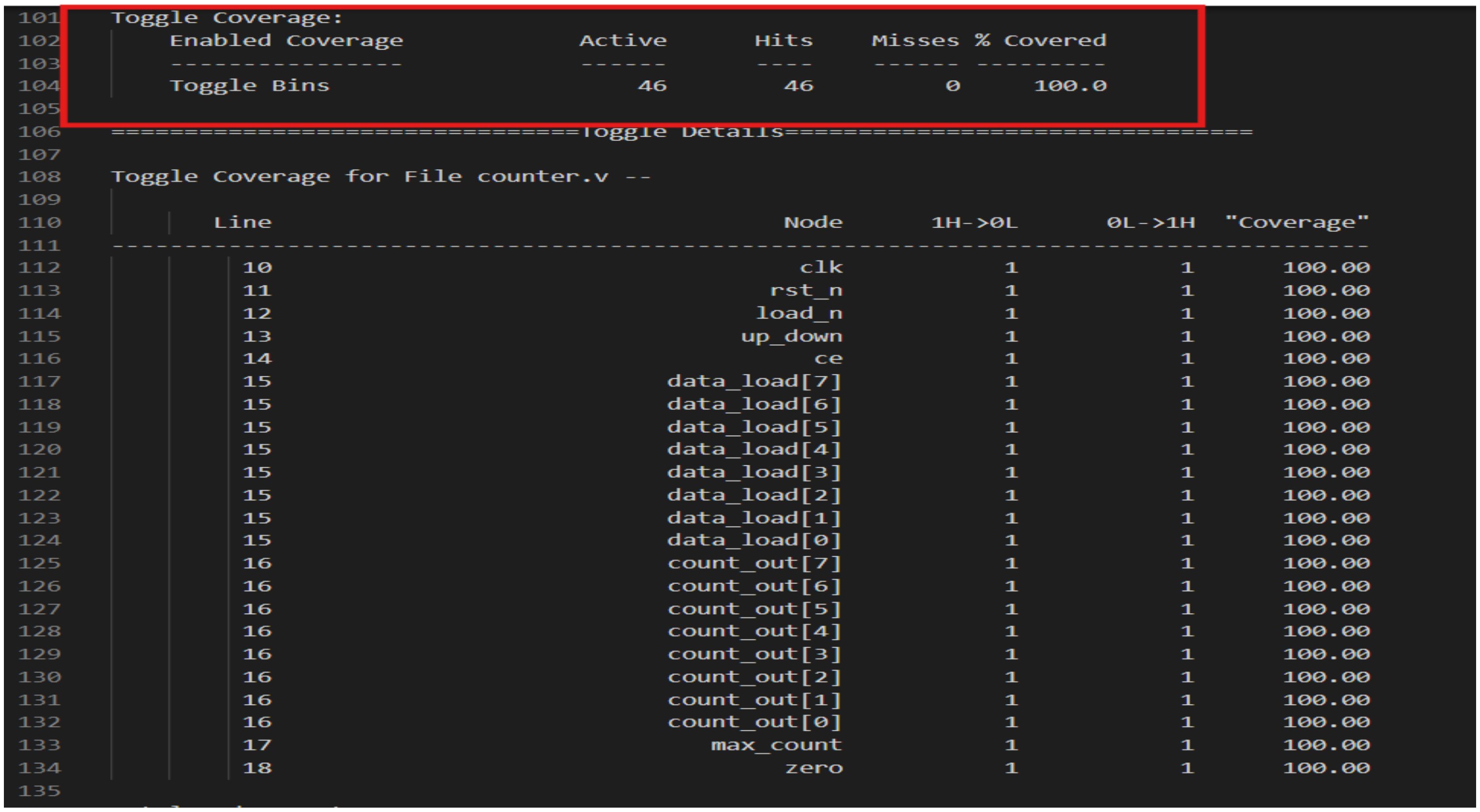
8. Code Coverage



```
================================================================================
=== File: counter.v
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            7         7         0     100.0

                                Statement Details

Statement Coverage for file counter.v --

    1                                           ////////////////////////////////////////////////////////////////////////////////
    2                                           // Author: Kareem Waseem
    3                                           // Course: Digital Verification using SV & UVM
    4                                           //
    5                                           // Description: Counter Design
    6                                           //
    7                                           ////////////////////////////////////////////////////////////////////////////////
    8                                           module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
    9                                           parameter WIDTH = 4;
    10                                          input clk;
    11                                          input rst_n;
    12                                          input load_n;
    13                                          input up_down;
    14                                          input ce;
    15                                          input [WIDTH-1:0] data_load;
    16                                          output reg [WIDTH-1:0] count_out;
    17                                          output max_count;
    18                                          output zero;
    19
```

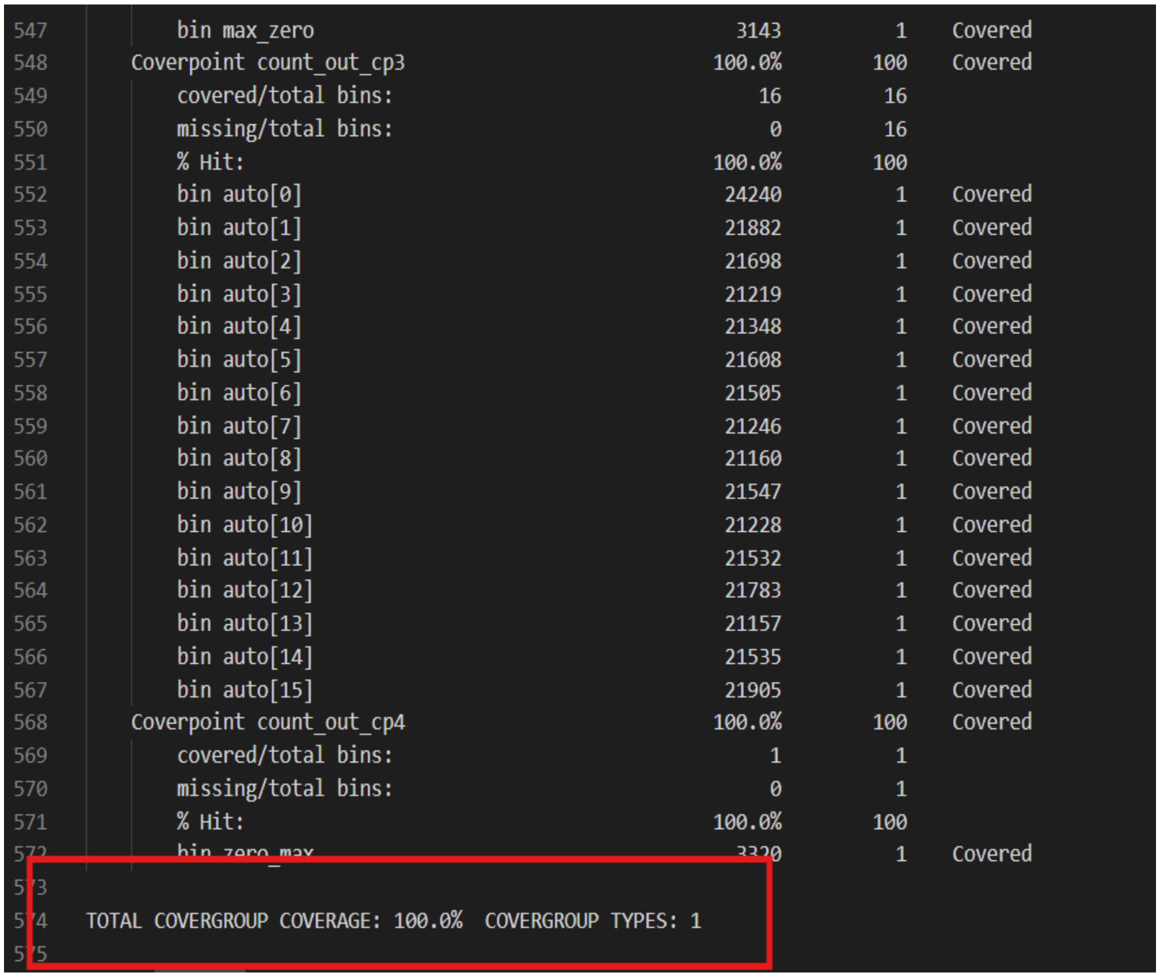


```
Branch Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Branches                          10        10         0     100.0

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

Branch Coverage for file counter.v --

------------------------------------IF Branch------------------------------------
    21                                      1000      Count coming in to IF
    21              1                          7          if (!rst_n)
    23              1                        707          else if (!load_n)
    25              1                        196          else if (ce)
                                              90      All False Count
Branch totals: 4 hits of 4 branches = 100.0%

------------------------------------IF Branch------------------------------------
    26                                       196      Count coming in to IF
    26              1                        116              if (up_down)
    28              1                         80              else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    32                                       906      Count coming in to IF
    32              1                          5      assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
    32              2                        901      assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    33                                       906      Count coming in to IF
    33              1                         11      assign zero = (count_out == 0)? 1:0;
    33              2                        895      assign zero = (count_out == 0)? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%
```



```
Toggle Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Toggle Bins                       46        46         0     100.0
================================Toggle Details================================

Toggle Coverage for File counter.v --

       Line                                 Node      1H->0L      0L->1H  "Coverage"
---------------------------------------------------------------------------------------
         10                                  clk           1           1      100.00
         11                                rst_n           1           1      100.00
         12                               load_n           1           1      100.00
         13                              up_down           1           1      100.00
         14                                   ce           1           1      100.00
         15                         data_load[7]           1           1      100.00
         15                         data_load[6]           1           1      100.00
         15                         data_load[5]           1           1      100.00
         15                         data_load[4]           1           1      100.00
         15                         data_load[3]           1           1      100.00
         15                         data_load[2]           1           1      100.00
         15                         data_load[1]           1           1      100.00
         15                         data_load[0]           1           1      100.00
         16                         count_out[7]           1           1      100.00
         16                         count_out[6]           1           1      100.00
         16                         count_out[5]           1           1      100.00
         16                         count_out[4]           1           1      100.00
         16                         count_out[3]           1           1      100.00
         16                         count_out[2]           1           1      100.00
         16                         count_out[1]           1           1      100.00
         16                         count_out[0]           1           1      100.00
         17                            max_count           1           1      100.00
         18                                 zero           1           1      100.00
```

9. function Coverage



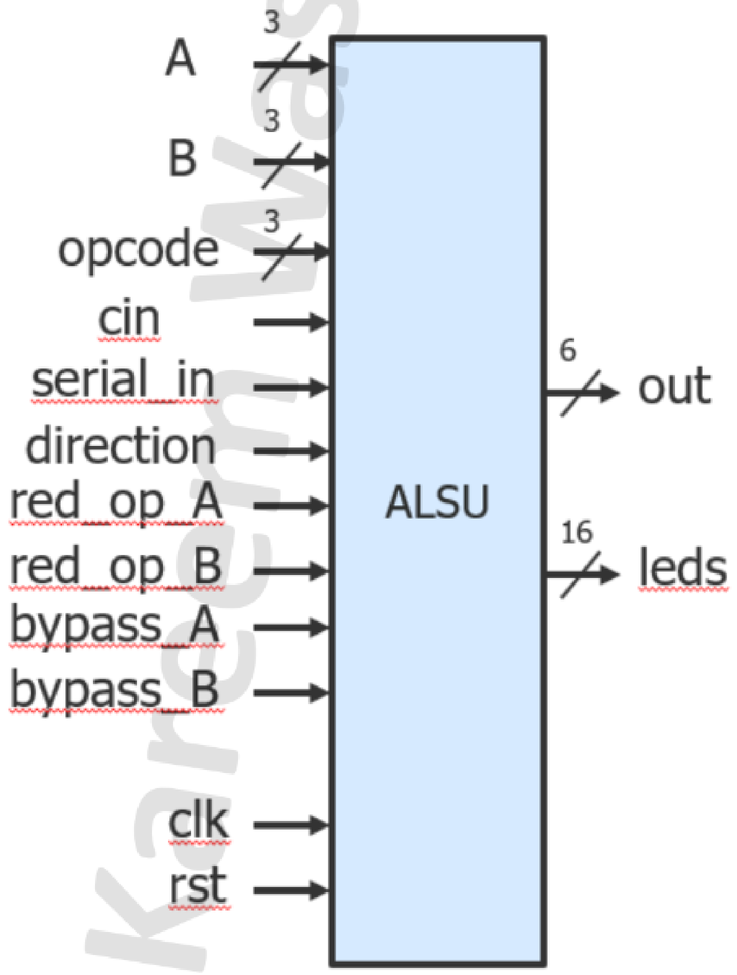
```
        bin max_zero                              3143          1    Covered
    Coverpoint count_out_cp3                    100.0%        100    Covered
        covered/total bins:                         16         16
        missing/total bins:                          0         16
        % Hit:                                  100.0%        100
        bin auto[0]                              24240          1    Covered
        bin auto[1]                              21882          1    Covered
        bin auto[2]                              21698          1    Covered
        bin auto[3]                              21219          1    Covered
        bin auto[4]                              21348          1    Covered
        bin auto[5]                              21608          1    Covered
        bin auto[6]                              21505          1    Covered
        bin auto[7]                              21246          1    Covered
        bin auto[8]                              21160          1    Covered
        bin auto[9]                              21547          1    Covered
        bin auto[10]                             21228          1    Covered
        bin auto[11]                             21532          1    Covered
        bin auto[12]                             21783          1    Covered
        bin auto[13]                             21157          1    Covered
        bin auto[14]                             21535          1    Covered
        bin auto[15]                             21905          1    Covered
    Coverpoint count_out_cp4                    100.0%        100    Covered
        covered/total bins:                          1          1
        missing/total bins:                          0          1
        % Hit:                                  100.0%        100
        bin zero_max                              3320          1    Covered

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1

```

Question 2

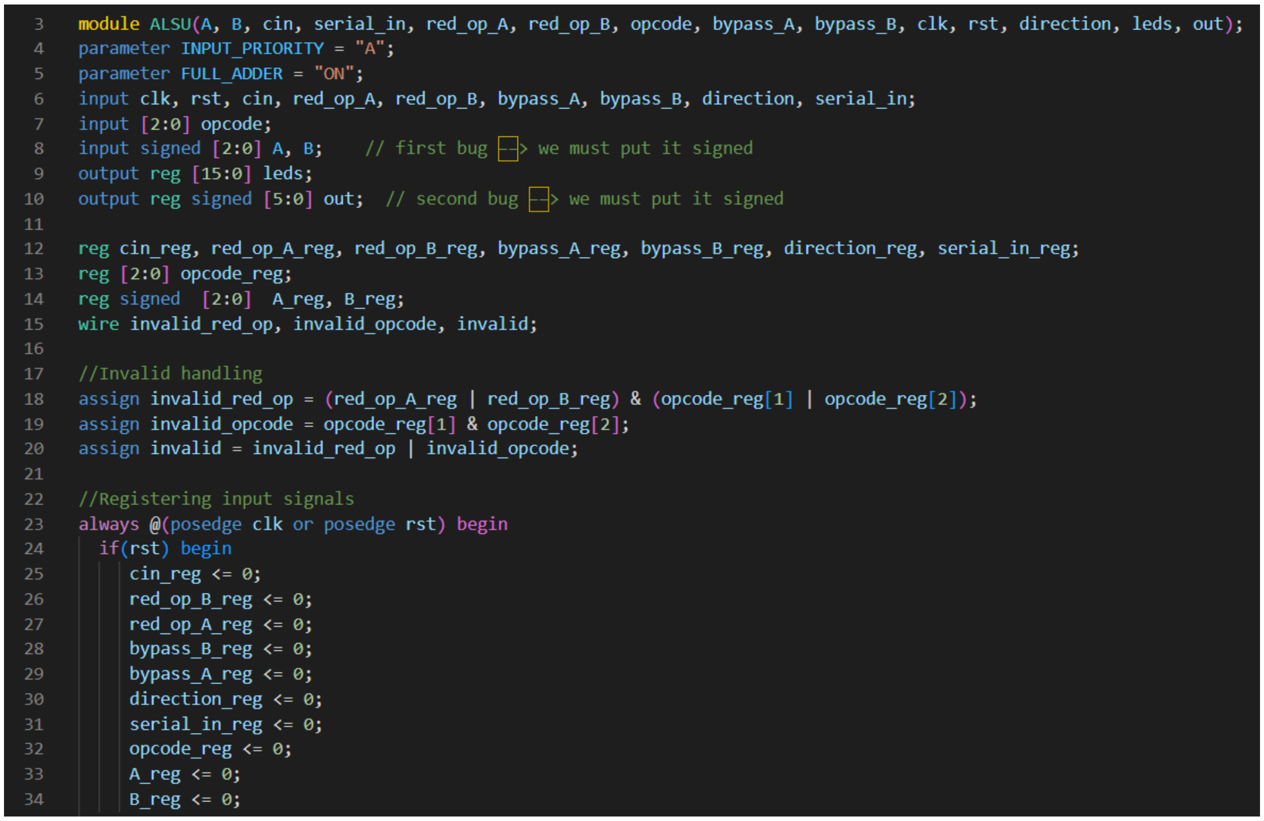
➢ ALSU

2) ALSU is a logic unit that can perform logical, arithmetic, and shift operations on input ports

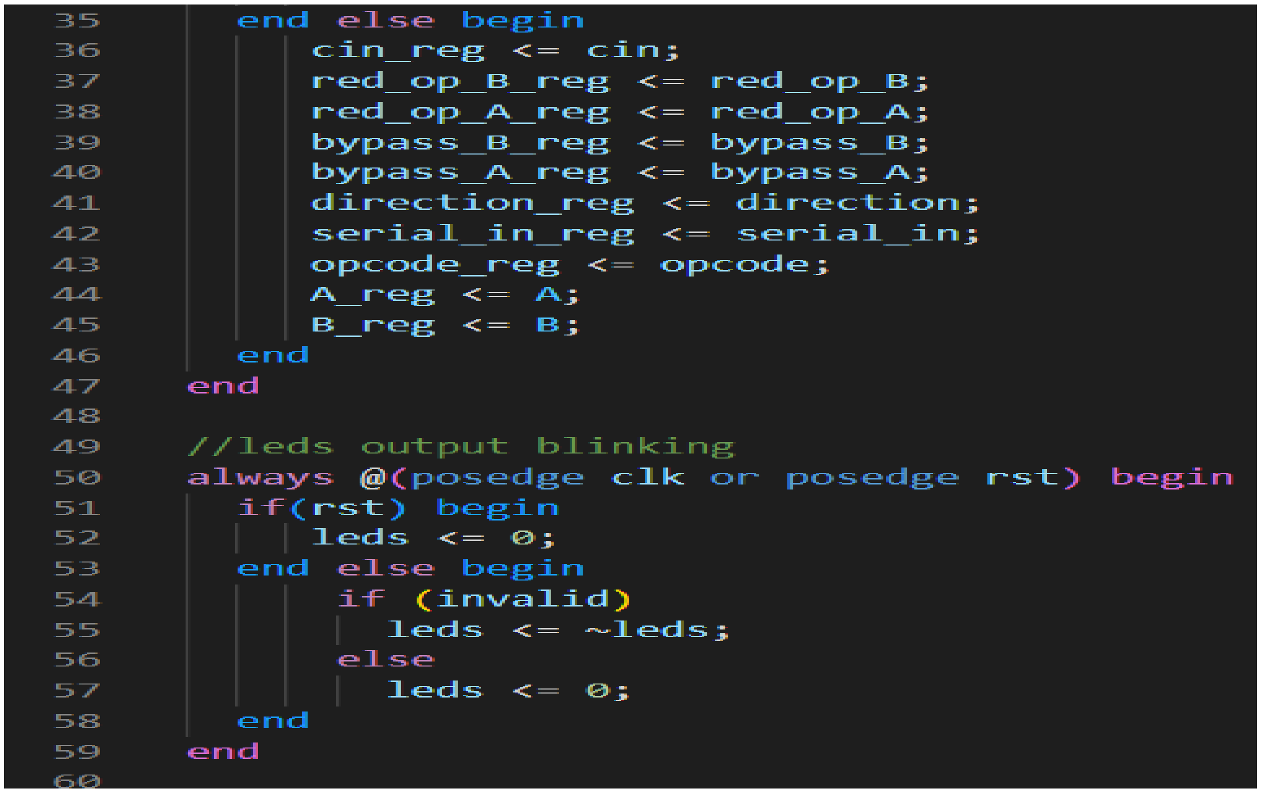
- Input ports A and B have various operations that can take place depending on the value of the opcode.
- Each input bit except for the clk and rst will be sampled at the rising edge before any processing so a D-FF is expected for each input bit at the design entry.
- The output of the ALSU is registered and is available at the rising edge of the clock.



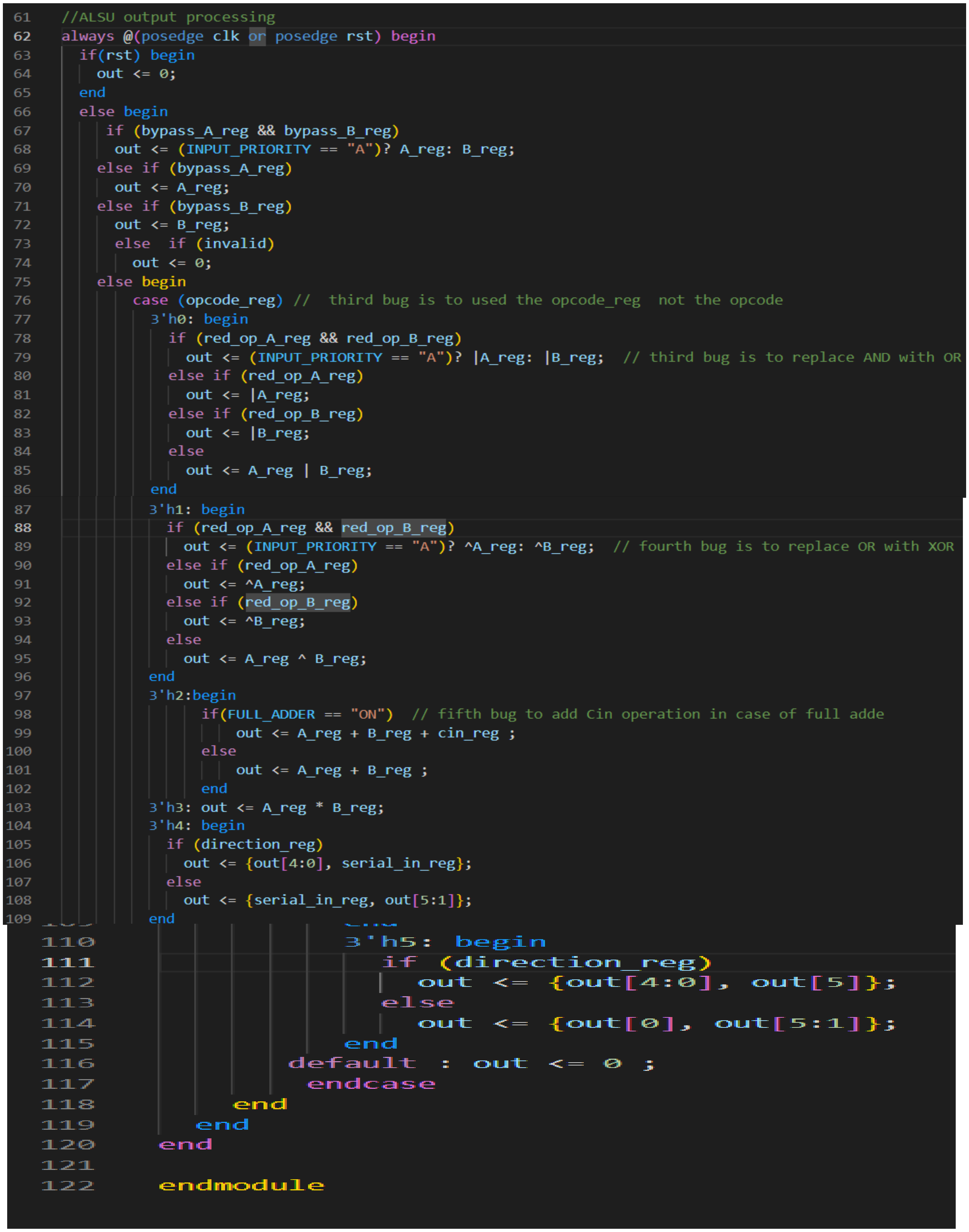
1. Code Design



```
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;    // first bug --> we must put it signed
output reg [15:0] leds;
output reg signed [5:0] out;  // second bug --> we must put it signed

reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg;
reg signed  [2:0]  A_reg, B_reg;
wire invalid_red_op, invalid_opcode, invalid;

//Invalid handling
assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
assign invalid = invalid_red_op | invalid_opcode;

//Registering input signals
always @(posedge clk or posedge rst) begin
  if(rst) begin
     cin_reg <= 0;
     red_op_B_reg <= 0;
     red_op_A_reg <= 0;
     bypass_B_reg <= 0;
     bypass_A_reg <= 0;
     direction_reg <= 0;
     serial_in_reg <= 0;
     opcode_reg <= 0;
     A_reg <= 0;
     B_reg <= 0;
```

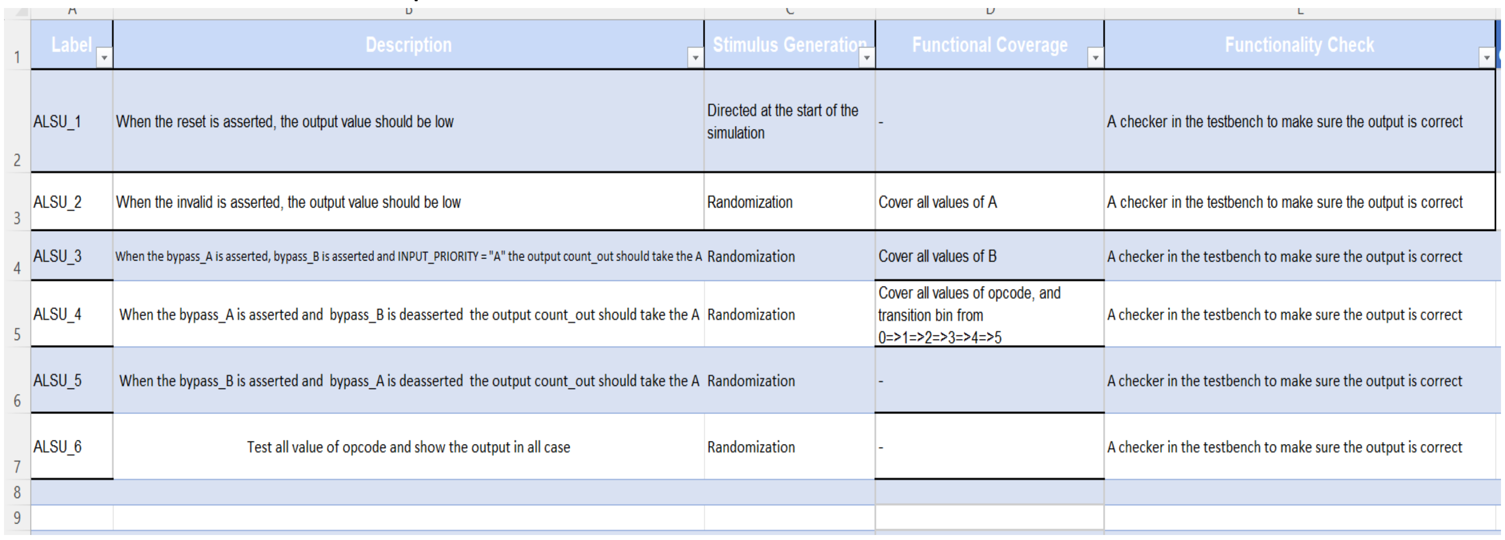


```
  end else begin
     cin_reg <= cin;
     red_op_B_reg <= red_op_B;
     red_op_A_reg <= red_op_A;
     bypass_B_reg <= bypass_B;
     bypass_A_reg <= bypass_A;
     direction_reg <= direction;
     serial_in_reg <= serial_in;
     opcode_reg <= opcode;
     A_reg <= A;
     B_reg <= B;
  end
end

//leds output blinking
always @(posedge clk or posedge rst) begin
  if(rst) begin
     leds <= 0;
  end else begin
       if (invalid)
         leds <= ~leds;
       else
         leds <= 0;
  end
end

```



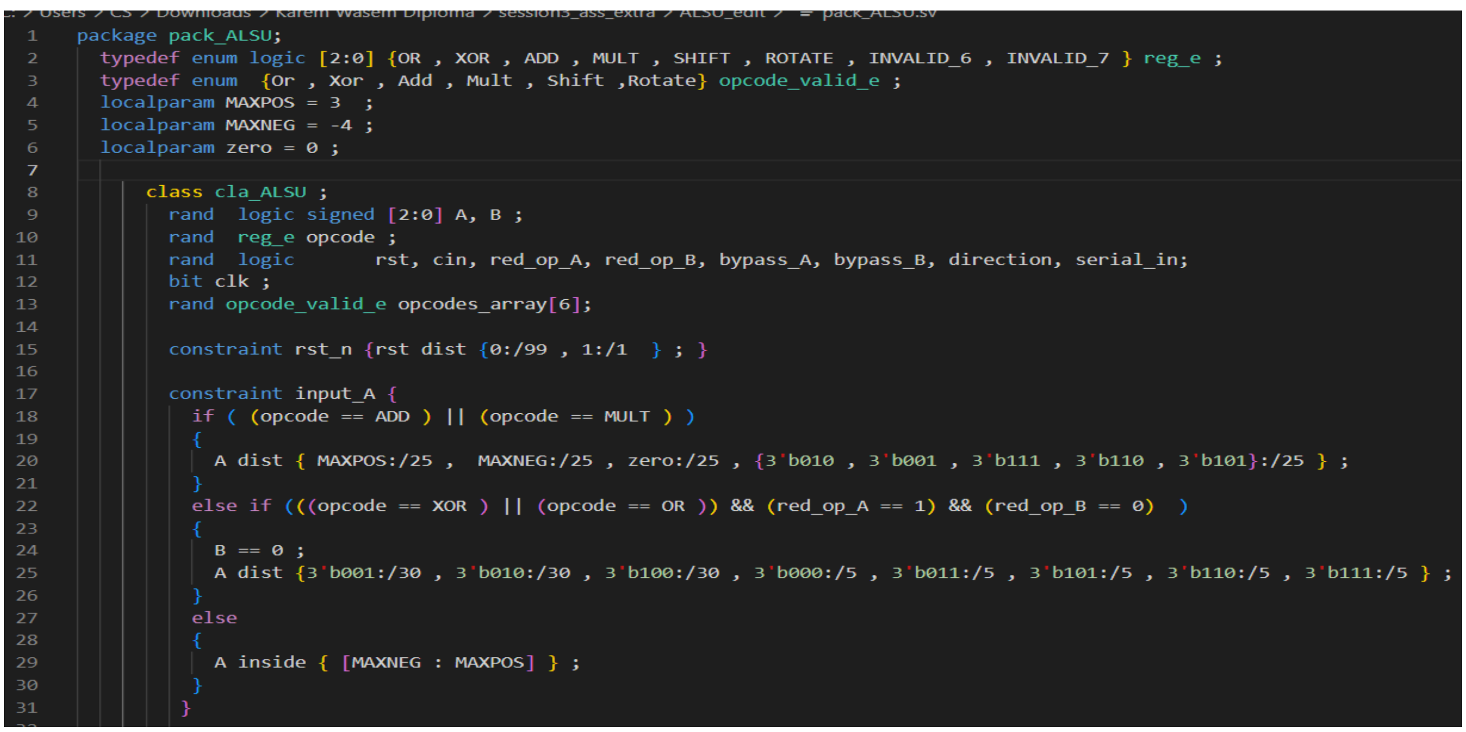
```verilog
//ALSU output processing
always @(posedge clk or posedge rst) begin
  if(rst) begin
    out <= 0;
  end
  else begin
     if (bypass_A_reg && bypass_B_reg)
      out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    else if (bypass_A_reg)
      out <= A_reg;
    else if (bypass_B_reg)
      out <= B_reg;
      else  if (invalid)
        out <= 0;
    else begin
        case (opcode_reg) //  third bug is to used the opcode_reg  not the opcode
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;  // third bug is to replace AND with OR
            else if (red_op_A_reg)
              out <= |A_reg;
            else if (red_op_B_reg)
              out <= |B_reg;
            else
              out <= A_reg | B_reg;
          end
          3'h1: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;  // fourth bug is to replace OR with XOR
            else if (red_op_A_reg)
              out <= ^A_reg;
            else if (red_op_B_reg)
              out <= ^B_reg;
            else
              out <= A_reg ^ B_reg;
          end
          3'h2:begin
                if(FULL_ADDER == "ON")  // fifth bug to add Cin operation in case of full adde
                    out <= A_reg + B_reg + cin_reg ;
                else
                    out <= A_reg + B_reg ;
                end
          3'h3: out <= A_reg * B_reg;
          3'h4: begin
            if (direction_reg)
              out <= {out[4:0], serial_in_reg};
            else
              out <= {serial_in_reg, out[5:1]};
          end
          3'h5: begin
            if (direction_reg)
              out <= {out[4:0], out[5]};
            else
              out <= {out[0], out[5:1]};
          end
        default : out <= 0 ;
        endcase
      end
    end
  end

  endmodule
```

2. Verification plan



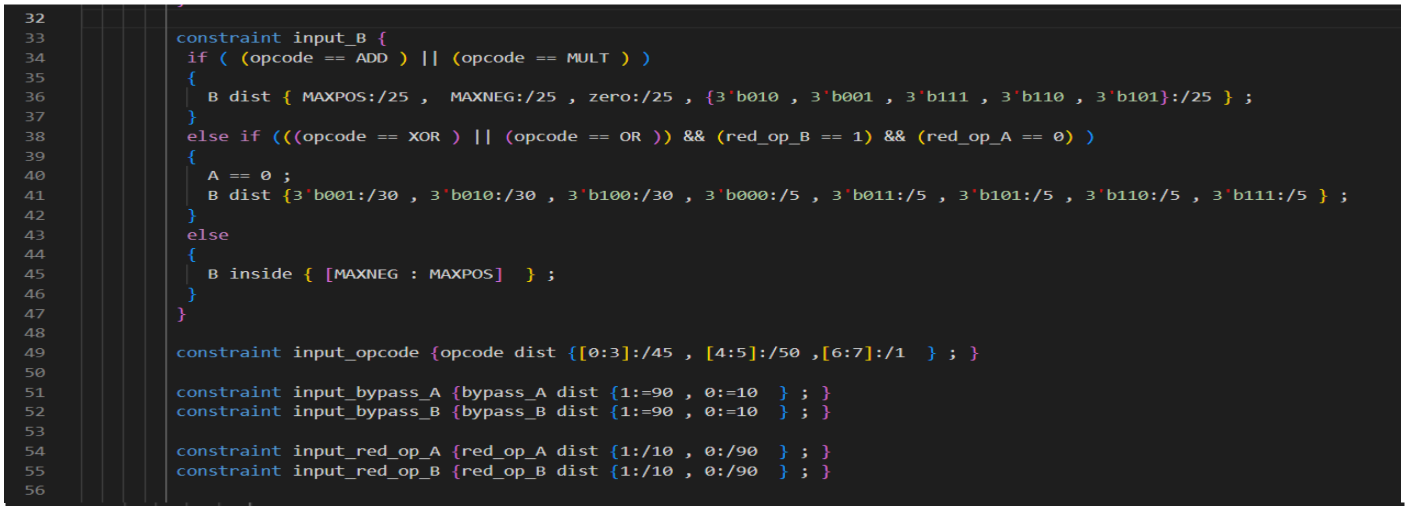
Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
ALSU_2	When the invalid is asserted, the output value should be low	Randomization	Cover all values of A	A checker in the testbench to make sure the output is correct
ALSU_3	When the bypass_A is asserted, bypass_B is asserted and INPUT_PRIORITY = "A" the output count_out should take the A	Randomization	Cover all values of B	A checker in the testbench to make sure the output is correct
ALSU_4	When the bypass_A is asserted and bypass_B is deasserted the output count_out should take the A	Randomization	Cover all values of opcode, and transition bin from 0=>1=>2=>3=>4=>5	A checker in the testbench to make sure the output is correct
ALSU_5	When the bypass_B is asserted and bypass_A is deasserted the output count_out should take the A	Randomization	-	A checker in the testbench to make sure the output is correct
ALSU_6	Test all value of opcode and show the output in all case	Randomization	-	A checker in the testbench to make sure the output is correct

3. ALSU Package

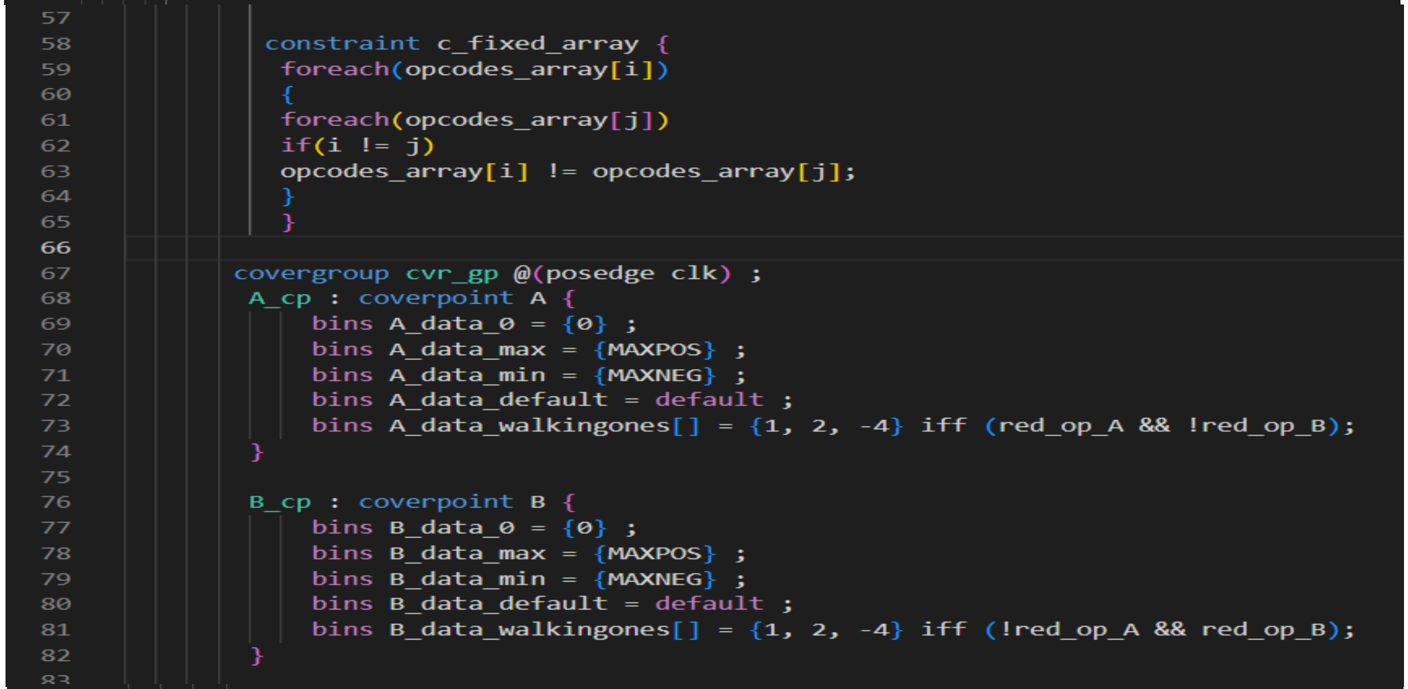


C: > Users > CS > Downloads > Karem Wasem Diploma > session3_ass_extra > ALSU_edit > ≡ pack_ALSU.sv

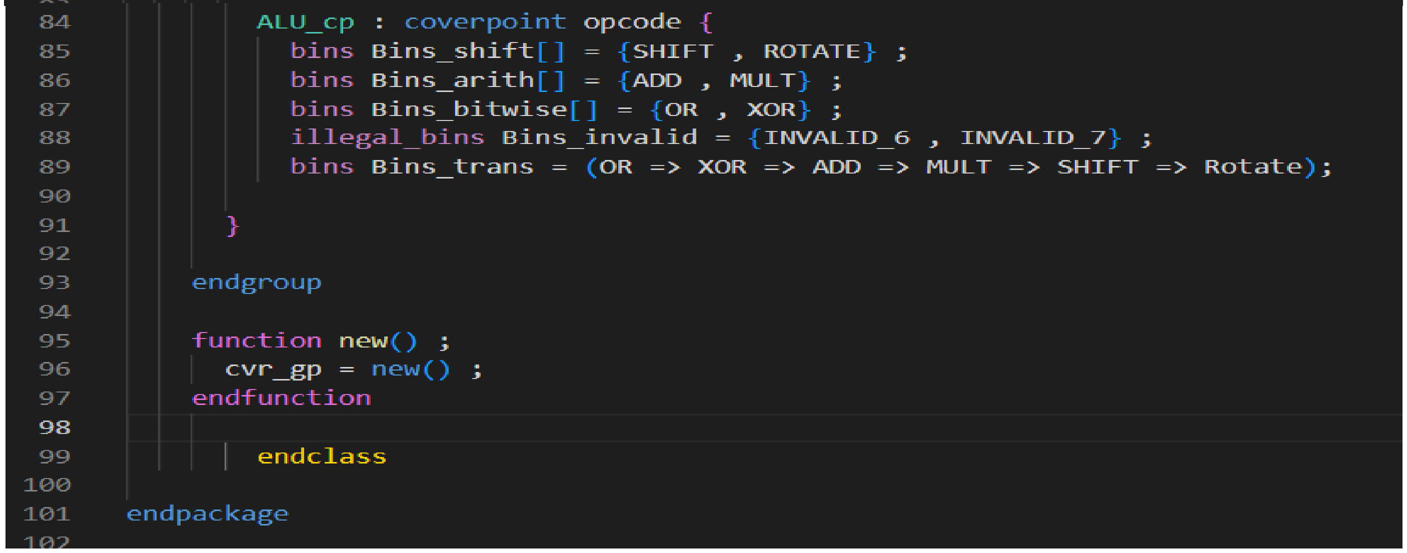
```
package pack_ALSU;
  typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
  typedef enum  {Or , Xor , Add , Mult , Shift ,Rotate} opcode_valid_e ;
  localparam MAXPOS = 3  ;
  localparam MAXNEG = -4 ;
  localparam zero = 0 ;

    class cla_ALSU ;
      rand  logic signed [2:0] A, B ;
      rand  reg_e opcode ;
      rand  logic       rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
      bit clk ;
      rand opcode_valid_e opcodes_array[6];

      constraint rst_n {rst dist {0:/99 , 1:/1  } ; }

      constraint input_A {
        if ( (opcode == ADD ) || (opcode == MULT ) )
        {
          A dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
        }
        else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0)  )
        {
          B == 0 ;
          A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
        }
        else
        {
          A inside { [MAXNEG : MAXPOS] } ;
        }
       }
```



```
constraint input_B {
 if ( (opcode == ADD ) || (opcode == MULT ) )
 {
   B dist { MAXPOS:/25 ,  MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
 }
 else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
 {
   A == 0 ;
   B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
 }
 else
 {
   B inside { [MAXNEG : MAXPOS]  } ;
 }
}

constraint input_opcode {opcode dist {[0:3]:/45 , [4:5]:/50 ,[6:7]:/1  } ; }

constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10  } ; }
constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10  } ; }

constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90  } ; }
constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90  } ; }
```

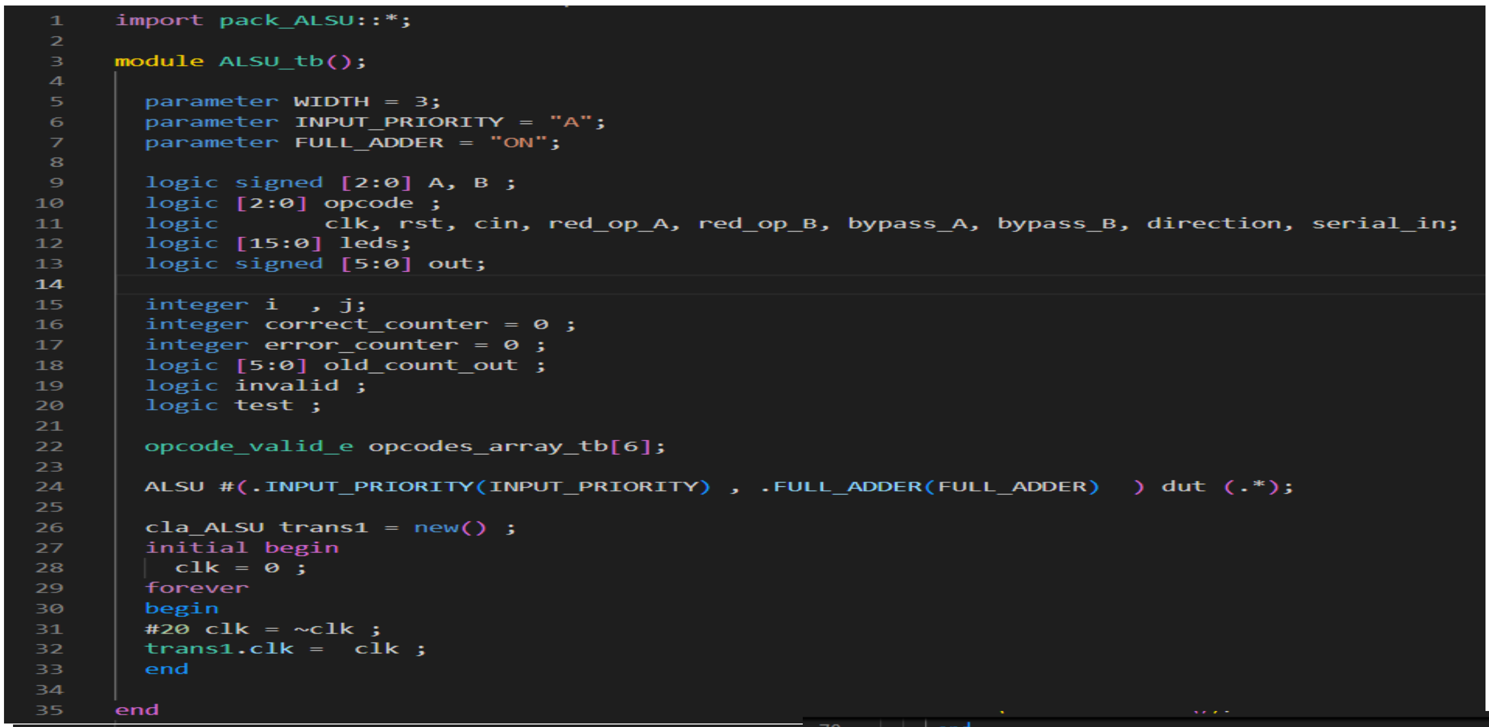


```
  constraint c_fixed_array {
   foreach(opcodes_array[i])
   {
   foreach(opcodes_array[j])
   if(i != j)
   opcodes_array[i] != opcodes_array[j];
   }
   }

covergroup cvr_gp @(posedge clk) ;
 A_cp : coverpoint A {
     bins A_data_0 = {0} ;
     bins A_data_max = {MAXPOS} ;
     bins A_data_min = {MAXNEG} ;
     bins A_data_default = default ;
     bins A_data_walkingones[] = {1, 2, -4} iff (red_op_A && !red_op_B);
 }

 B_cp : coverpoint B {
     bins B_data_0 = {0} ;
     bins B_data_max = {MAXPOS} ;
     bins B_data_min = {MAXNEG} ;
     bins B_data_default = default ;
     bins B_data_walkingones[] = {1, 2, -4} iff (!red_op_A && red_op_B);
 }
```

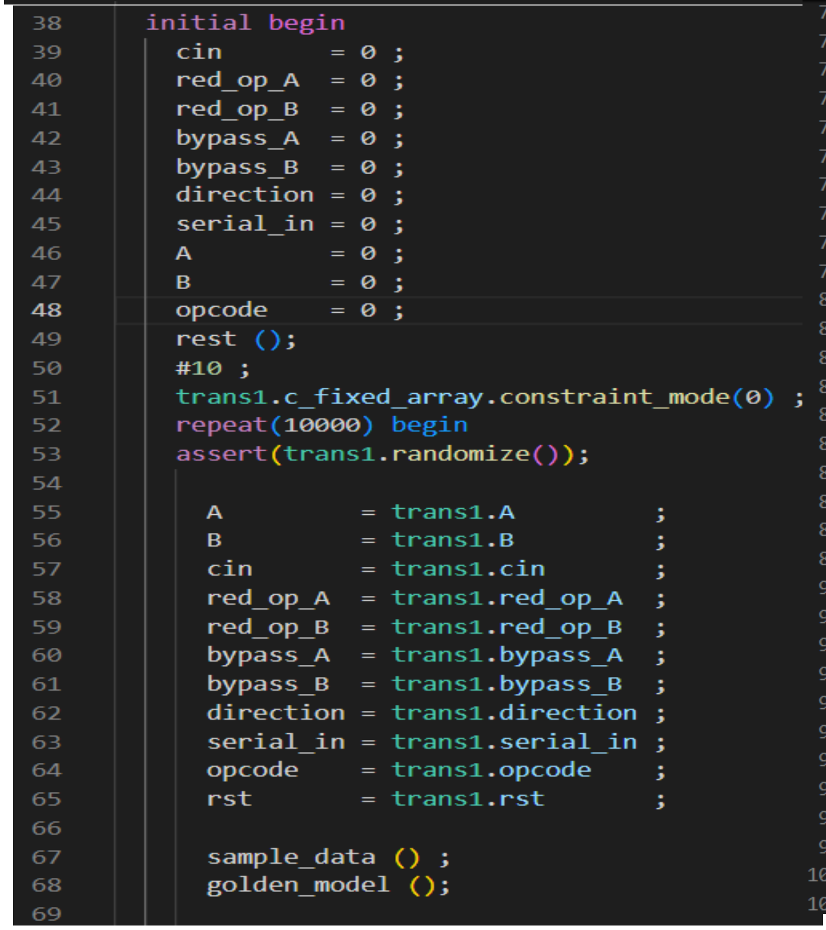


```
    ALU_cp : coverpoint opcode {
      bins Bins_shift[] = {SHIFT , ROTATE} ;
      bins Bins_arith[] = {ADD , MULT} ;
      bins Bins_bitwise[] = {OR , XOR} ;
      illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
      bins Bins_trans = (OR => XOR => ADD => MULT => SHIFT => Rotate);

    }

  endgroup

  function new() ;
    cvr_gp = new() ;
  endfunction

      endclass

endpackage
```

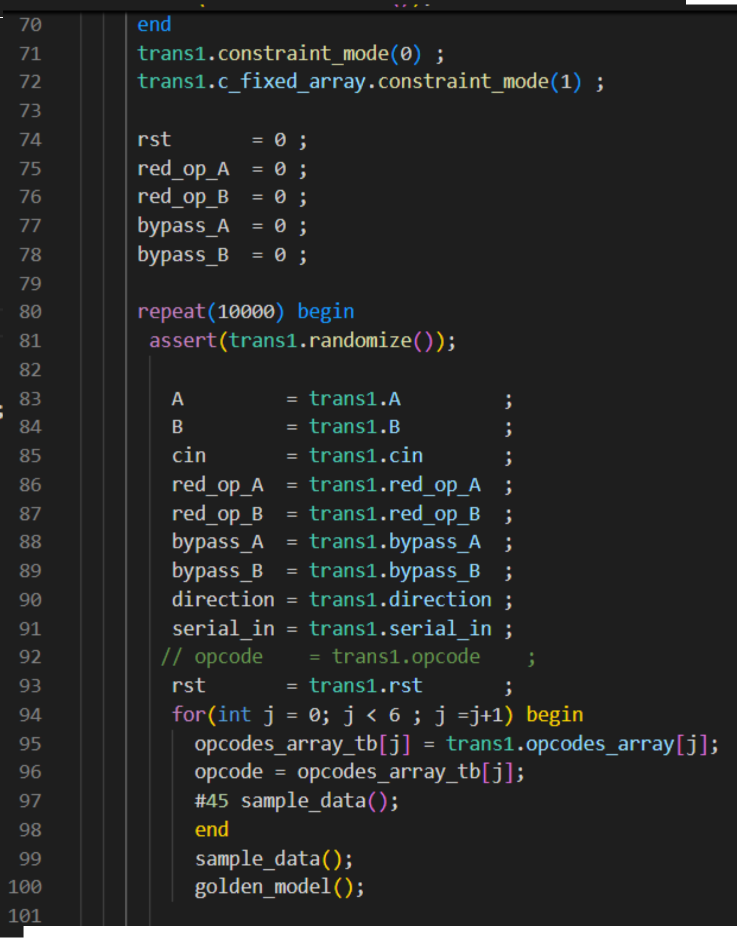
4. ALSU Testbench



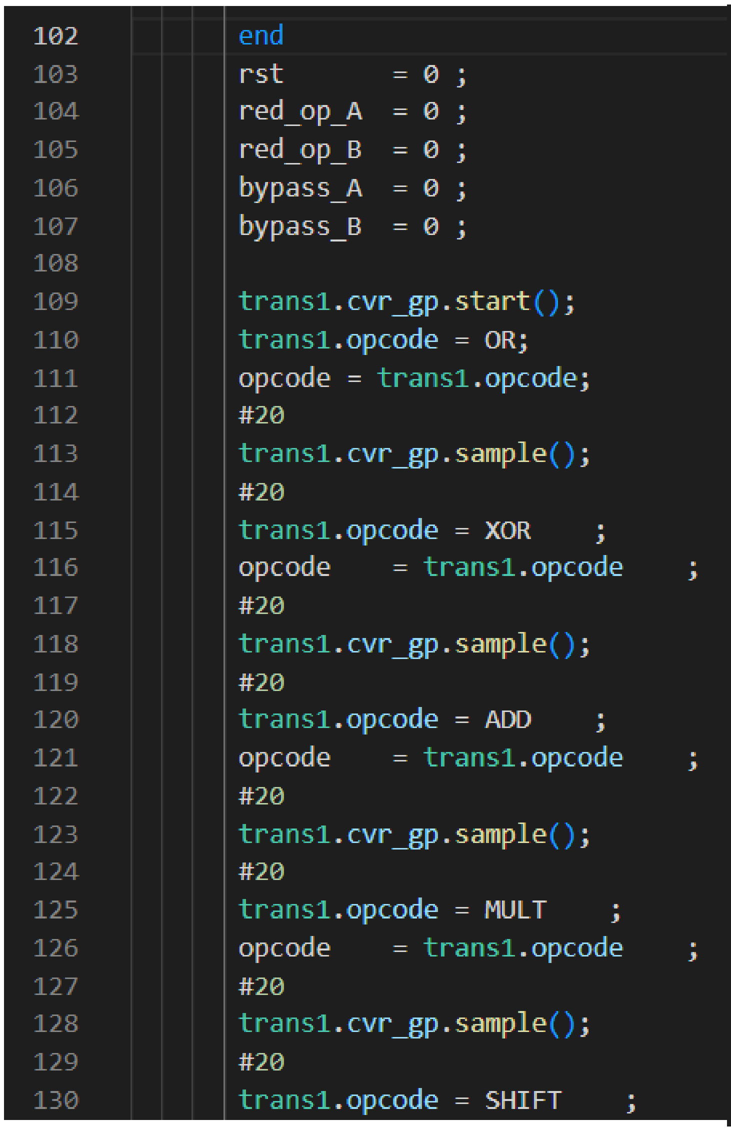
```
import pack_ALSU::*;

module ALSU_tb();

    parameter WIDTH = 3;
    parameter INPUT_PRIORITY = "A";
    parameter FULL_ADDER = "ON";

    logic signed [2:0] A, B ;
    logic [2:0] opcode ;
    logic        clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    logic [15:0] leds;
    logic signed [5:0] out;

    integer i  , j;
    integer correct_counter = 0 ;
    integer error_counter = 0 ;
    logic [5:0] old_count_out ;
    logic invalid ;
    logic test ;

    opcode_valid_e opcodes_array_tb[6];

    ALSU #(.INPUT_PRIORITY(INPUT_PRIORITY) , .FULL_ADDER(FULL_ADDER)  ) dut (.*);

    cla_ALSU trans1 = new() ;
    initial begin
        clk = 0 ;
    forever
    begin
    #20 clk = ~clk ;
    trans1.clk =  clk ;
    end

end
```



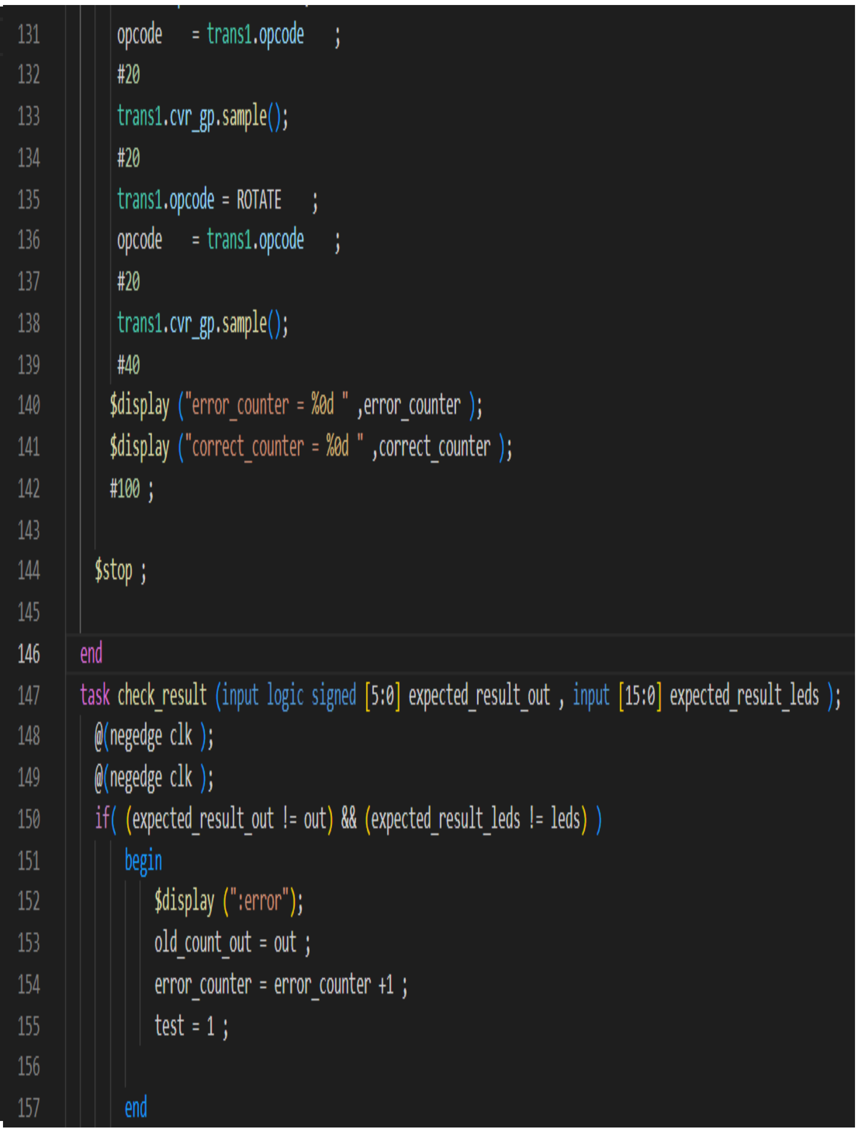
```
    initial begin
        cin       = 0 ;
        red_op_A  = 0 ;
        red_op_B  = 0 ;
        bypass_A  = 0 ;
        bypass_B  = 0 ;
        direction = 0 ;
        serial_in = 0 ;
        A         = 0 ;
        B         = 0 ;
        opcode    = 0 ;
        rest ();
        #10 ;
        trans1.c_fixed_array.constraint_mode(0) ;
        repeat(10000) begin
        assert(trans1.randomize());

            A         = trans1.A         ;
            B         = trans1.B         ;
            cin       = trans1.cin       ;
            red_op_A  = trans1.red_op_A  ;
            red_op_B  = trans1.red_op_B  ;
            bypass_A  = trans1.bypass_A  ;
            bypass_B  = trans1.bypass_B  ;
            direction = trans1.direction ;
            serial_in = trans1.serial_in ;
            opcode    = trans1.opcode    ;
            rst       = trans1.rst       ;

            sample_data () ;
            golden_model ();
```



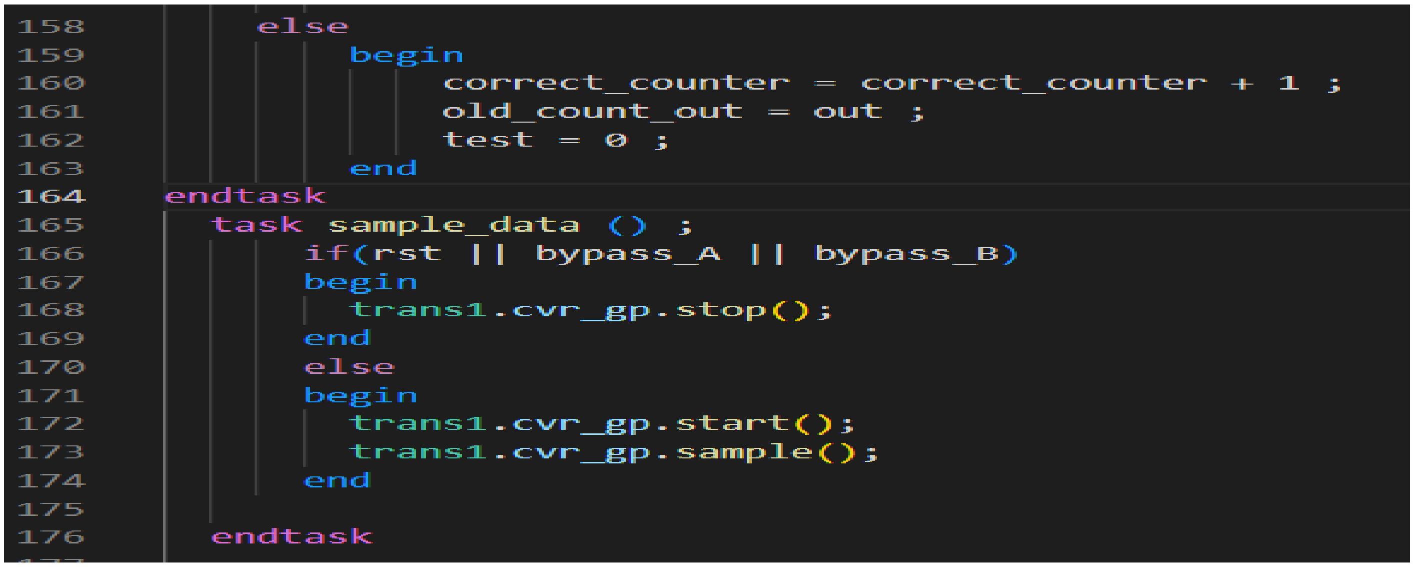
```
        end
        trans1.constraint_mode(0) ;
        trans1.c_fixed_array.constraint_mode(1) ;

        rst      = 0 ;
        red_op_A  = 0 ;
        red_op_B  = 0 ;
        bypass_A  = 0 ;
        bypass_B  = 0 ;

        repeat(10000) begin
         assert(trans1.randomize());

            A         = trans1.A         ;
            B         = trans1.B         ;
            cin       = trans1.cin       ;
            red_op_A  = trans1.red_op_A  ;
            red_op_B  = trans1.red_op_B  ;
            bypass_A  = trans1.bypass_A  ;
            bypass_B  = trans1.bypass_B  ;
            direction = trans1.direction ;
            serial_in = trans1.serial_in ;
          // opcode     = trans1.opcode    ;
            rst       = trans1.rst       ;
            for(int j = 0; j < 6 ; j =j+1) begin
              opcodes_array_tb[j] = trans1.opcodes_array[j];
              opcode = opcodes_array_tb[j];
              #45 sample_data();
              end
              sample_data();
              golden_model();
```



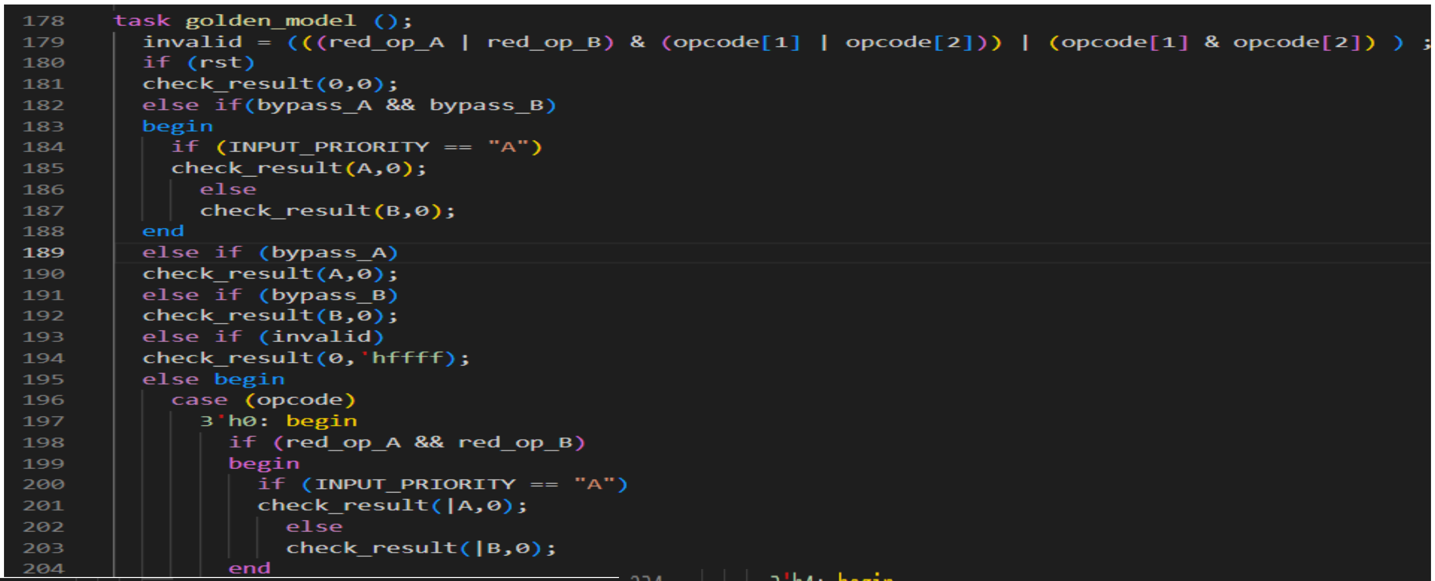
```
        end
        rst         = 0 ;
        red_op_A  = 0 ;
        red_op_B  = 0 ;
        bypass_A  = 0 ;
        bypass_B  = 0 ;

        trans1.cvr_gp.start();
        trans1.opcode = OR;
        opcode = trans1.opcode;
        #20
        trans1.cvr_gp.sample();
        #20
        trans1.opcode = XOR     ;
        opcode      = trans1.opcode     ;
        #20
        trans1.cvr_gp.sample();
        #20
        trans1.opcode = ADD     ;
        opcode      = trans1.opcode     ;
        #20
        trans1.cvr_gp.sample();
        #20
        trans1.opcode = MULT     ;
        opcode      = trans1.opcode     ;
        #20
        trans1.cvr_gp.sample();
        #20
        trans1.opcode = SHIFT     ;
```



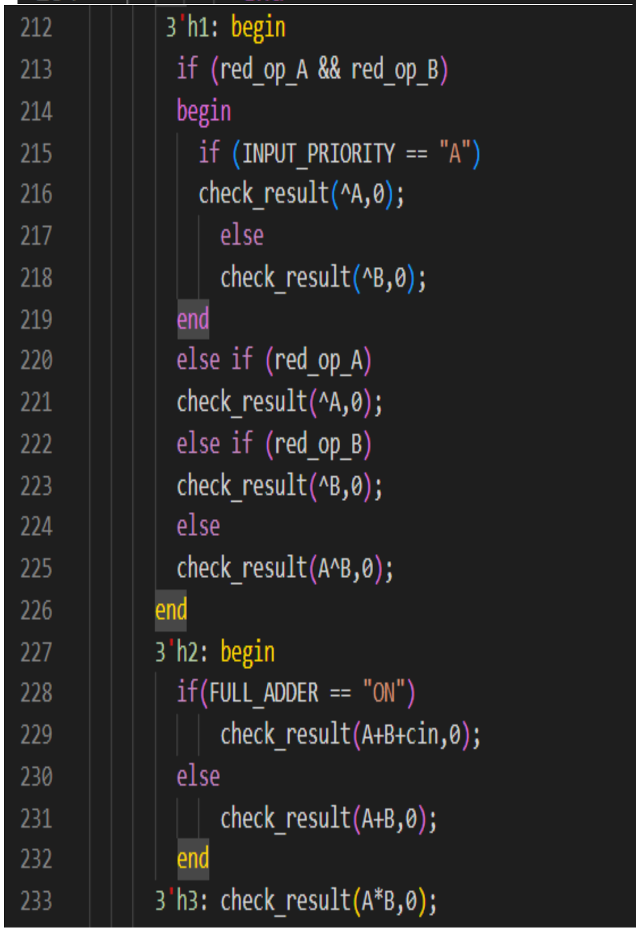
```
        opcode     = trans1.opcode    ;
        #20
        trans1.cvr_gp.sample();
        #20
        trans1.opcode = ROTATE    ;
        opcode     = trans1.opcode    ;
        #20
        trans1.cvr_gp.sample();
        #40
       $display ("error_counter = %0d " ,error_counter );
       $display ("correct_counter = %0d " ,correct_counter );
       #100 ;

    $stop ;

end
task check_result (input logic signed [5:0] expected_result_out , input [15:0] expected_result_leds );
  @(negedge clk );
  @(negedge clk );
  if( (expected_result_out != out) && (expected_result_leds != leds) )
      begin
          $display (":error");
          old_count_out = out ;
          error_counter = error_counter +1 ;
          test = 1 ;

      end
```



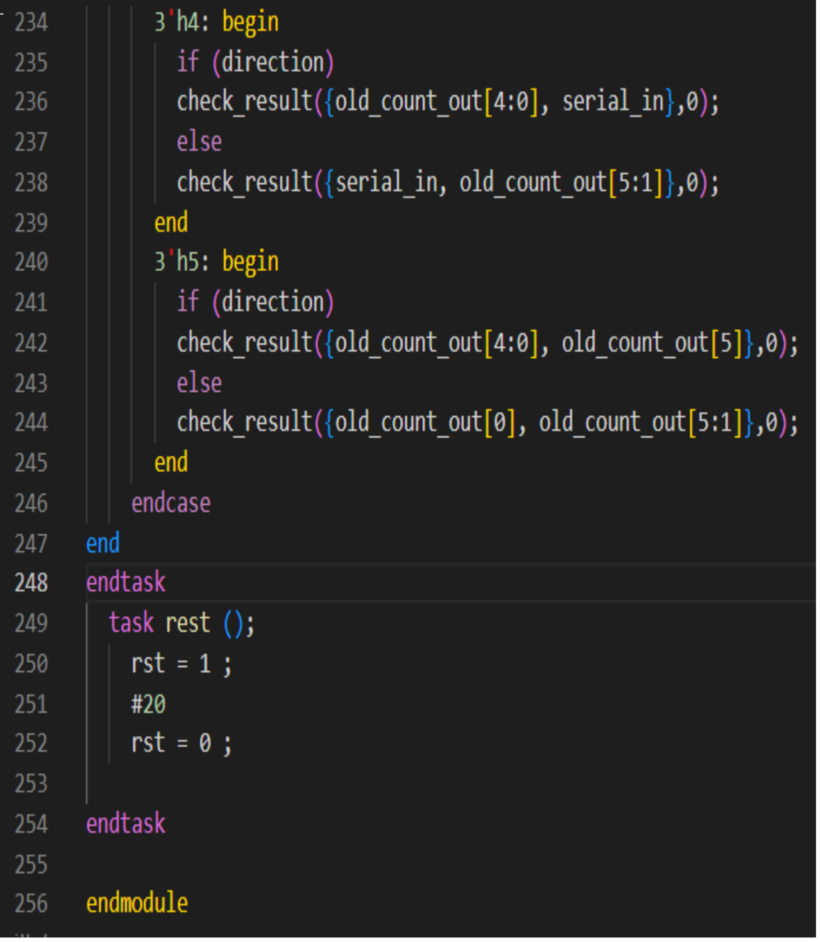
```
    else
        begin
            correct_counter = correct_counter + 1 ;
            old_count_out = out ;
            test = 0 ;
        end
endtask
  task sample_data () ;
      if(rst || bypass_A || bypass_B)
      begin
        trans1.cvr_gp.stop();
      end
      else
      begin
        trans1.cvr_gp.start();
        trans1.cvr_gp.sample();
      end

  endtask
```



```
task golden_model ();
  invalid = (((red_op_A | red_op_B) & (opcode[1] | opcode[2])) | (opcode[1] & opcode[2]) ) ;
  if (rst)
  check_result(0,0);
  else if(bypass_A && bypass_B)
  begin
    if (INPUT_PRIORITY == "A")
    check_result(A,0);
      else
      check_result(B,0);
  end
  else if (bypass_A)
  check_result(A,0);
  else if (bypass_B)
  check_result(B,0);
  else if (invalid)
  check_result(0,'hffff);
  else begin
    case (opcode)
      3'h0: begin
        if (red_op_A && red_op_B)
        begin
          if (INPUT_PRIORITY == "A")
          check_result(|A,0);
            else
            check_result(|B,0);
        end
```

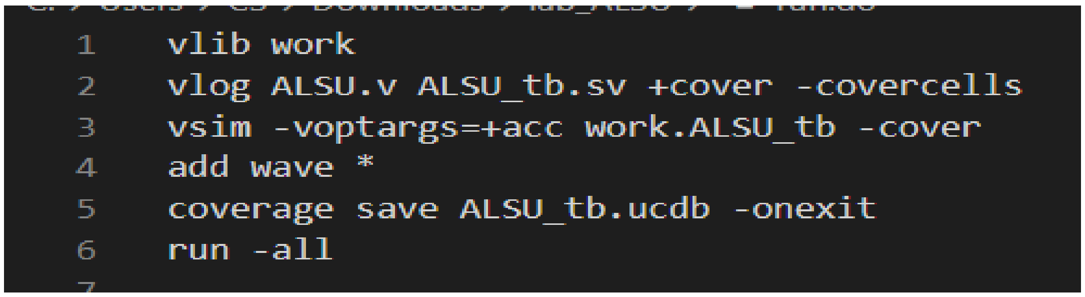


```
        3'h1: begin
        if (red_op_A && red_op_B)
        begin
          if (INPUT_PRIORITY == "A")
          check_result(^A,0);
            else
            check_result(^B,0);
        end
        else if (red_op_A)
        check_result(^A,0);
        else if (red_op_B)
        check_result(^B,0);
        else
        check_result(A^B,0);
      end
      3'h2: begin
        if(FULL_ADDER == "ON")
            check_result(A+B+cin,0);
        else
            check_result(A+B,0);
        end
      3'h3: check_result(A*B,0);
```



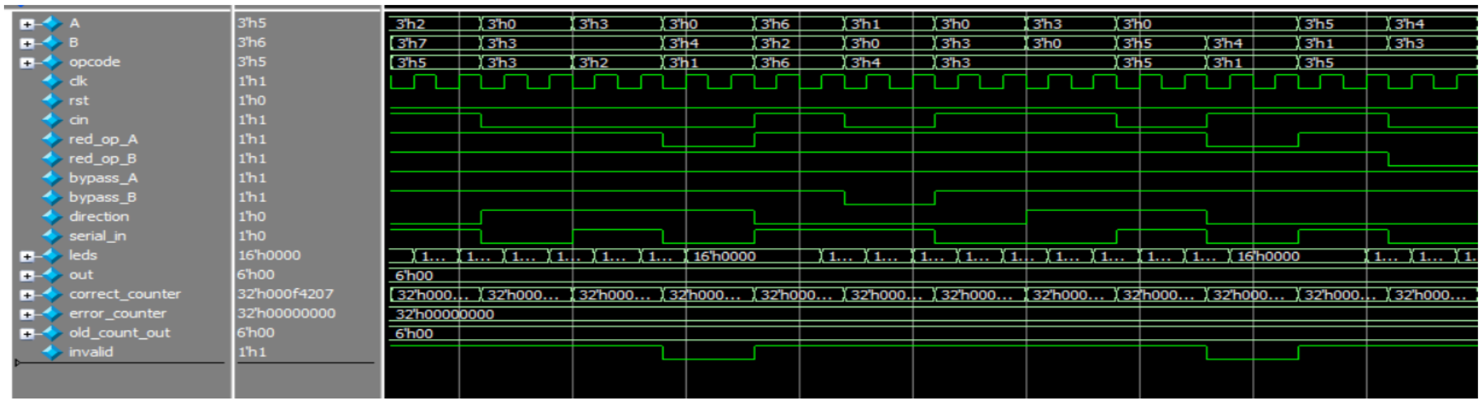
```
      3'h4: begin
        if (direction)
        check_result({old_count_out[4:0], serial_in},0);
        else
        check_result({serial_in, old_count_out[5:1]},0);
      end
      3'h5: begin
        if (direction)
        check_result({old_count_out[4:0], old_count_out[5]},0);
        else
        check_result({old_count_out[0], old_count_out[5:1]},0);
      end
    endcase
end
endtask
  task rest ();
    rst = 1 ;
    #20
    rst = 0 ;

endtask

endmodule
```

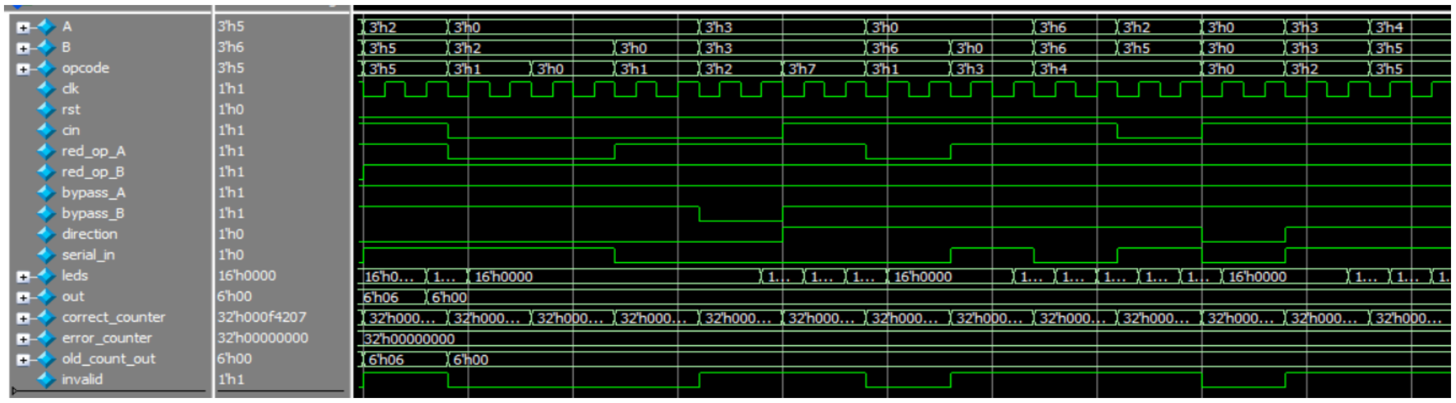
5. Do File



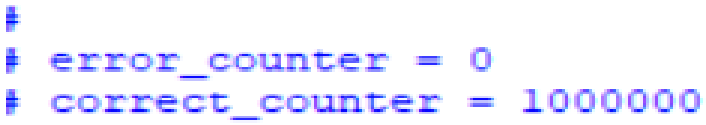
```
vlib work
vlog ALSU.v ALSU_tb.sv +cover -covercells
vsim -voptargs=+acc work.ALSU_tb -cover
add wave *
coverage save ALSU_tb.ucdb -onexit
run -all
```

6. Waveform

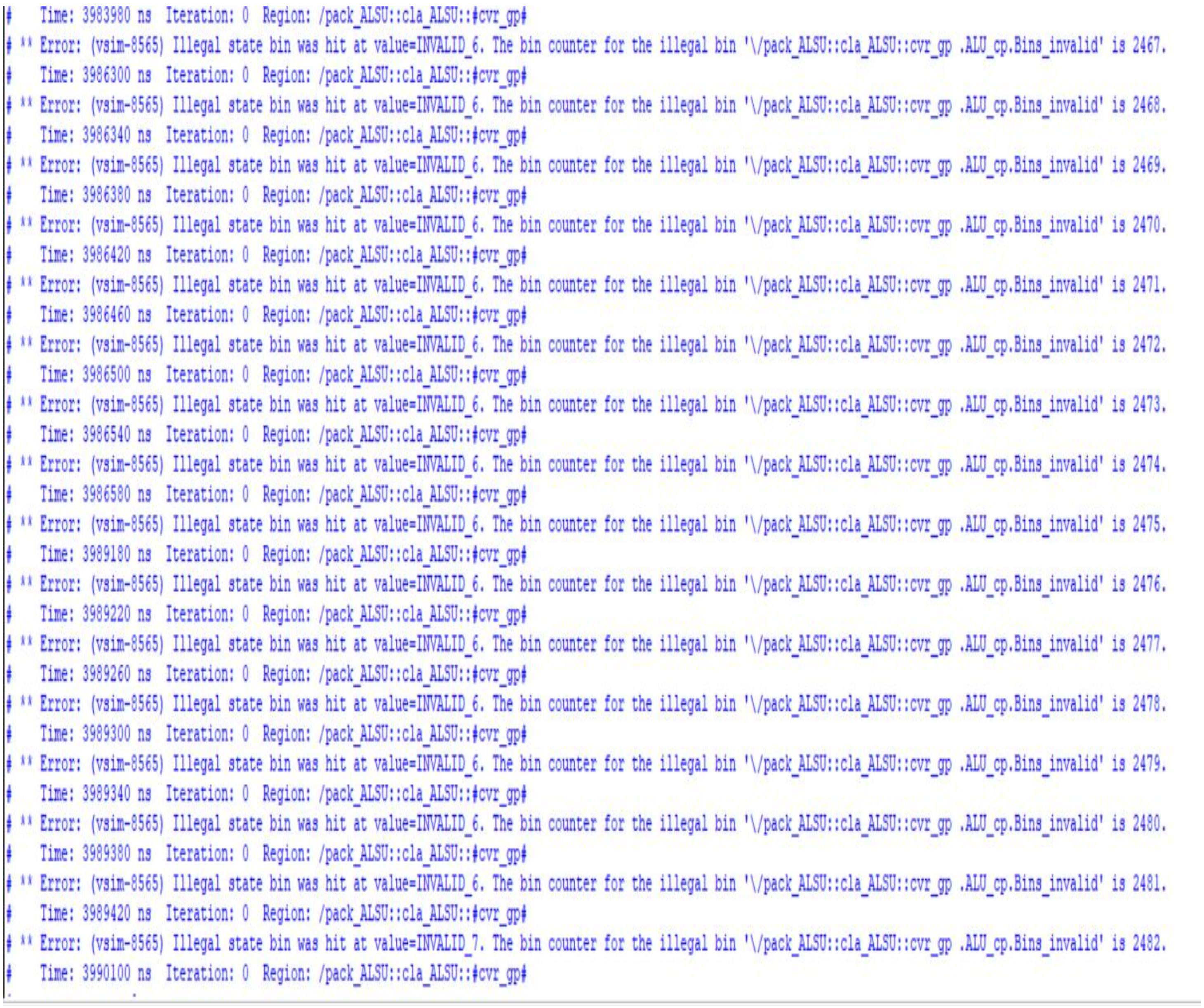




7. Transcript



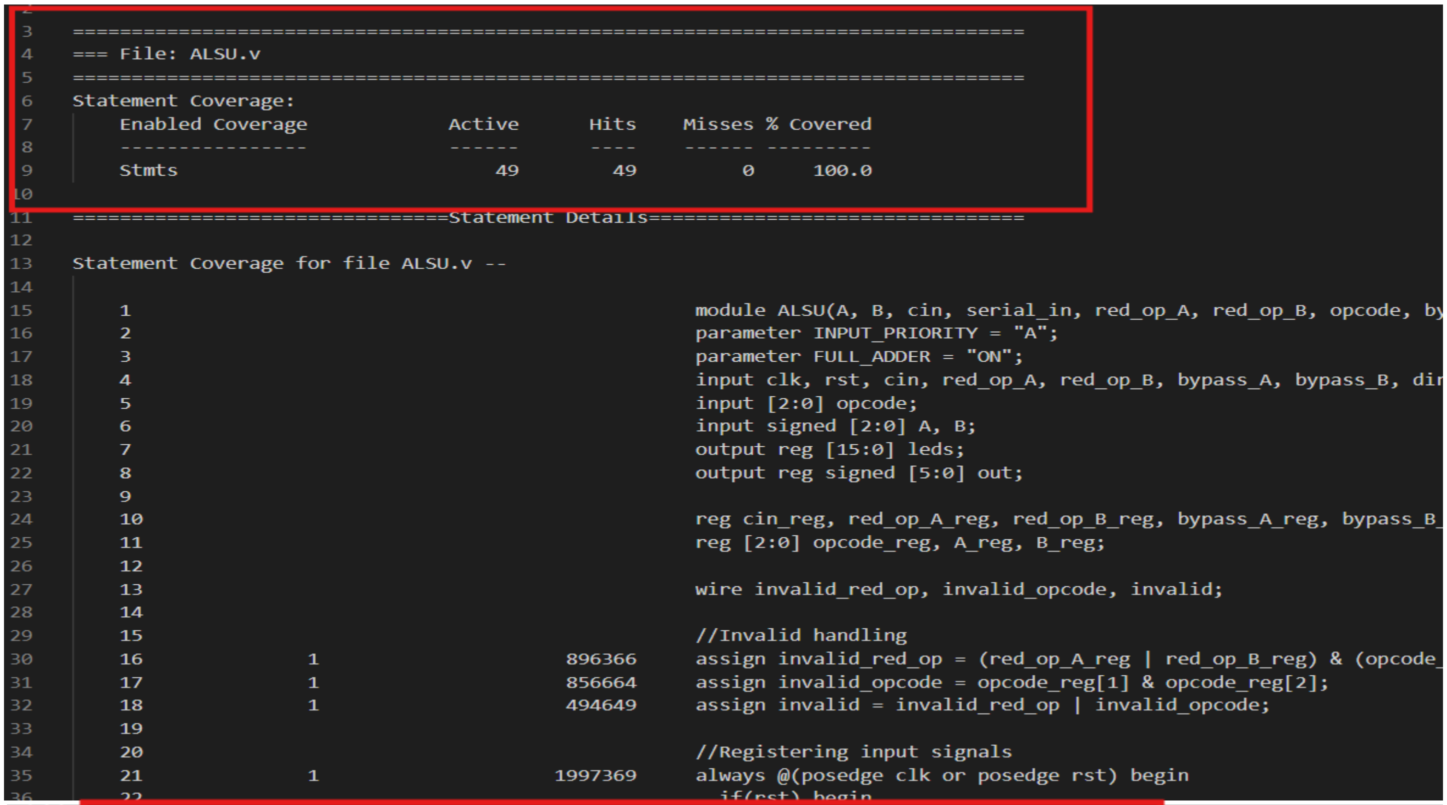
```
#
# error_counter = 0
# correct_counter = 1000000
```



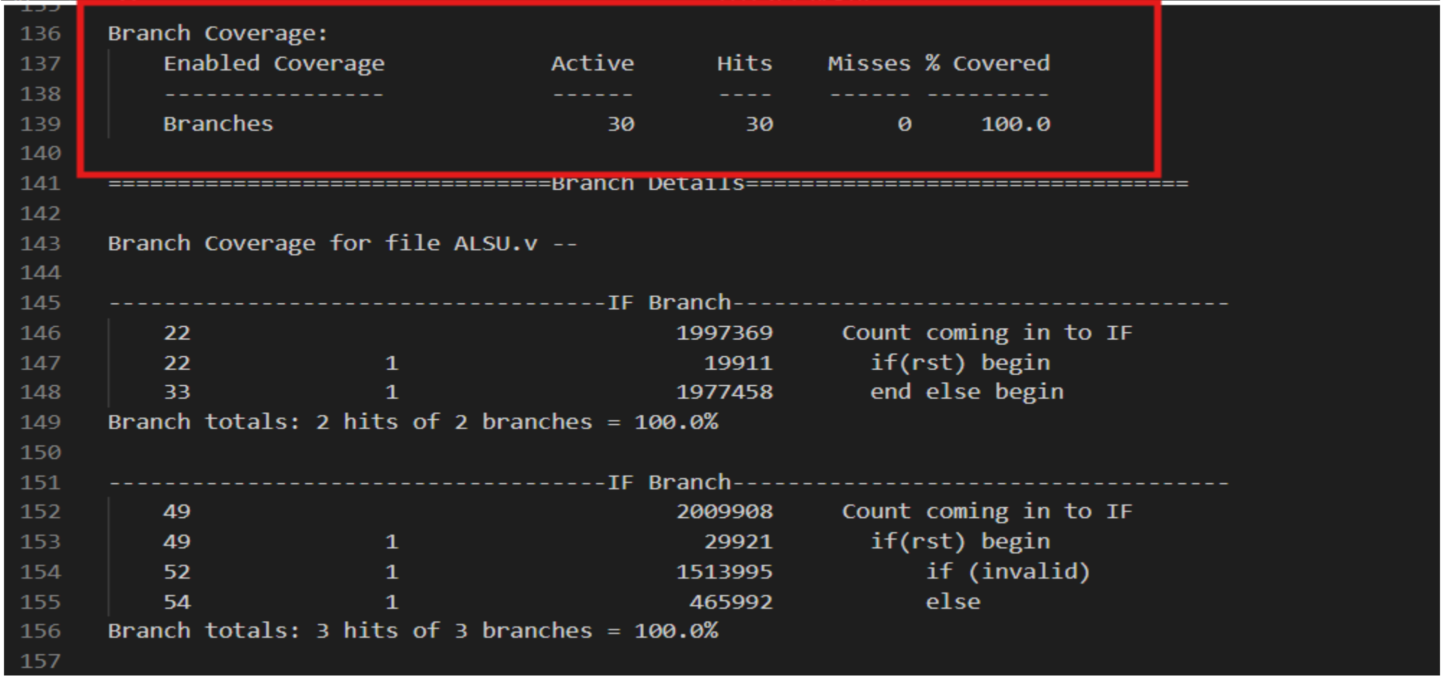
```
#    Time: 3983980 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2467.
#    Time: 3986300 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2468.
#    Time: 3986340 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2469.
#    Time: 3986380 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2470.
#    Time: 3986420 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2471.
#    Time: 3986460 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2472.
#    Time: 3986500 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2473.
#    Time: 3986540 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2474.
#    Time: 3986580 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2475.
#    Time: 3989180 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2476.
#    Time: 3989220 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2477.
#    Time: 3989260 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2478.
#    Time: 3989300 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2479.
#    Time: 3989340 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2480.
#    Time: 3989380 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_6. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2481.
#    Time: 3989420 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
# ** Error: (vsim-8565) Illegal state bin was hit at value=INVALID_7. The bin counter for the illegal bin '\/pack_ALSU::cla_ALSU::cvr_gp .ALU_cp.Bins_invalid' is 2482.
#    Time: 3990100 ns  Iteration: 0  Region: /pack_ALSU::cla_ALSU::#cvr_gp#
```

⇨ All This Errors Resulting from Hitting the Illegal Bins.

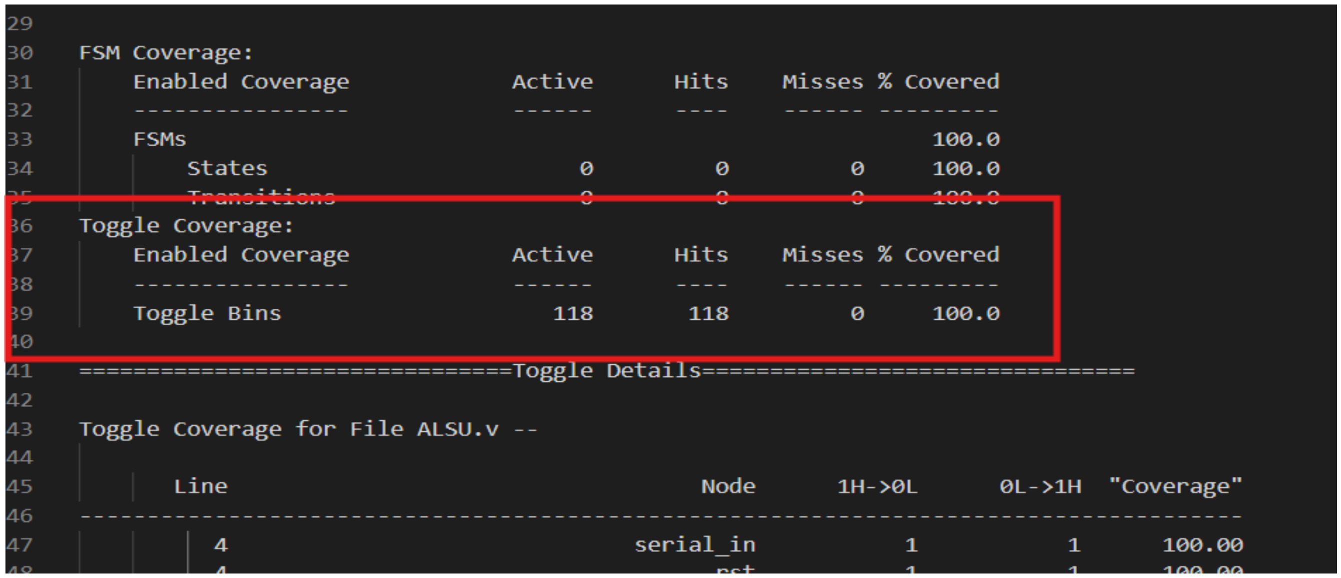
8. Code Coverage



```
=================================================================================
=== File: ALSU.v
=================================================================================
Statement Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Stmts                             49        49         0     100.0

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

Statement Coverage for file ALSU.v --

    1                                                module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, by
    2                                                parameter INPUT_PRIORITY = "A";
    3                                                parameter FULL_ADDER = "ON";
    4                                                input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, dir
    5                                                input [2:0] opcode;
    6                                                input signed [2:0] A, B;
    7                                                output reg [15:0] leds;
    8                                                output reg signed [5:0] out;
    9
    10                                               reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_
    11                                               reg [2:0] opcode_reg, A_reg, B_reg;
    12
    13                                               wire invalid_red_op, invalid_opcode, invalid;
    14
    15                                               //Invalid handling
    16               1                     896366    assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_
    17               1                     856664    assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
    18               1                     494649    assign invalid = invalid_red_op | invalid_opcode;
    19
    20                                               //Registering input signals
    21               1                    1997369    always @(posedge clk or posedge rst) begin
    22                                                 if(rst) begin
```

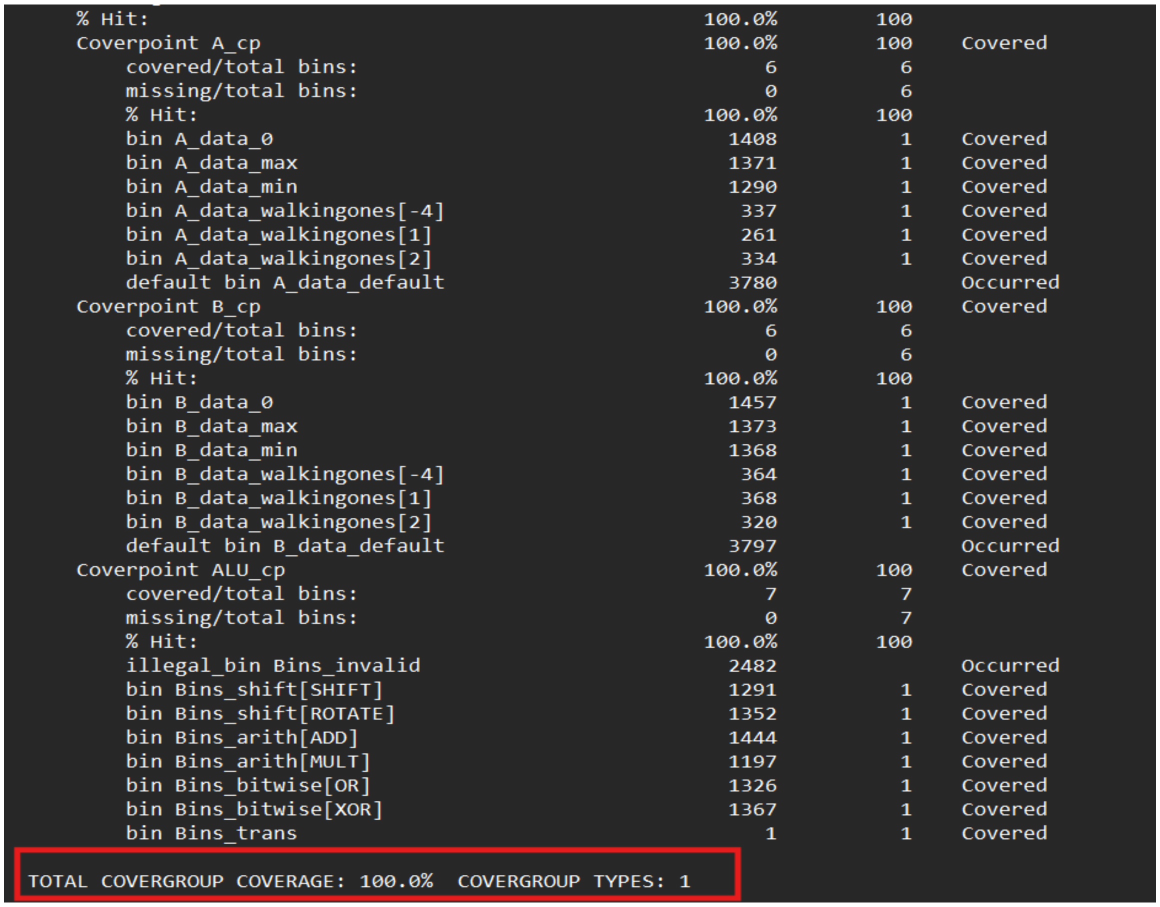


```
Branch Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Branches                          30        30         0     100.0

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

Branch Coverage for file ALSU.v --

------------------------------------IF Branch------------------------------------
    22                                   1997369     Count coming in to IF
    22                1                    19911       if(rst) begin
    33                1                  1977458       end else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    49                                   2009908     Count coming in to IF
    49                1                    29921       if(rst) begin
    52                1                  1513995           if (invalid)
    54                1                   465992           else
Branch totals: 3 hits of 3 branches = 100.0%
```



```

FSM Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    FSMs                                                       100.0
        States                         0         0         0   100.0
        Transitions                    0         0         0   100.0
Toggle Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Toggle Bins                      118       118         0   100.0

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

Toggle Coverage for File ALSU.v --

       Line                                Node      1H->0L      0L->1H  "Coverage"
---------------------------------------------------------------------------------
          4                           serial_in           1           1      100.00
```

9. function Coverage



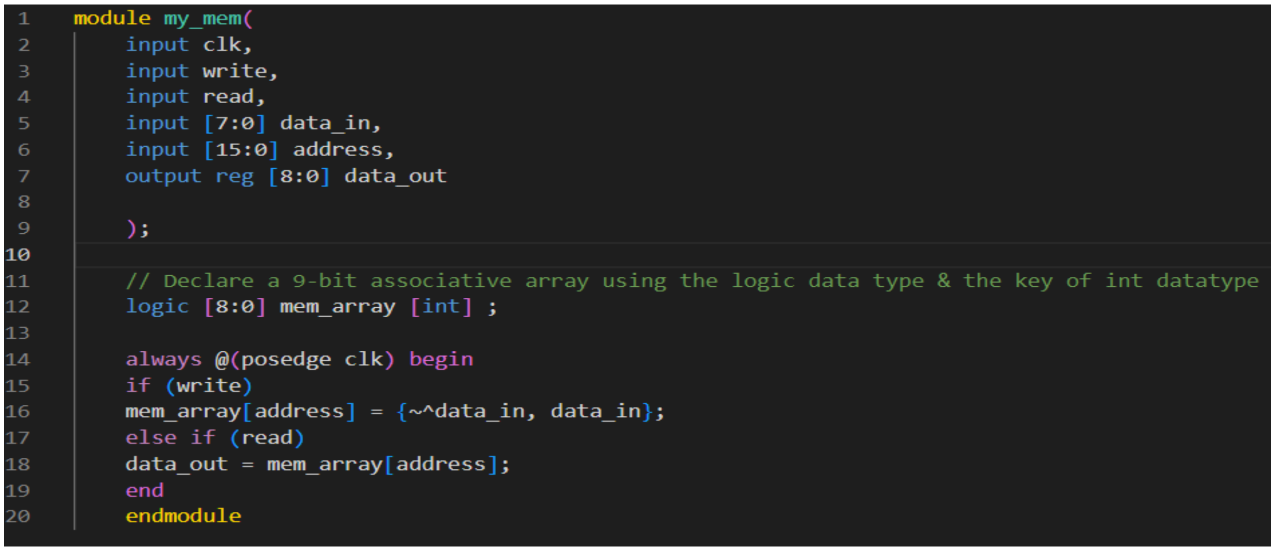
```
     % Hit:                                       100.0%        100
     Coverpoint A_cp                              100.0%        100    Covered
         covered/total bins:                           6          6
         missing/total bins:                           0          6
         % Hit:                                   100.0%        100
         bin A_data_0                               1408          1    Covered
         bin A_data_max                             1371          1    Covered
         bin A_data_min                             1290          1    Covered
         bin A_data_walkingones[-4]                  337          1    Covered
         bin A_data_walkingones[1]                   261          1    Covered
         bin A_data_walkingones[2]                   334          1    Covered
         default bin A_data_default                 3780               Occurred
     Coverpoint B_cp                              100.0%        100    Covered
         covered/total bins:                           6          6
         missing/total bins:                           0          6
         % Hit:                                   100.0%        100
         bin B_data_0                               1457          1    Covered
         bin B_data_max                             1373          1    Covered
         bin B_data_min                             1368          1    Covered
         bin B_data_walkingones[-4]                  364          1    Covered
         bin B_data_walkingones[1]                   368          1    Covered
         bin B_data_walkingones[2]                   320          1    Covered
         default bin B_data_default                 3797               Occurred
     Coverpoint ALU_cp                            100.0%        100    Covered
         covered/total bins:                           7          7
         missing/total bins:                           0          7
         % Hit:                                   100.0%        100
         illegal_bin Bins_invalid                   2482               Occurred
         bin Bins_shift[SHIFT]                      1291          1    Covered
         bin Bins_shift[ROTATE]                     1352          1    Covered
         bin Bins_arith[ADD]                        1444          1    Covered
         bin Bins_arith[MULT]                       1197          1    Covered
         bin Bins_bitwise[OR]                       1326          1    Covered
         bin Bins_bitwise[XOR]                      1367          1    Covered
         bin Bins_trans                                1          1    Covered

 TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1
```

Question3

➢ memory

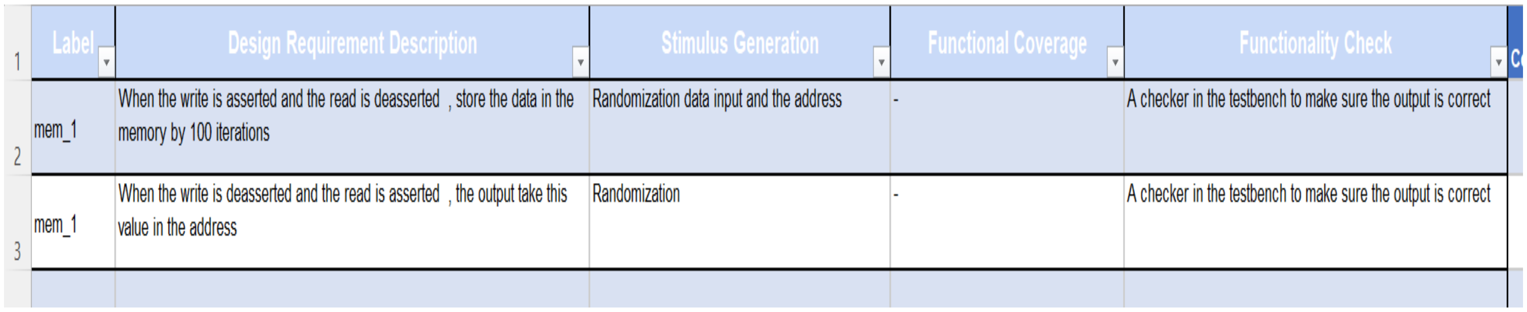
The design to be tested is a synchronous single-port 8-bit x64K (512kBit) RAM. The RAM will read on the positive edge of the clock when input read =1 and write on the positive edge of the clock when input write = 1. Write enable signal has a higher priority than the read enable signal and both write and read data from the RAM is not allowed at the same time. Even parity will be calculated on data written to the RAM and placed in the 9th bit of the memory. The partially completed memory model is below (add the memory declaration)

1. Code Design



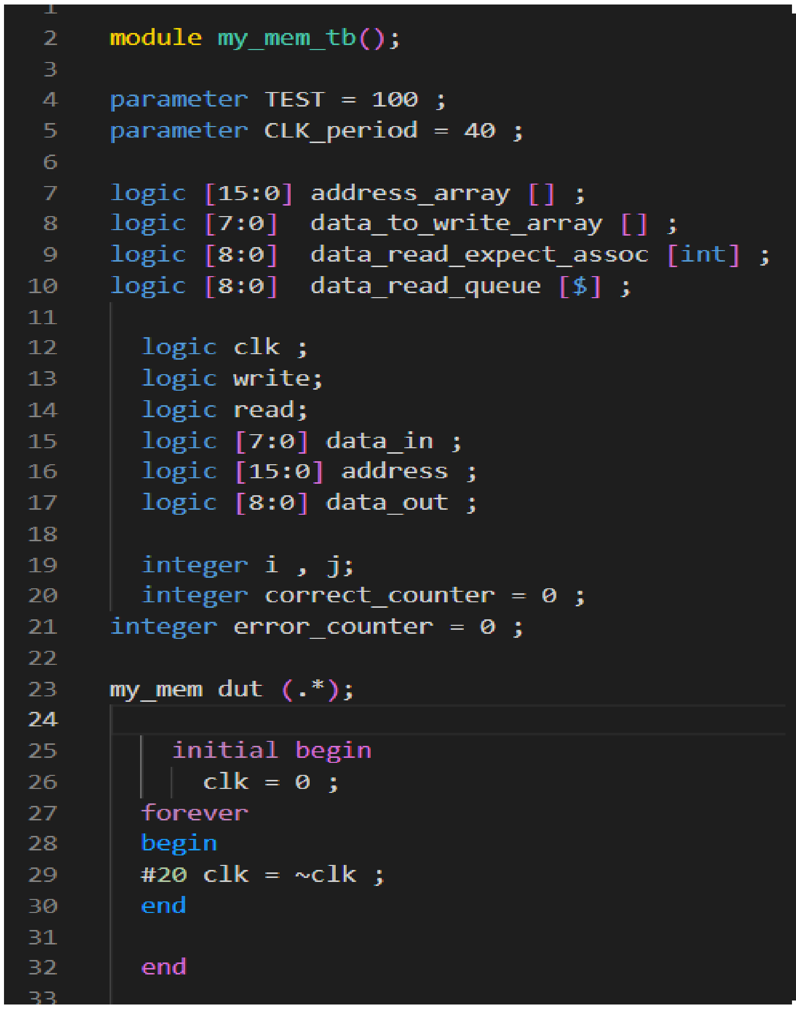
```
module my_mem(
    input clk,
    input write,
    input read,
    input [7:0] data_in,
    input [15:0] address,
    output reg [8:0] data_out

    );

    // Declare a 9-bit associative array using the logic data type & the key of int datatype
    logic [8:0] mem_array [int] ;

    always @(posedge clk) begin
    if (write)
    mem_array[address] = {~^data_in, data_in};
    else if (read)
    data_out = mem_array[address];
    end
    endmodule
```

2. Verification plan

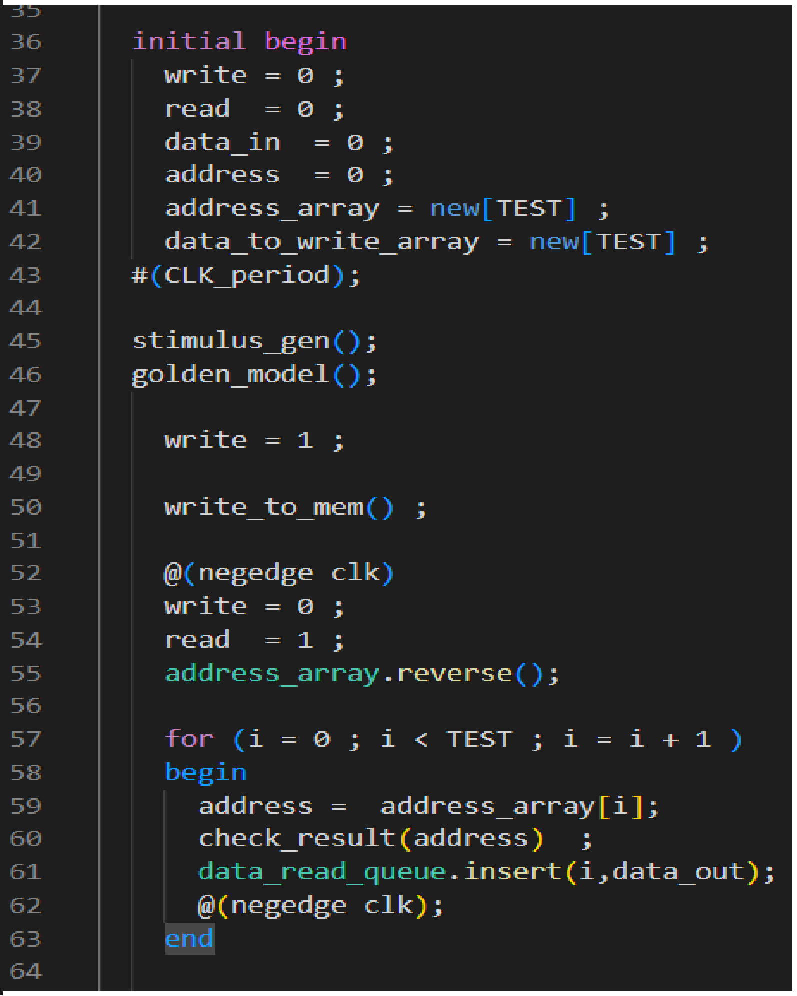


Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check	C
mem_1	When the write is asserted and the read is deasserted , store the data in the memory by 100 iterations	Randomization data input and the address	-	A checker in the testbench to make sure the output is correct	
mem_1	When the write is deasserted and the read is asserted , the output take this value in the address	Randomization	-	A checker in the testbench to make sure the output is correct	

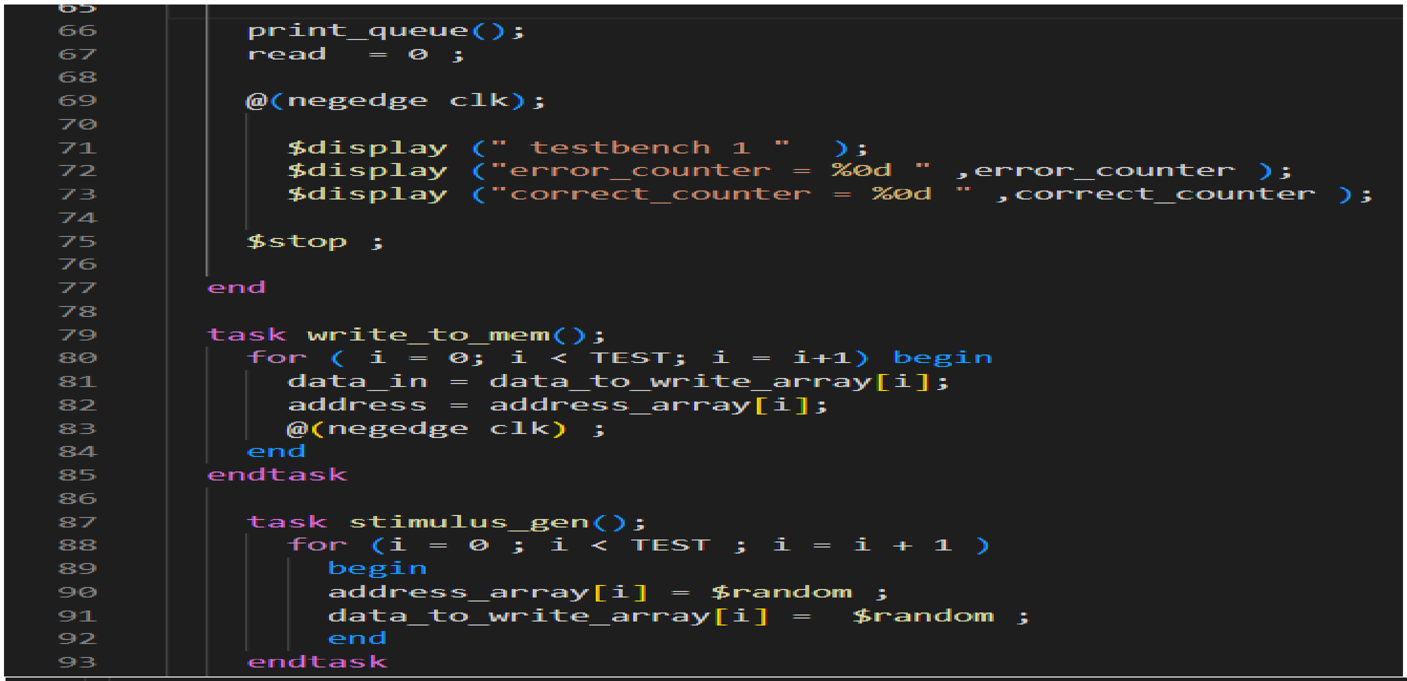
3. memory Testbench



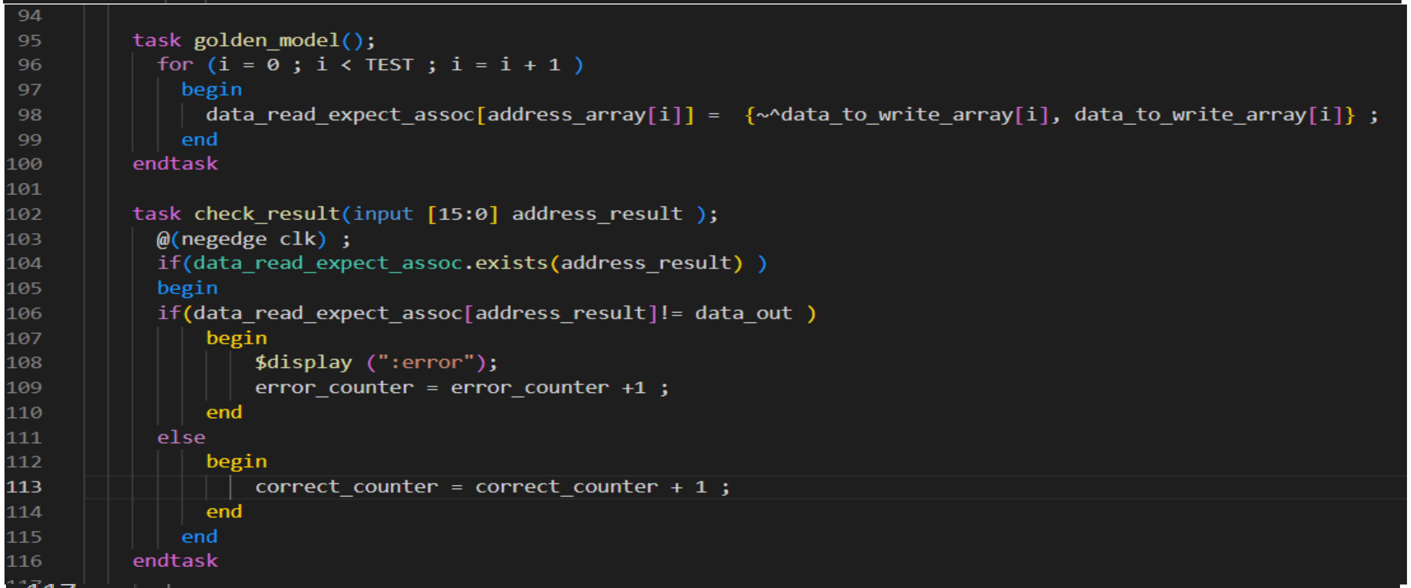
```
module my_mem_tb();

parameter TEST = 100 ;
parameter CLK_period = 40 ;

logic [15:0] address_array [] ;
logic [7:0]  data_to_write_array [] ;
logic [8:0]  data_read_expect_assoc [int] ;
logic [8:0]  data_read_queue [$] ;

  logic clk ;
  logic write;
  logic read;
  logic [7:0] data_in ;
  logic [15:0] address ;
  logic [8:0] data_out ;

  integer i , j;
  integer correct_counter = 0 ;
integer error_counter = 0 ;

my_mem dut (.*);


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

  end
```



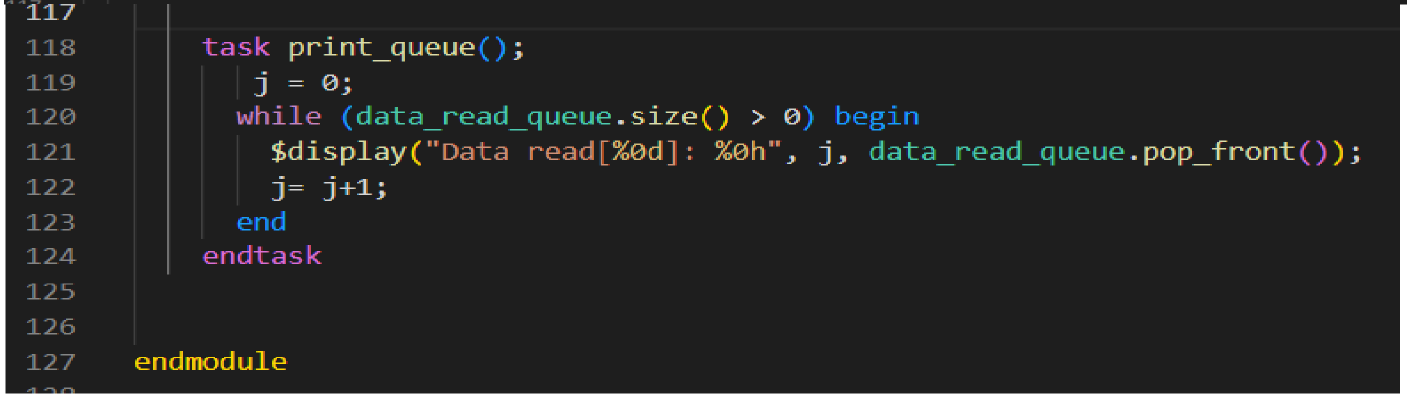
```
initial begin
  write = 0 ;
  read  = 0 ;
  data_in   = 0 ;
  address   = 0 ;
  address_array = new[TEST] ;
  data_to_write_array = new[TEST] ;
#(CLK_period);

stimulus_gen();
golden_model();

  write = 1 ;

  write_to_mem() ;

  @(negedge clk)
  write = 0 ;
  read  = 1 ;
  address_array.reverse();

  for (i = 0 ; i < TEST ; i = i + 1 )
  begin
    address =  address_array[i];
    check_result(address)   ;
    data_read_queue.insert(i,data_out);
    @(negedge clk);
  end
```



```
    print_queue();
    read  = 0 ;

    @(negedge clk);

      $display (" testbench 1 "  );
      $display ("error_counter = %0d " ,error_counter );
      $display ("correct_counter = %0d " ,correct_counter );

    $stop ;

  end

  task write_to_mem();
    for ( i = 0; i < TEST; i = i+1) begin
      data_in = data_to_write_array[i];
      address = address_array[i];
      @(negedge clk) ;
    end
  endtask

    task stimulus_gen();
      for (i = 0 ; i < TEST ; i = i + 1 )
        begin
        address_array[i]  = $random ;
        data_to_write_array[i] =  $random ;
        end
    endtask
```

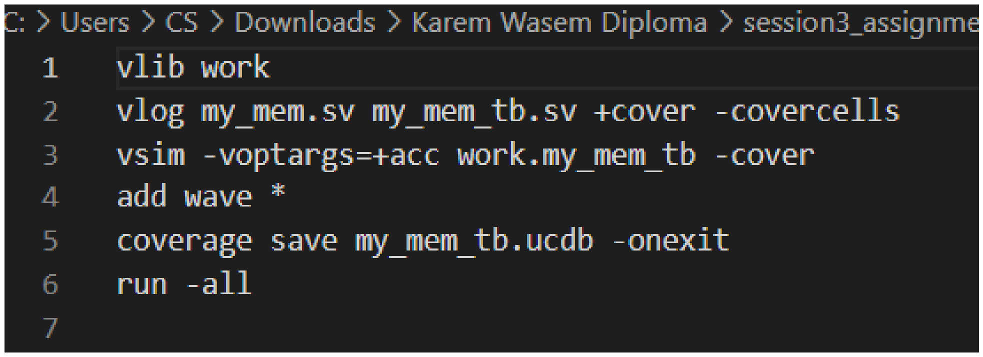


```
  task golden_model();
    for (i = 0 ; i < TEST ; i = i + 1 )
      begin
        data_read_expect_assoc[address_array[i]] =  {~^data_to_write_array[i], data_to_write_array[i]} ;
      end
  endtask

  task check_result(input [15:0] address_result );
    @(negedge clk) ;
    if(data_read_expect_assoc.exists(address_result) )
    begin
    if(data_read_expect_assoc[address_result]!= data_out )
        begin
          $display (":error");
          error_counter = error_counter +1 ;
        end
    else
        begin
          correct_counter = correct_counter + 1 ;
        end
      end
  endtask
```



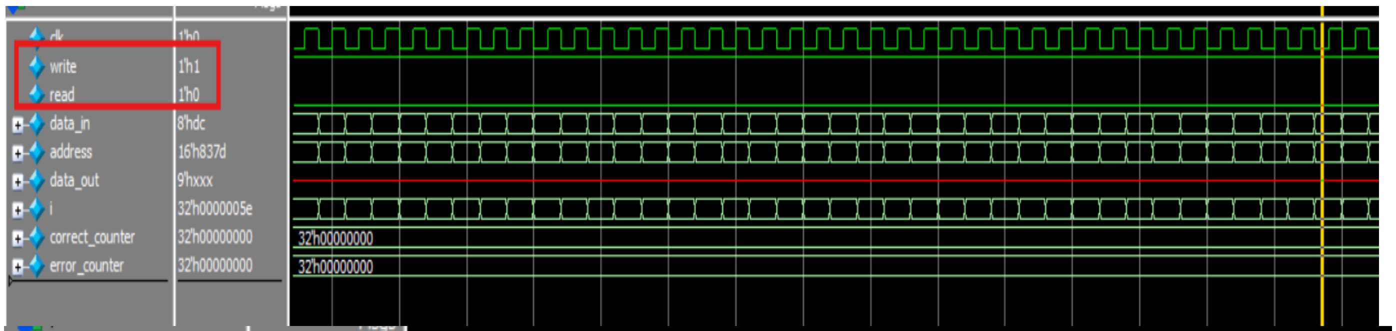
```
  task print_queue();
    j = 0;
    while (data_read_queue.size() > 0) begin
      $display("Data read[%0d]: %0h", j, data_read_queue.pop_front());
      j= j+1;
    end
  endtask


endmodule
```

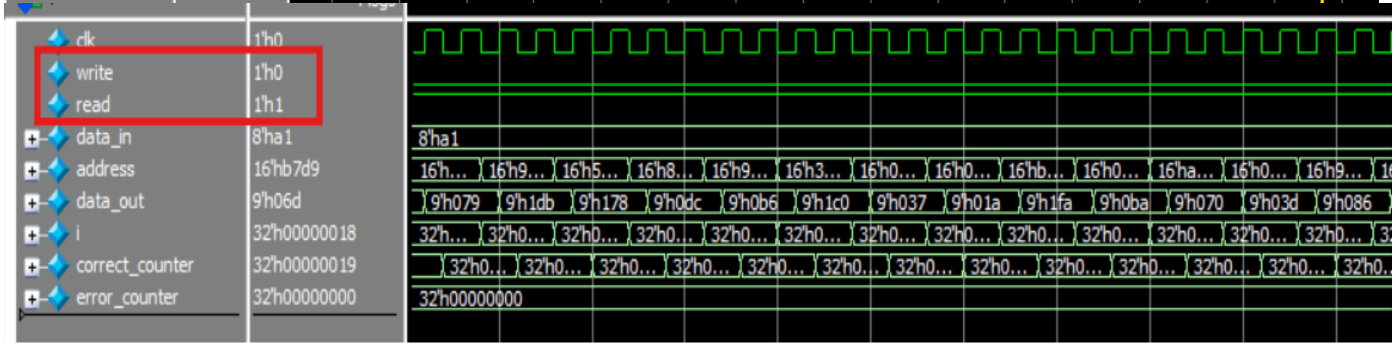
4. Do File



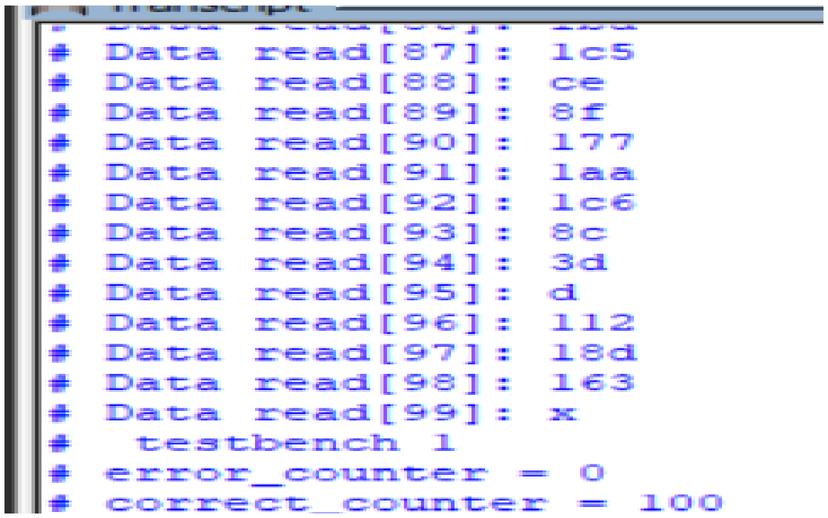
```
C: > Users > CS > Downloads > Karem Wasem Diploma > session3_assignme
vlib work
vlog my_mem.sv my_mem_tb.sv +cover -covercells
vsim -voptargs=+acc work.my_mem_tb -cover
add wave *
coverage save my_mem_tb.ucdb -onexit
run -all
```

5. Waveform



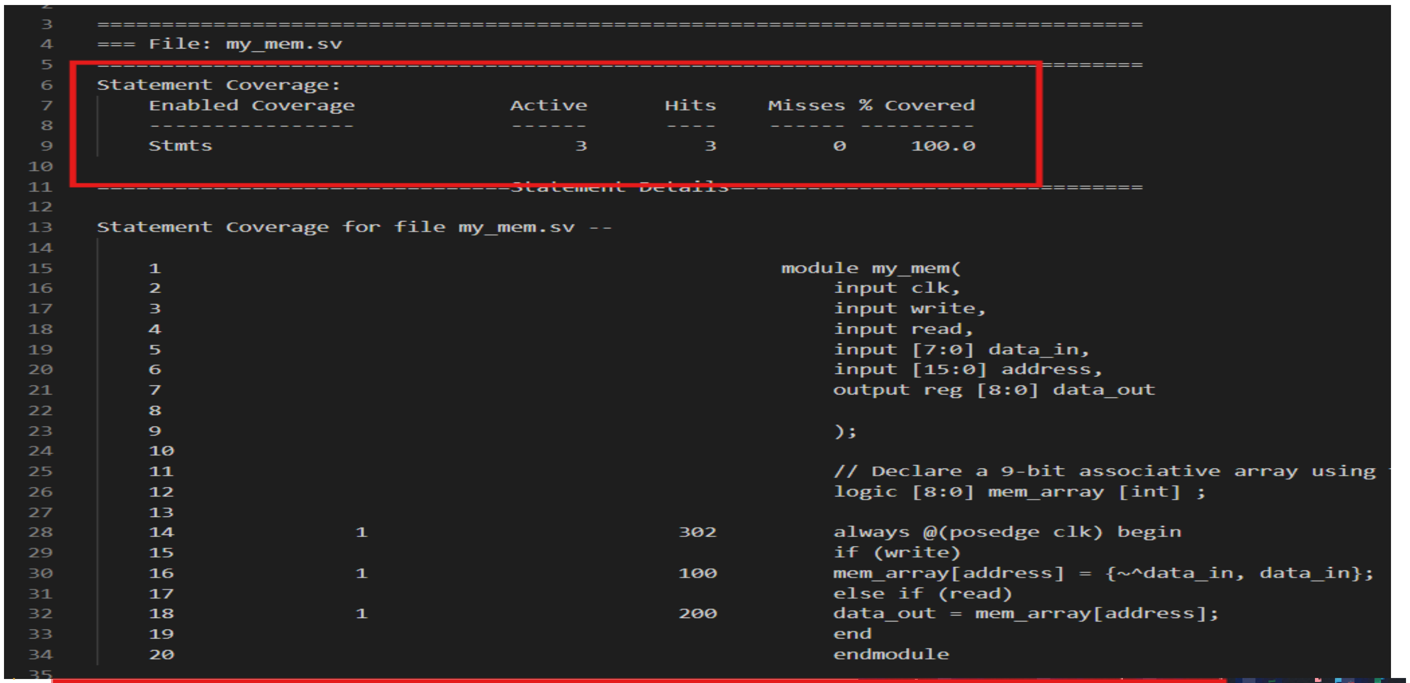


6. Transcript

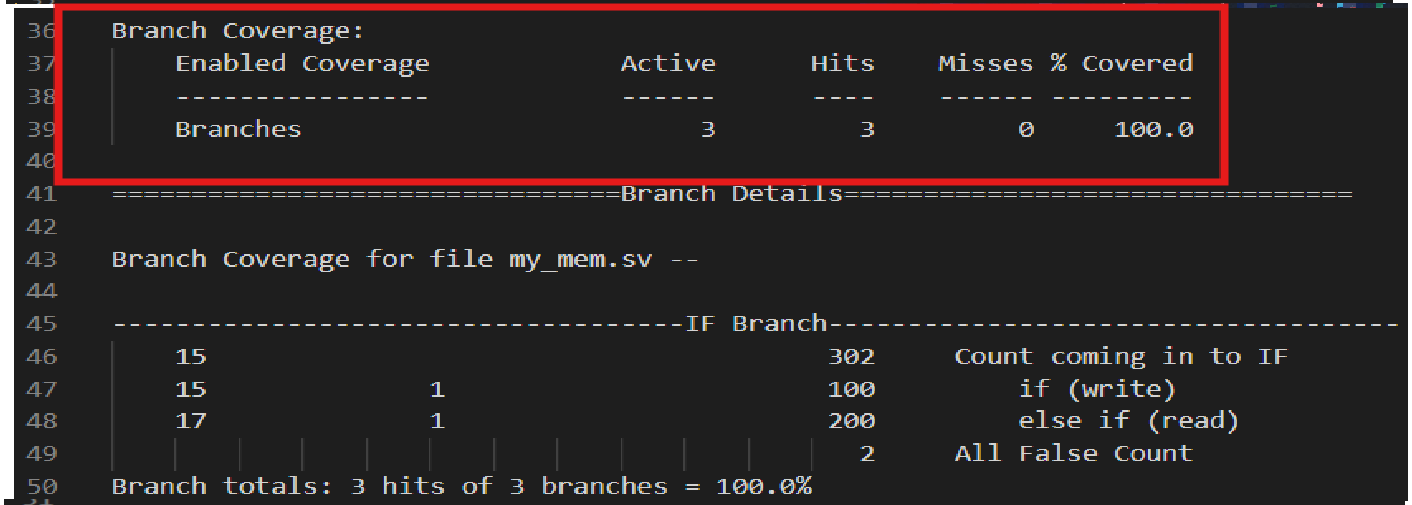


```
# Data read[87]: 1c5
# Data read[88]: ce
# Data read[89]: 8f
# Data read[90]: 177
# Data read[91]: 1aa
# Data read[92]: 1c6
# Data read[93]: 8c
# Data read[94]: 3d
# Data read[95]: d
# Data read[96]: 112
# Data read[97]: 18d
# Data read[98]: 163
# Data read[99]: x
#  testbench 1
# error_counter = 0
# correct_counter = 100
```

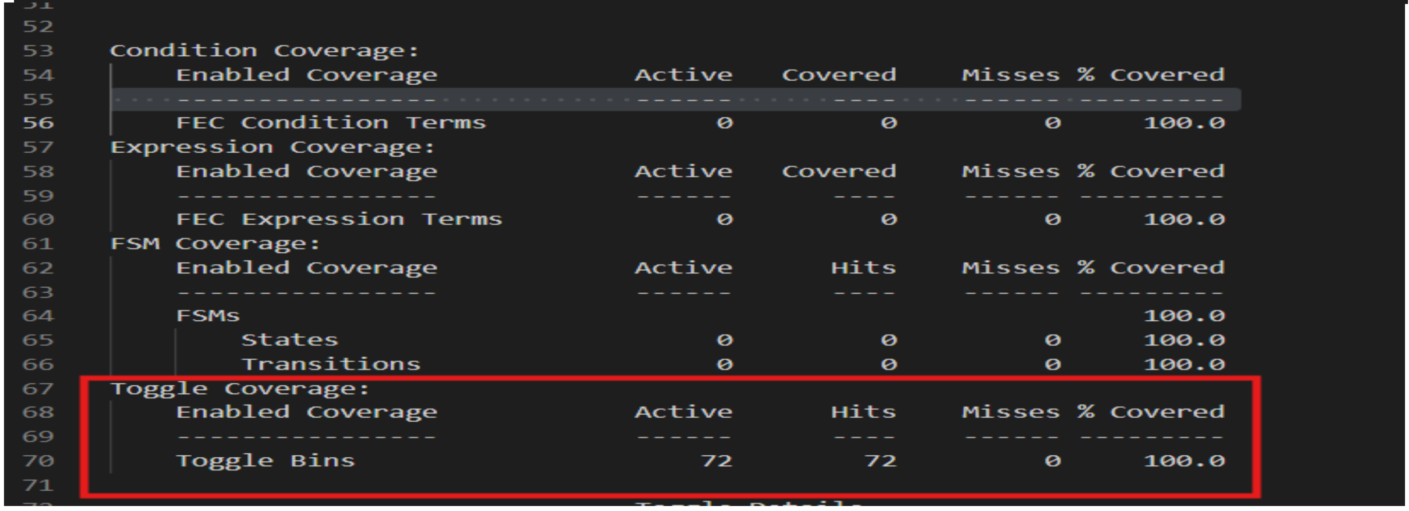
7. Code Coverage



```
===============================================================================
=== File: my_mem.sv
===============================================================================
Statement Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Stmts                              3         3         0     100.0

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

Statement Coverage for file my_mem.sv --

    1                                                module my_mem(
    2                                                    input clk,
    3                                                    input write,
    4                                                    input read,
    5                                                    input [7:0] data_in,
    6                                                    input [15:0] address,
    7                                                    output reg [8:0] data_out
    8
    9                                                    );
    10
    11                                                   // Declare a 9-bit associative array using
    12                                                   logic [8:0] mem_array [int] ;
    13
    14               1                   302             always @(posedge clk) begin
    15                                                   if (write)
    16               1                   100             mem_array[address] = {~^data_in, data_in};
    17                                                   else if (read)
    18               1                   200             data_out = mem_array[address];
    19                                                   end
    20                                                   endmodule
```



```
Branch Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Branches                           3         3         0     100.0

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

Branch Coverage for file my_mem.sv --

------------------------------------IF Branch------------------------------------
    15                                   302     Count coming in to IF
    15           1                       100         if (write)
    17           1                       200         else if (read)
                                           2     All False Count
Branch totals: 3 hits of 3 branches = 100.0%
```



```
Condition Coverage:
    Enabled Coverage              Active   Covered    Misses % Covered
    ----------------              ------      ----    ------ ---------
    FEC Condition Terms                0         0         0     100.0
Expression Coverage:
    Enabled Coverage              Active   Covered    Misses % Covered
    ----------------              ------      ----    ------ ---------
    FEC Expression Terms               0         0         0     100.0
FSM Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    FSMs                                                         100.0
        States                         0         0         0     100.0
        Transitions                    0         0         0     100.0
Toggle Coverage:
    Enabled Coverage              Active      Hits    Misses % Covered
    ----------------              ------      ----    ------ ---------
    Toggle Bins                       72        72         0     100.0
```