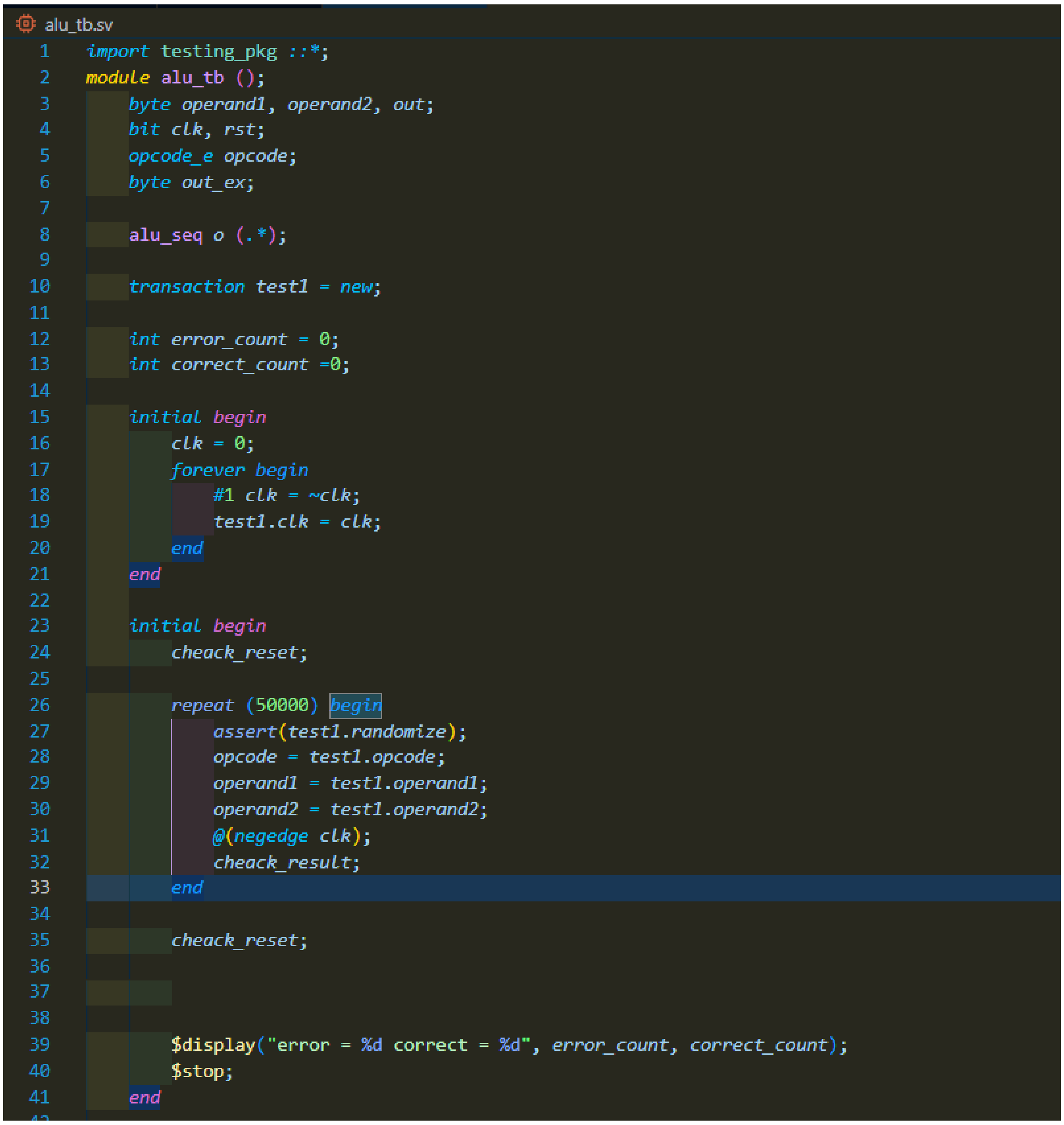
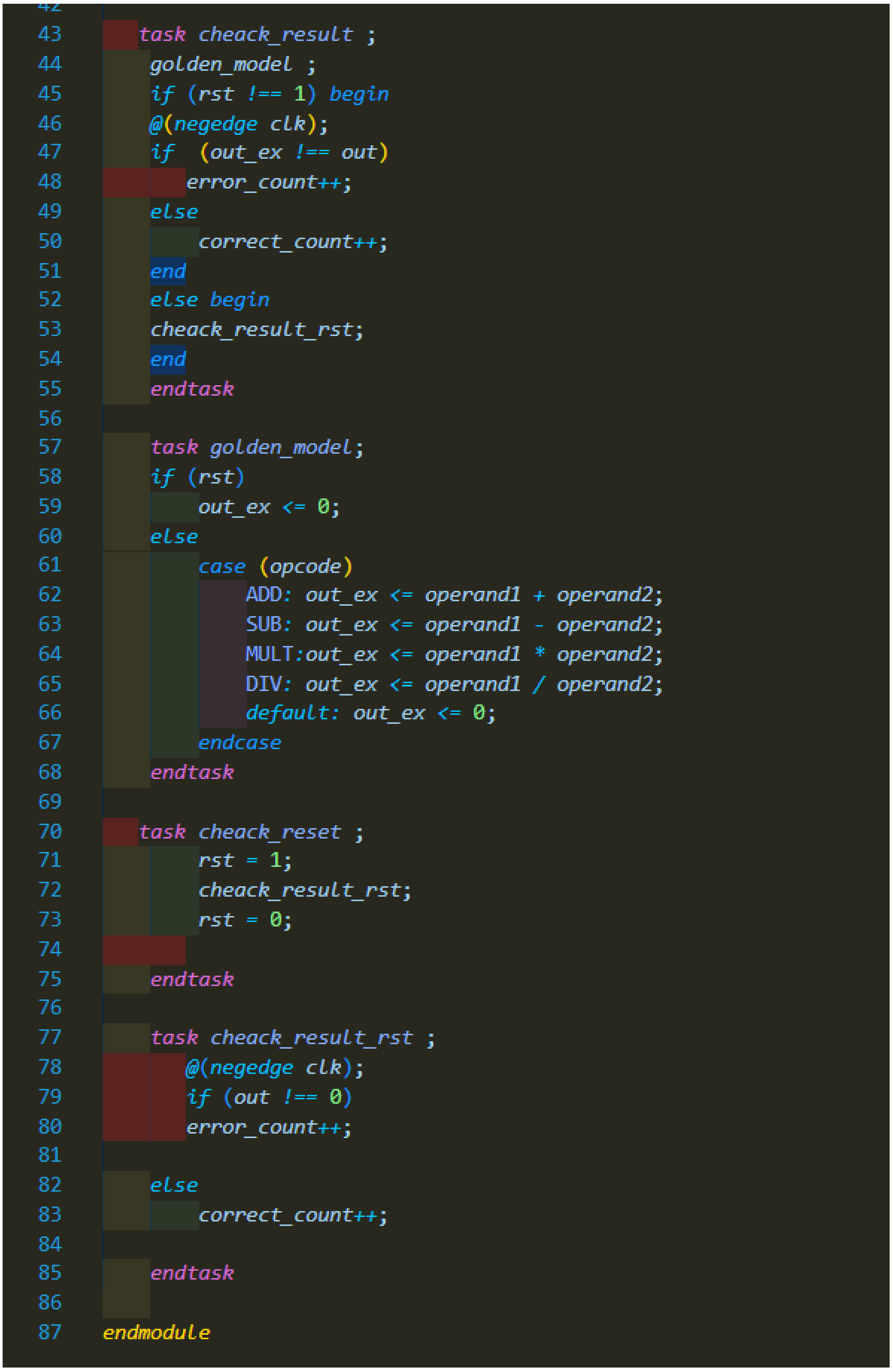
Assignment 3

Q1

CODE TB :

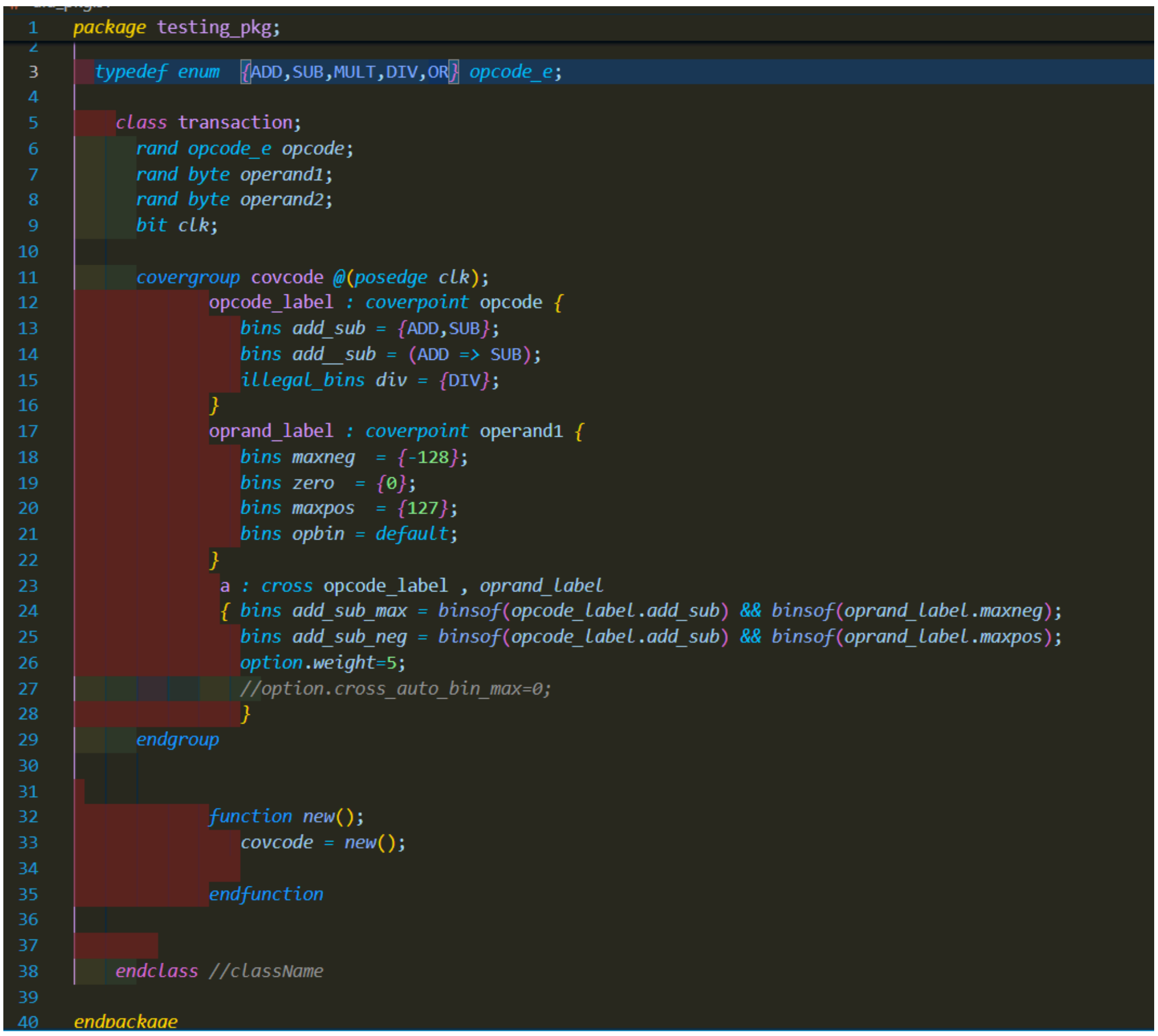

alu_tb.sv

```
import testing_pkg ::*;
module alu_tb ();
    byte operand1, operand2, out;
    bit clk, rst;
    opcode_e opcode;
    byte out_ex;

    alu_seq o (.*);

    transaction test1 = new;

    int error_count = 0;
    int correct_count =0;

    initial begin
        clk = 0;
        forever begin
            #1 clk = ~clk;
            test1.clk = clk;
        end
    end

    initial begin
        cheack_reset;

        repeat (50000) begin
            assert(test1.randomize);
            opcode = test1.opcode;
            operand1 = test1.operand1;
            operand2 = test1.operand2;
            @(negedge clk);
            cheack_result;
        end

        cheack_reset;


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



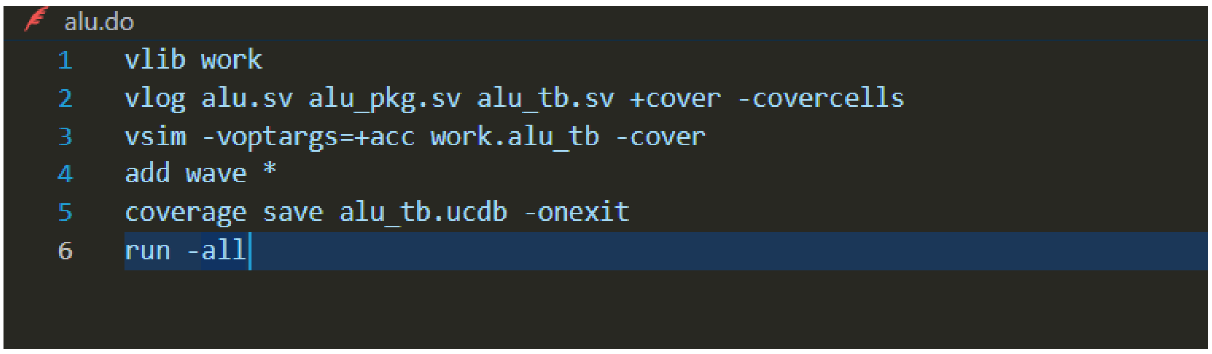
```
task cheack_result ;
    golden_model ;
    if (rst !== 1) begin
    @(negedge clk);
    if  (out_ex !== out)
        error_count++;
    else
        correct_count++;
    end
    else begin
    cheack_result_rst;
    end
    endtask

    task golden_model;
    if (rst)
        out_ex <= 0;
    else
        case (opcode)
            ADD: out_ex <= operand1 + operand2;
            SUB: out_ex <= operand1 - operand2;
            MULT:out_ex <= operand1 * operand2;
            DIV: out_ex <= operand1 / operand2;
            default: out_ex <= 0;
        endcase
    endtask

task cheack_reset ;
        rst = 1;
        cheack_result_rst;
        rst = 0;

    endtask

    task cheack_result_rst ;
       @(negedge clk);
       if (out !== 0)
       error_count++;

    else
        correct_count++;

    endtask

endmodule
```

CODE PKG :



```
package testing_pkg;

  typedef enum  {ADD,SUB,MULT,DIV,OR} opcode_e;

    class transaction;
      rand opcode_e opcode;
      rand byte operand1;
      rand byte operand2;
      bit clk;

      covergroup covcode @(posedge clk);
             opcode_label : coverpoint opcode {
                bins add_sub = {ADD,SUB};
                bins add__sub = (ADD => SUB);
                illegal_bins div = {DIV};
             }
             oprand_label : coverpoint operand1 {
                bins maxneg  = {-128};
                bins zero  = {0};
                bins maxpos  = {127};
                bins opbin = default;
             }
              a : cross opcode_label , oprand_label
              { bins add_sub_max = binsof(opcode_label.add_sub) && binsof(oprand_label.maxneg);
                bins add_sub_neg = binsof(opcode_label.add_sub) && binsof(oprand_label.maxpos);
                option.weight=5;
                //option.cross_auto_bin_max=0;
                }
      endgroup


             function new();
                covcode = new();

             endfunction


    endclass //className

endpackage
```

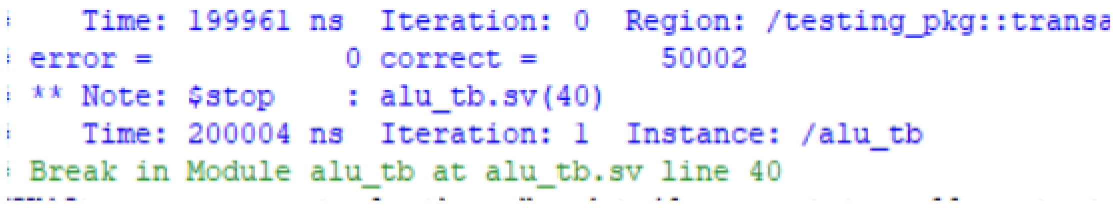
DO FILE :



alu.do

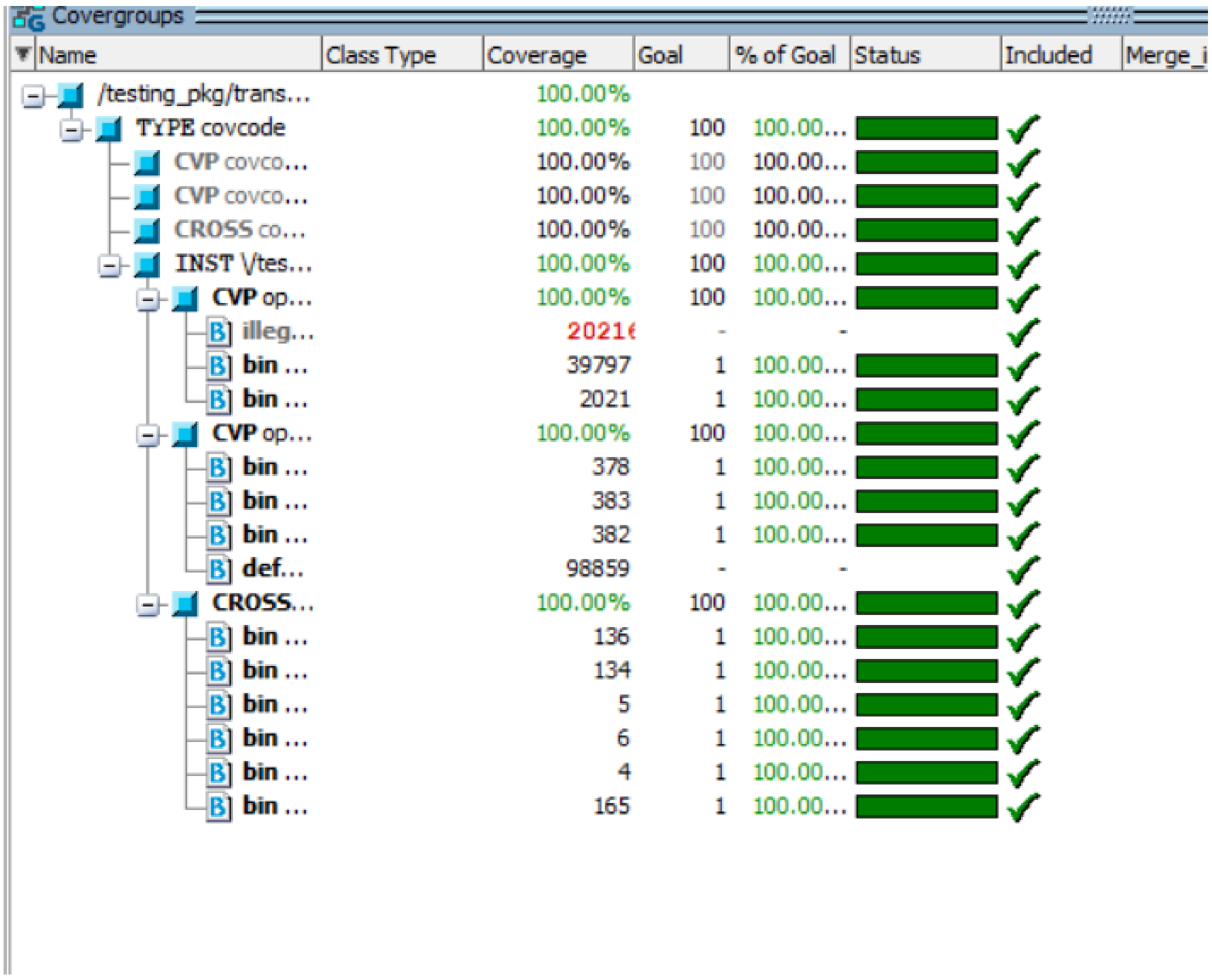
```
vlib work
vlog alu.sv alu_pkg.sv alu_tb.sv +cover -covercells
vsim -voptargs=+acc work.alu_tb -cover
add wave *
coverage save alu_tb.ucdb -onexit
run -all
```

DISPLAY :



```
   Time: 199961 ns  Iteration: 0  Region: /testing_pkg::transa
error =           0 correct =        50002
** Note: $stop    : alu_tb.sv(40)
   Time: 200004 ns  Iteration: 1  Instance: /alu_tb
Break in Module alu_tb at alu_tb.sv line 40
```

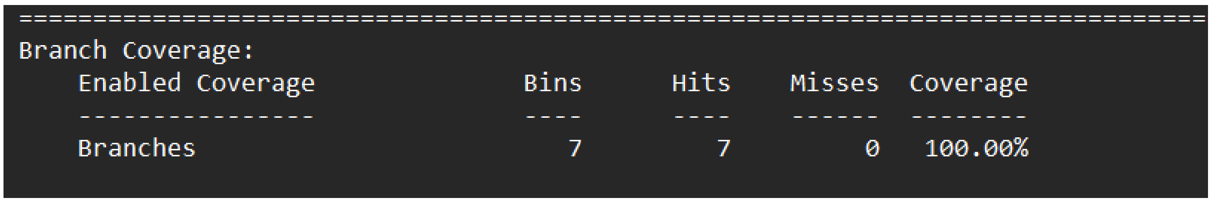
COVERGROUP :



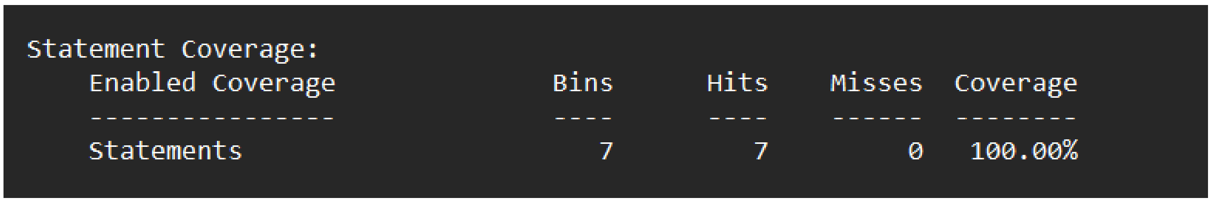
Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_i
/testing_pkg/trans...		100.00%					
TYPE covcode		100.00%	100	100.00...	🟩	✓	
CVP covco...		100.00%	100	100.00...	🟩	✓	
CVP covco...		100.00%	100	100.00...	🟩	✓	
CROSS co...		100.00%	100	100.00...	🟩	✓	
INST \/tes...		100.00%	100	100.00...	🟩	✓	
CVP op...		100.00%	100	100.00...	🟩	✓	
illeg...		2021	-	-		✓	
bin ...		39797	1	100.00...	🟩	✓	
bin ...		2021	1	100.00...	🟩	✓	
CVP op...		100.00%	100	100.00...	🟩	✓	
bin ...		378	1	100.00...	🟩	✓	
bin ...		383	1	100.00...	🟩	✓	
bin ...		382	1	100.00...	🟩	✓	
def...		98859	-	-		✓	
CROSS...		100.00%	100	100.00...	🟩	✓	
bin ...		136	1	100.00...	🟩	✓	
bin ...		134	1	100.00...	🟩	✓	
bin ...		5	1	100.00...	🟩	✓	
bin ...		6	1	100.00...	🟩	✓	
bin ...		4	1	100.00...	🟩	✓	
bin ...		165	1	100.00...	🟩	✓	

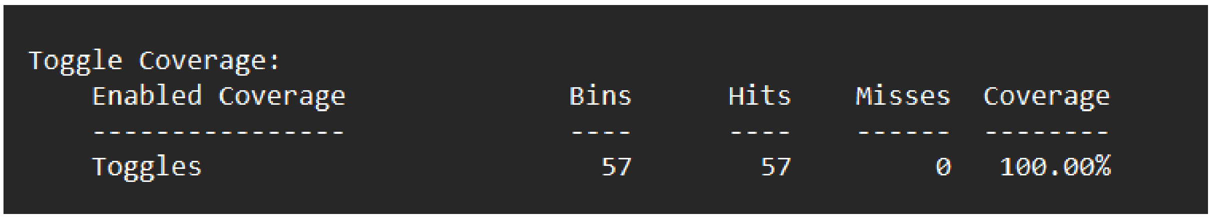
COVERAGR :



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



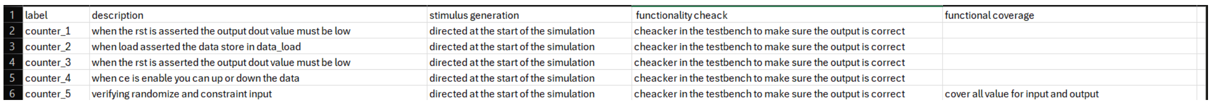
```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       7         7         0   100.00%
```



```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         57        57         0   100.00%
```

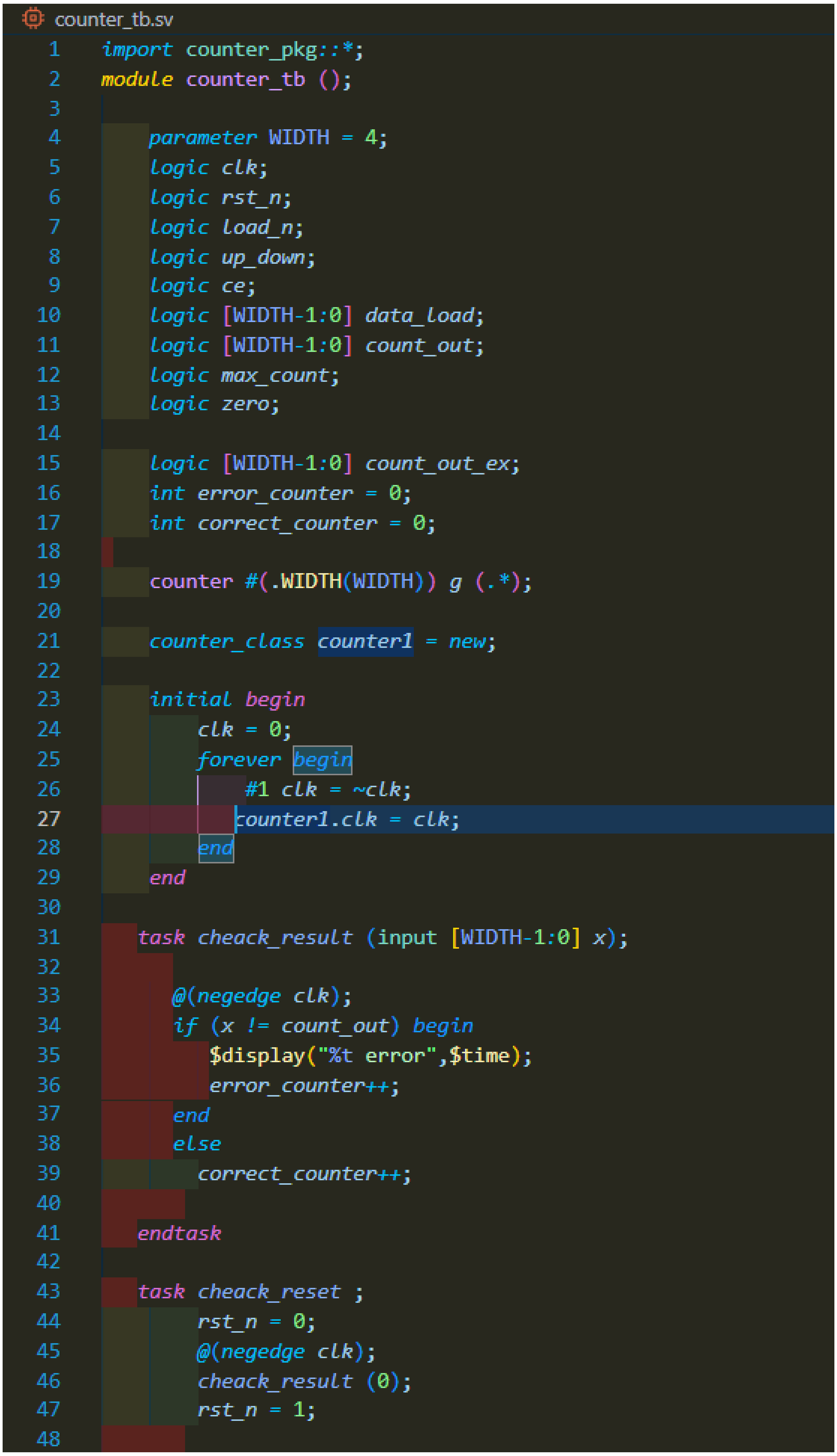
Q2

Verification plan :

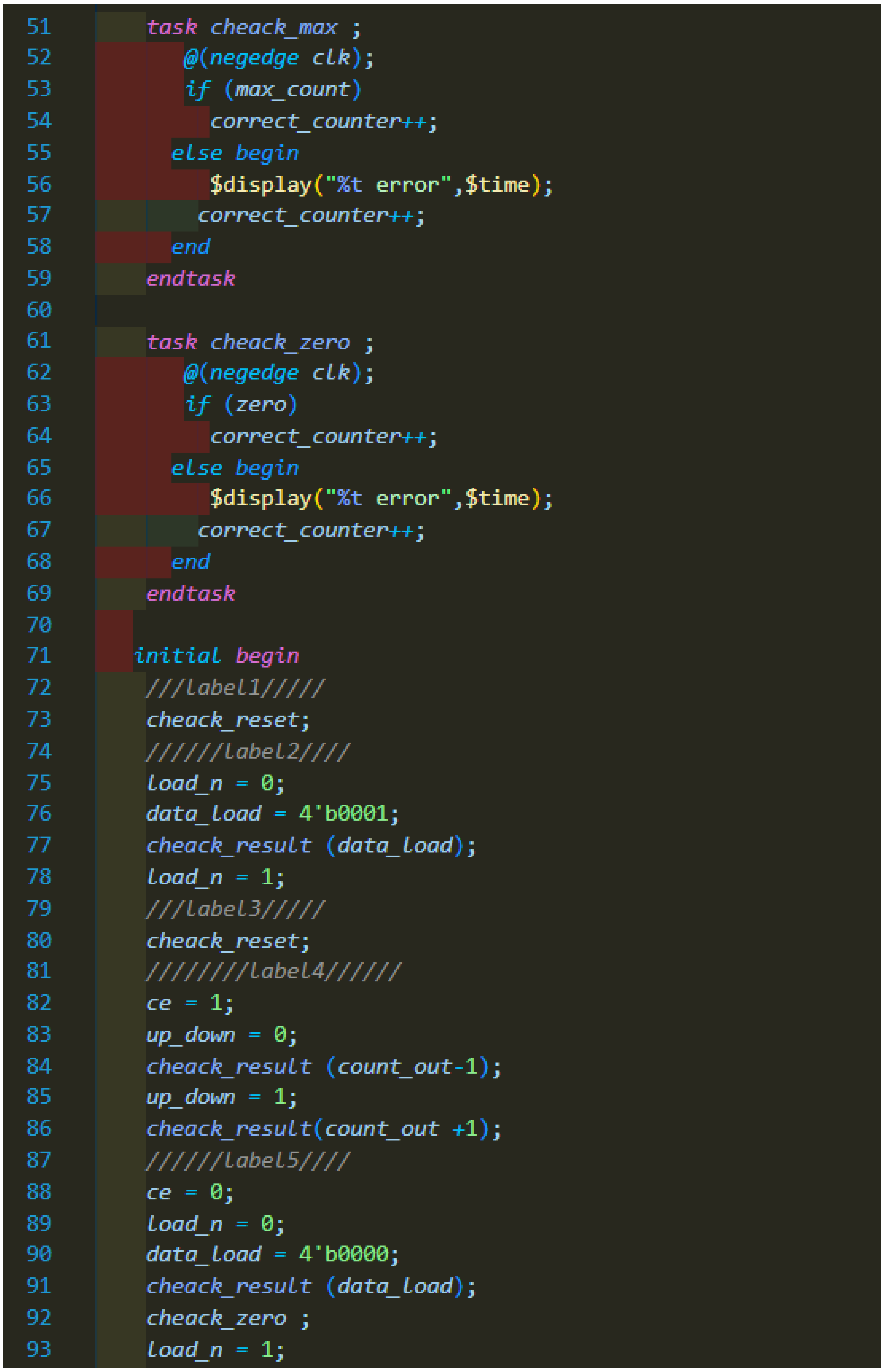


label	description	stimulus generation	functionality cheack	functional coverage
counter_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
counter_2	when load asserted the data store in data_load	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
counter_3	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
counter_4	when ce is enable you can up or down the data	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
counter_5	verifying randomize and constraint input	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover all value for input and output

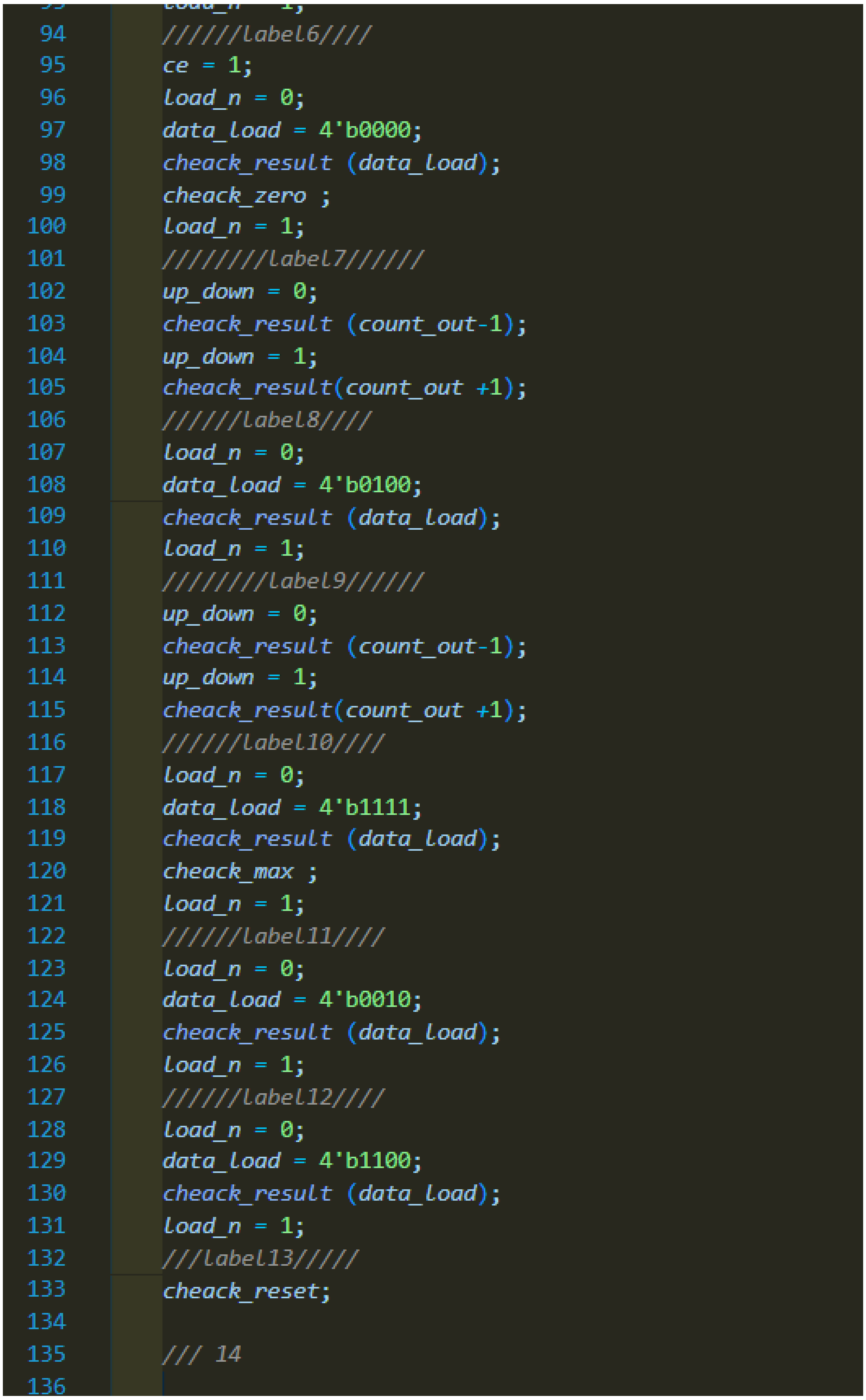
CODE TB :


counter_tb.sv

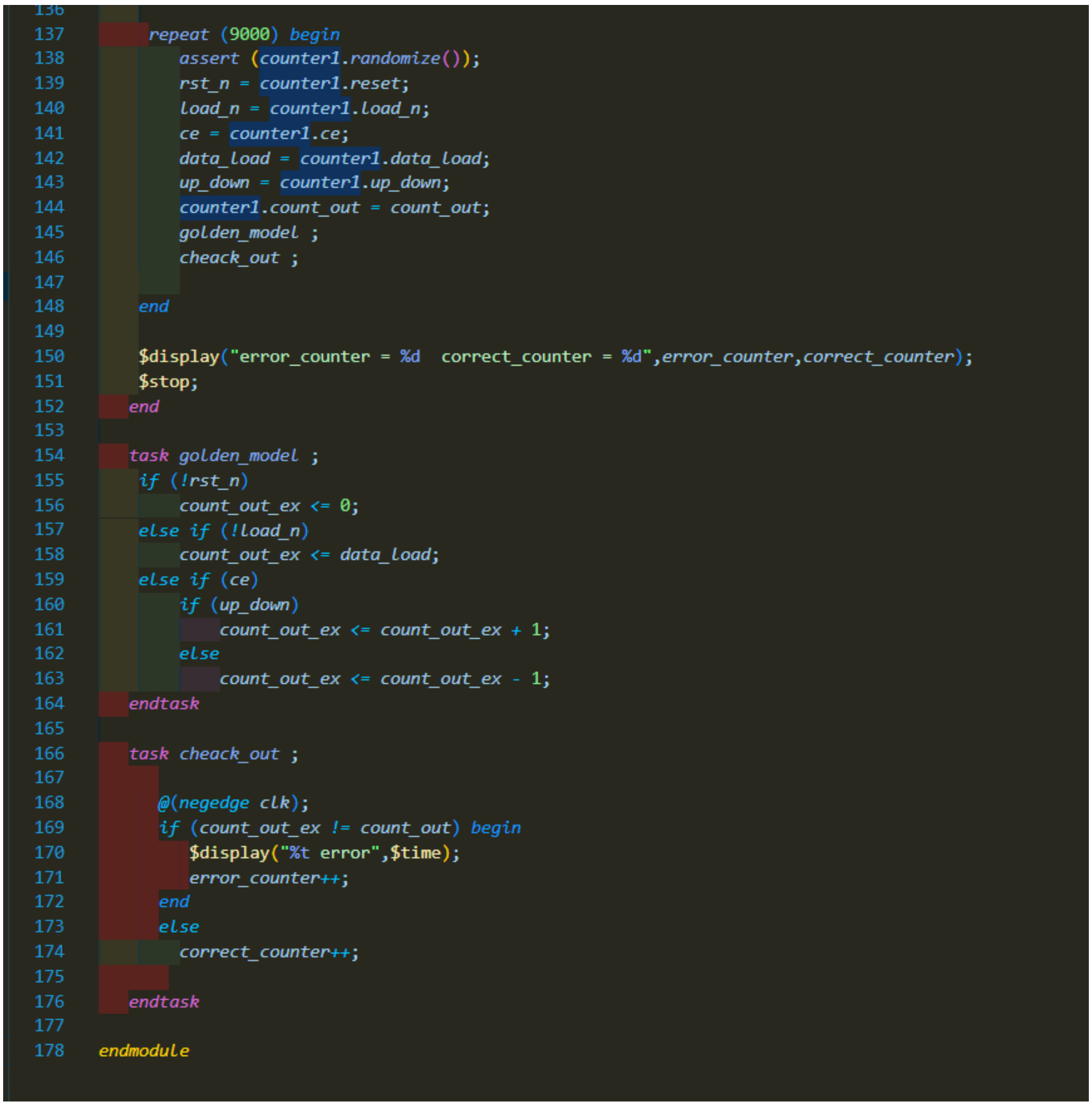
```systemverilog
import counter_pkg::*;
module counter_tb ();

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

    logic [WIDTH-1:0] count_out_ex;
    int error_counter = 0;
    int correct_counter = 0;

    counter #(.WIDTH(WIDTH)) g (.*);

    counter_class counter1 = new;

    initial begin
        clk = 0;
        forever begin
            #1 clk = ~clk;
           counter1.clk = clk;
        end
    end

   task cheack_result (input [WIDTH-1:0] x);

      @(negedge clk);
      if (x != count_out) begin
         $display("%t error",$time);
         error_counter++;
      end
      else
        correct_counter++;

   endtask

   task cheack_reset ;
        rst_n = 0;
        @(negedge clk);
        cheack_result (0);
        rst_n = 1;

```



```
task cheack_max ;
   @(negedge clk);
   if (max_count)
     correct_counter++;
  else begin
     $display("%t error",$time);
    correct_counter++;
  end
endtask

task cheack_zero ;
   @(negedge clk);
   if (zero)
     correct_counter++;
  else begin
     $display("%t error",$time);
    correct_counter++;
  end
endtask

initial begin
 ///label1/////
 cheack_reset;
 //////label2////
 load_n = 0;
 data_load = 4'b0001;
 cheack_result (data_load);
 load_n = 1;
 ///label3/////
 cheack_reset;
 ////////label4//////
 ce = 1;
 up_down = 0;
 cheack_result (count_out-1);
 up_down = 1;
 cheack_result(count_out +1);
 //////label5////
 ce = 0;
 load_n = 0;
 data_load = 4'b0000;
 cheack_result (data_load);
 cheack_zero ;
 load_n = 1;
```

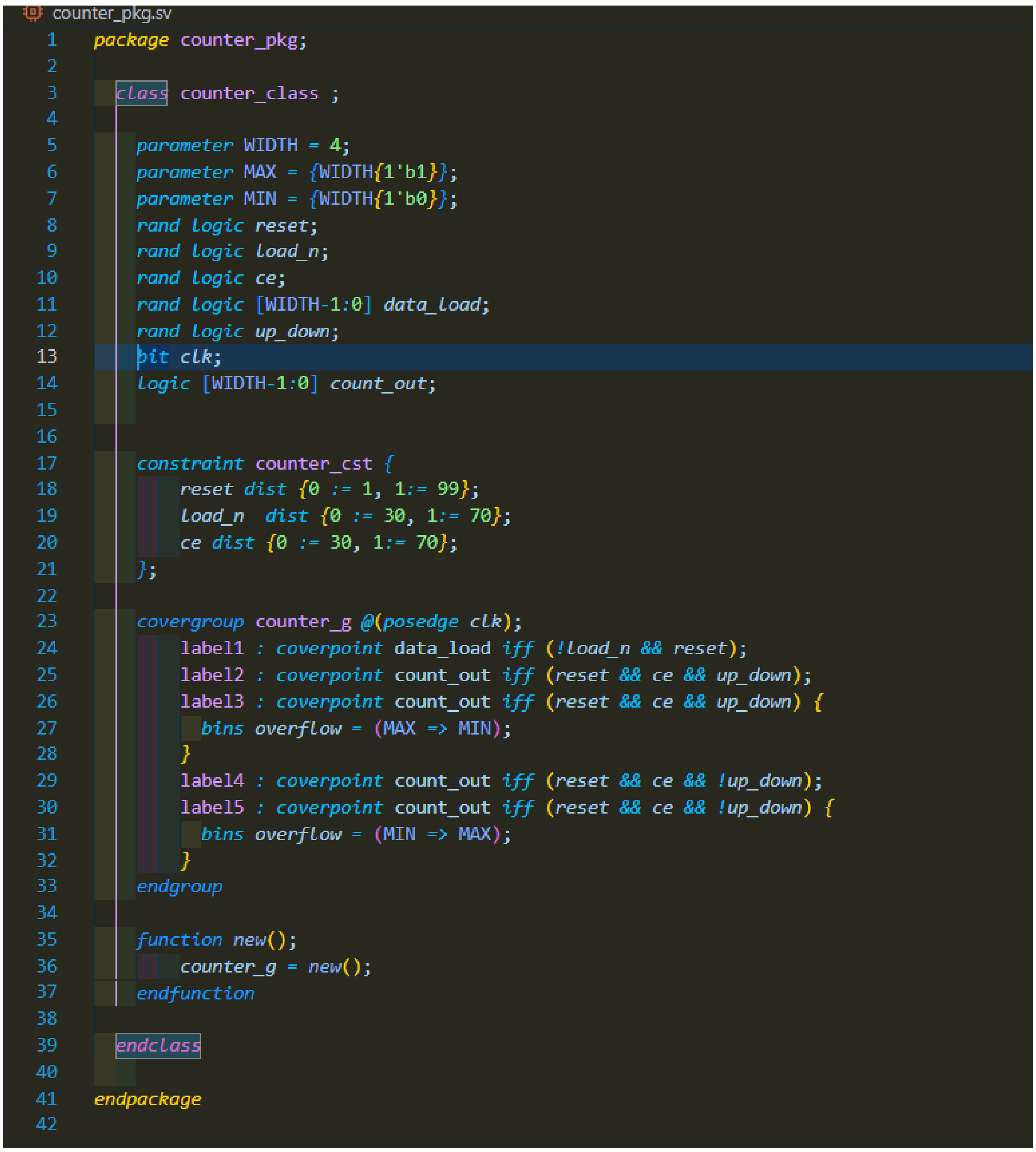


```verilog
load_n = 1;
//////label6////
ce = 1;
load_n = 0;
data_load = 4'b0000;
cheack_result (data_load);
cheack_zero ;
load_n = 1;
////////label7//////
up_down = 0;
cheack_result (count_out-1);
up_down = 1;
cheack_result(count_out +1);
//////label8////
load_n = 0;
data_load = 4'b0100;
cheack_result (data_load);
load_n = 1;
////////label9//////
up_down = 0;
cheack_result (count_out-1);
up_down = 1;
cheack_result(count_out +1);
//////label10////
load_n = 0;
data_load = 4'b1111;
cheack_result (data_load);
cheack_max ;
load_n = 1;
//////label11////
load_n = 0;
data_load = 4'b0010;
cheack_result (data_load);
load_n = 1;
//////label12////
load_n = 0;
data_load = 4'b1100;
cheack_result (data_load);
load_n = 1;
///label13/////
cheack_reset;

/// 14

```



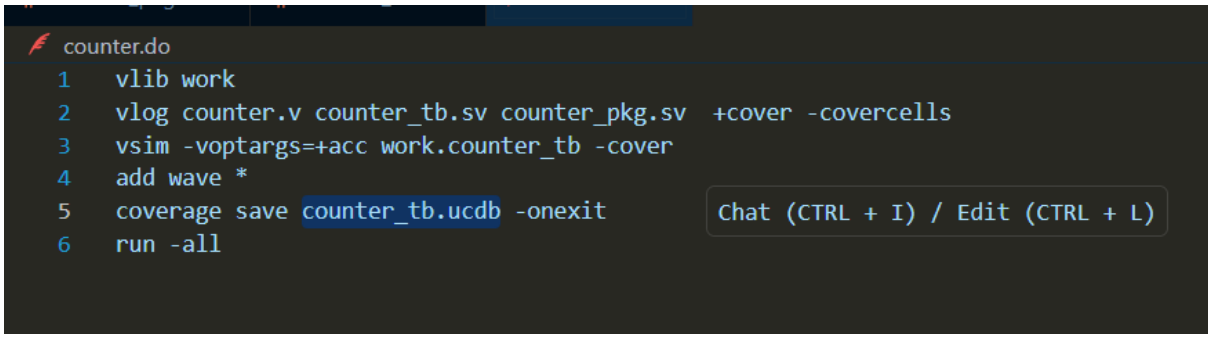
```
      repeat (9000) begin
          assert (counter1.randomize());
          rst_n = counter1.reset;
          load_n = counter1.load_n;
          ce = counter1.ce;
          data_load = counter1.data_load;
          up_down = counter1.up_down;
          counter1.count_out = count_out;
          golden_model ;
          cheack_out ;

      end

      $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
      $stop;
    end

    task golden_model ;
      if (!rst_n)
          count_out_ex <= 0;
      else if (!load_n)
          count_out_ex <= data_load;
      else if (ce)
          if (up_down)
              count_out_ex <= count_out_ex + 1;
          else
              count_out_ex <= count_out_ex - 1;
    endtask

    task cheack_out ;

        @(negedge clk);
        if (count_out_ex != count_out) begin
           $display("%t error",$time);
           error_counter++;
        end
        else
          correct_counter++;

    endtask

endmodule
```

CODE PKG :


counter_pkg.sv

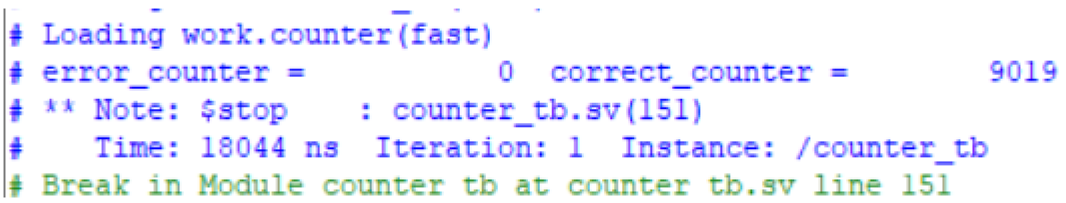
```
package counter_pkg;

  class counter_class ;

    parameter WIDTH = 4;
    parameter MAX = {WIDTH{1'b1}};
    parameter MIN = {WIDTH{1'b0}};
    rand logic reset;
    rand logic load_n;
    rand logic ce;
    rand logic [WIDTH-1:0] data_load;
    rand logic up_down;
    bit clk;
    logic [WIDTH-1:0] count_out;


    constraint counter_cst {
        reset dist {0 := 1, 1:= 99};
        load_n  dist {0 := 30, 1:= 70};
        ce dist {0 := 30, 1:= 70};
    };

    covergroup counter_g @(posedge clk);
        label1 : coverpoint data_load iff (!load_n && reset);
        label2 : coverpoint count_out iff (reset && ce && up_down);
        label3 : coverpoint count_out iff (reset && ce && up_down) {
          bins overflow = (MAX => MIN);
        }
        label4 : coverpoint count_out iff (reset && ce && !up_down);
        label5 : coverpoint count_out iff (reset && ce && !up_down) {
          bins overflow = (MIN => MAX);
        }
    endgroup

    function new();
        counter_g = new();
    endfunction

  endclass

endpackage

```

DO FILE :


counter.do

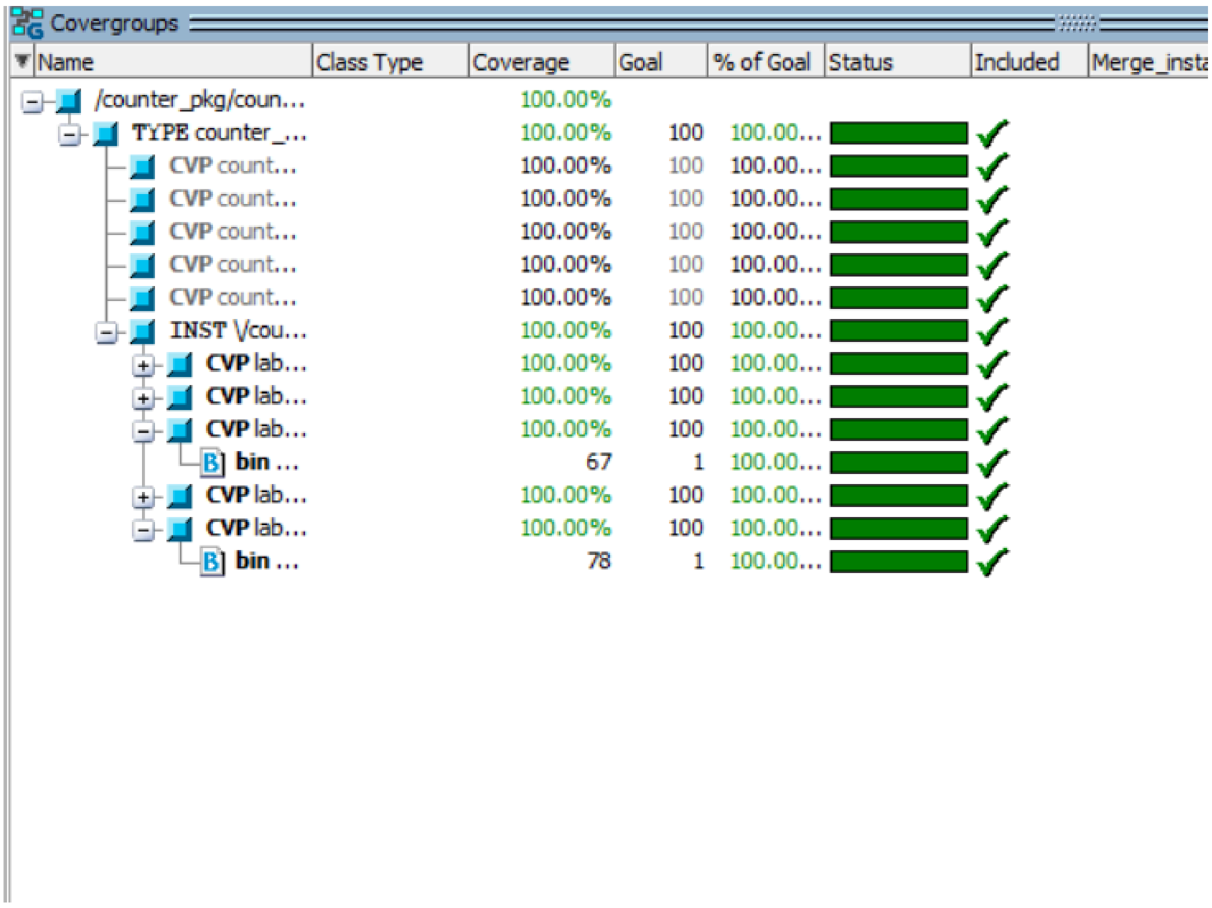
```
vlib work
vlog counter.v counter_tb.sv counter_pkg.sv  +cover -covercells
vsim -voptargs=+acc work.counter_tb -cover
add wave *
coverage save counter_tb.ucdb -onexit
run -all
```

DISPLAY :



```
# Loading work.counter(fast)
# error_counter =           0   correct_counter =        9019
# ** Note: $stop    : counter_tb.sv(151)
#    Time: 18044 ns  Iteration: 1  Instance: /counter_tb
# Break in Module counter tb at counter tb.sv line 151
```

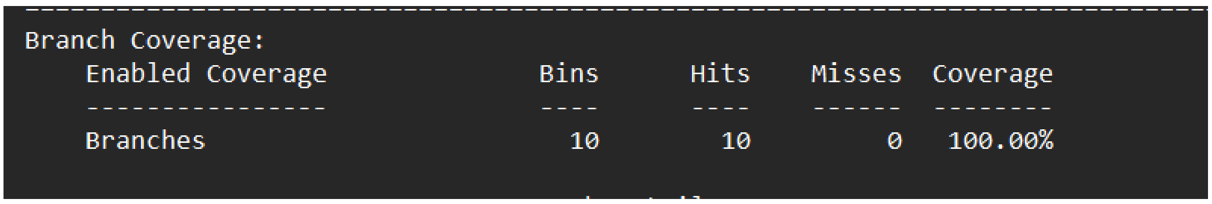
COVERGROUP :



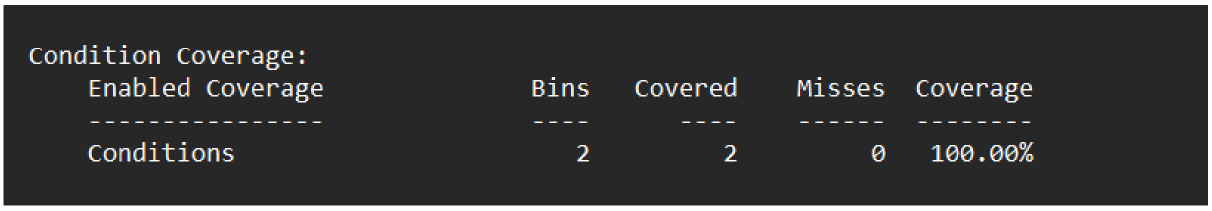
Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_insta
/counter_pkg/coun...		100.00%					
TYPE counter_...		100.00%	100	100.00...	🟩	✓	
CVP count...		100.00%	100	100.00...	🟩	✓	
CVP count...		100.00%	100	100.00...	🟩	✓	
CVP count...		100.00%	100	100.00...	🟩	✓	
CVP count...		100.00%	100	100.00...	🟩	✓	
CVP count...		100.00%	100	100.00...	🟩	✓	
INST \/cou...		100.00%	100	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		67	1	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		78	1	100.00...	🟩	✓	

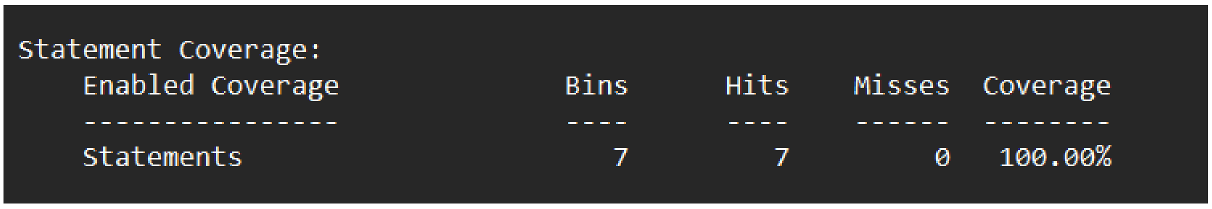
CODE COVERAGE :



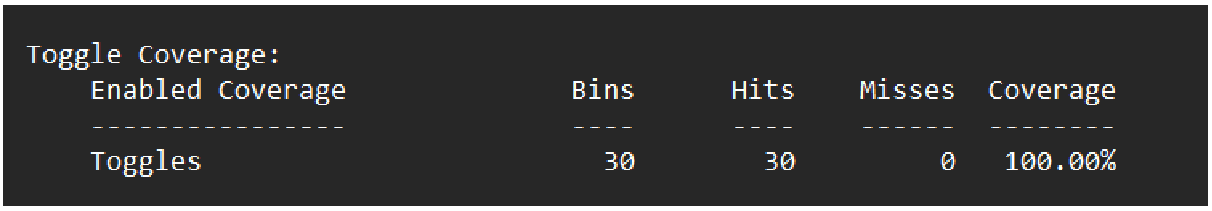
```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%
```



```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       2         2         0   100.00%
```



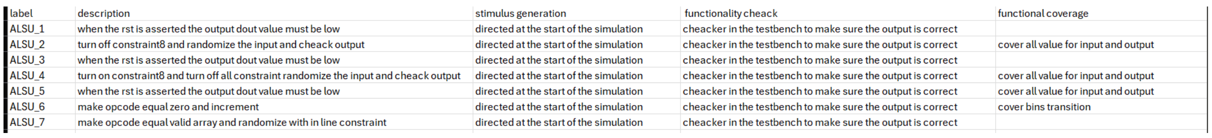
```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       7         7         0   100.00%
```



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

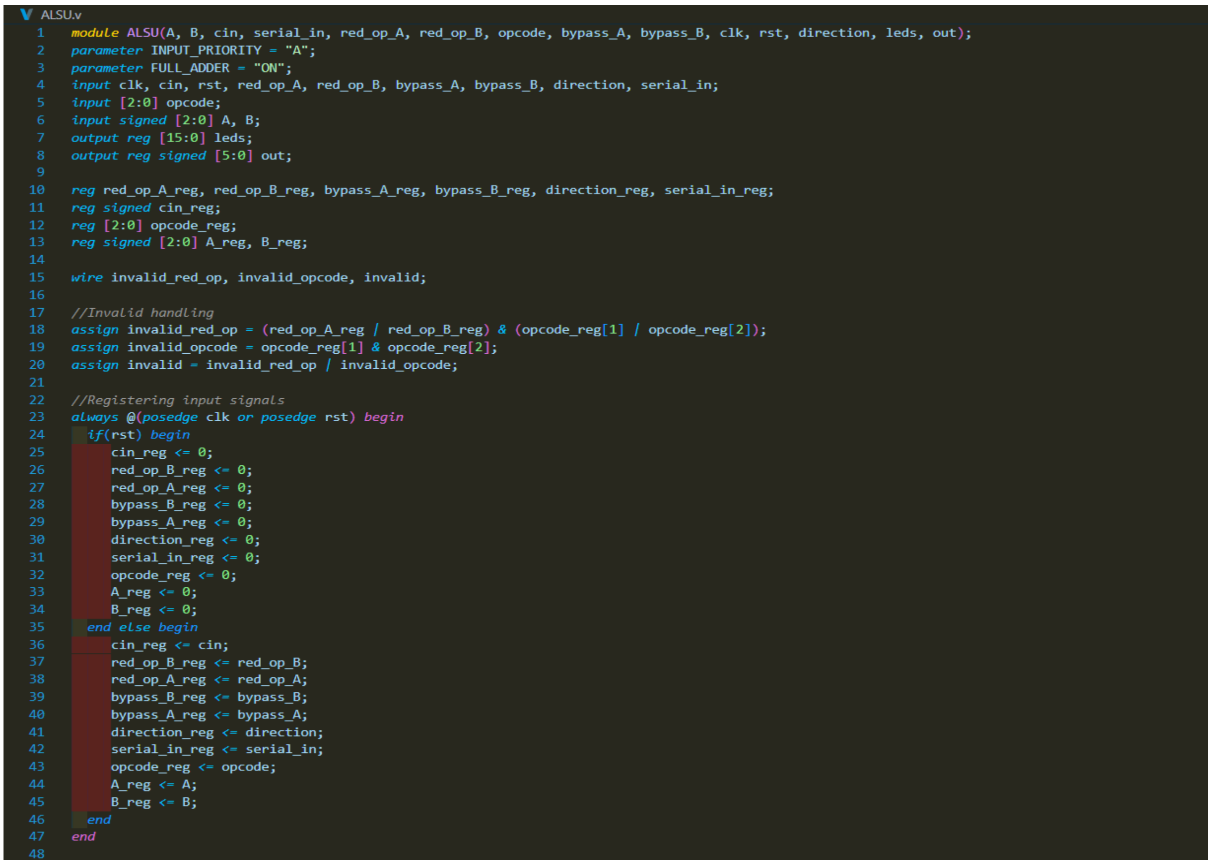
Q3

Verification plan :

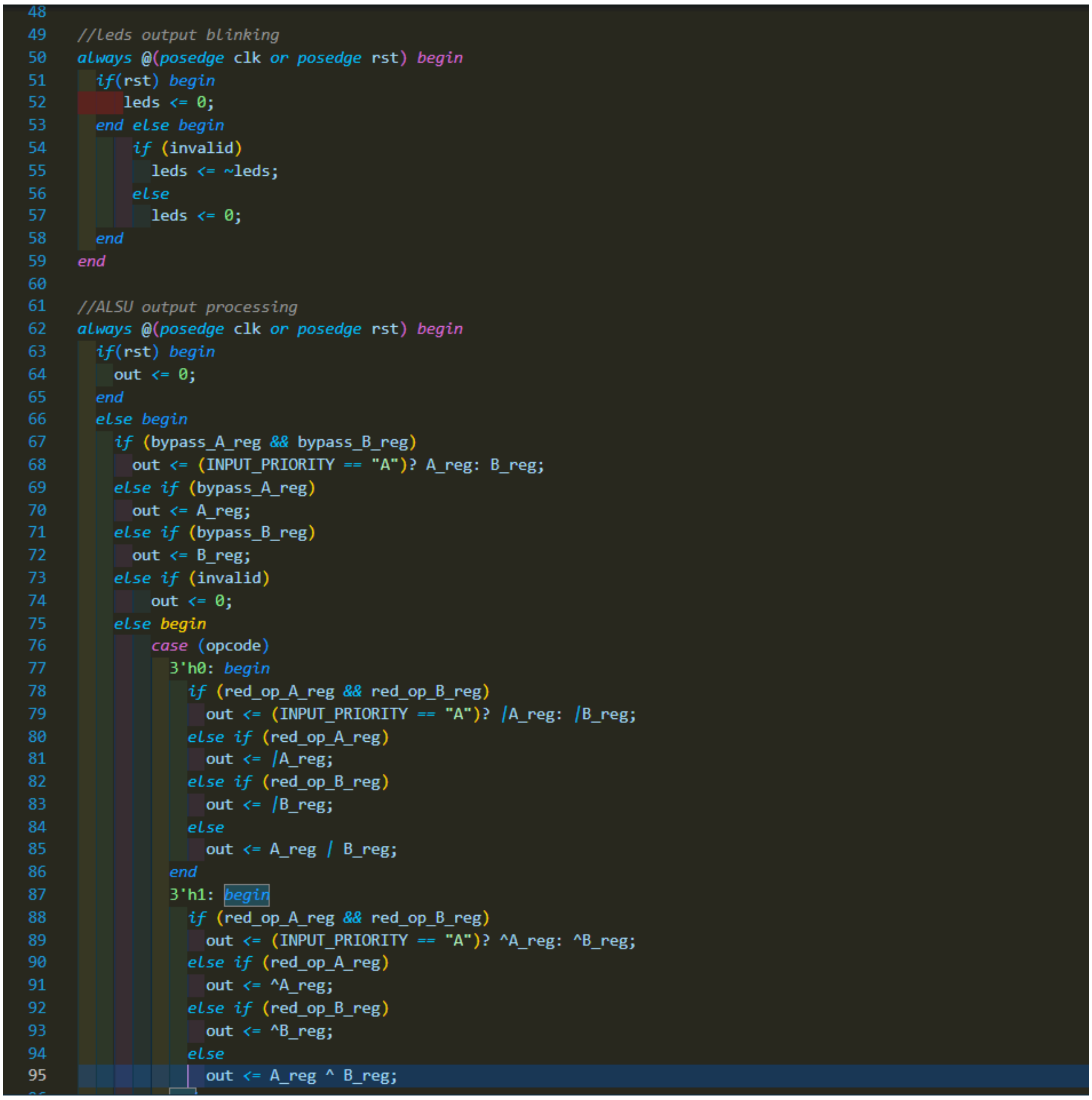


label	description	stimulus generation	functionality cheack	functional coverage
ALSU_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
ALSU_2	turn off constraint8 and randomize the input and cheack output	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_3	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
ALSU_4	turn on constraint8 and turn off all constraint randomize the input and cheack output	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_5	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_6	make opcode equal zero and increment	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover bins transition
ALSU_7	make opcode equal valid array and randomize with in line constraint	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	

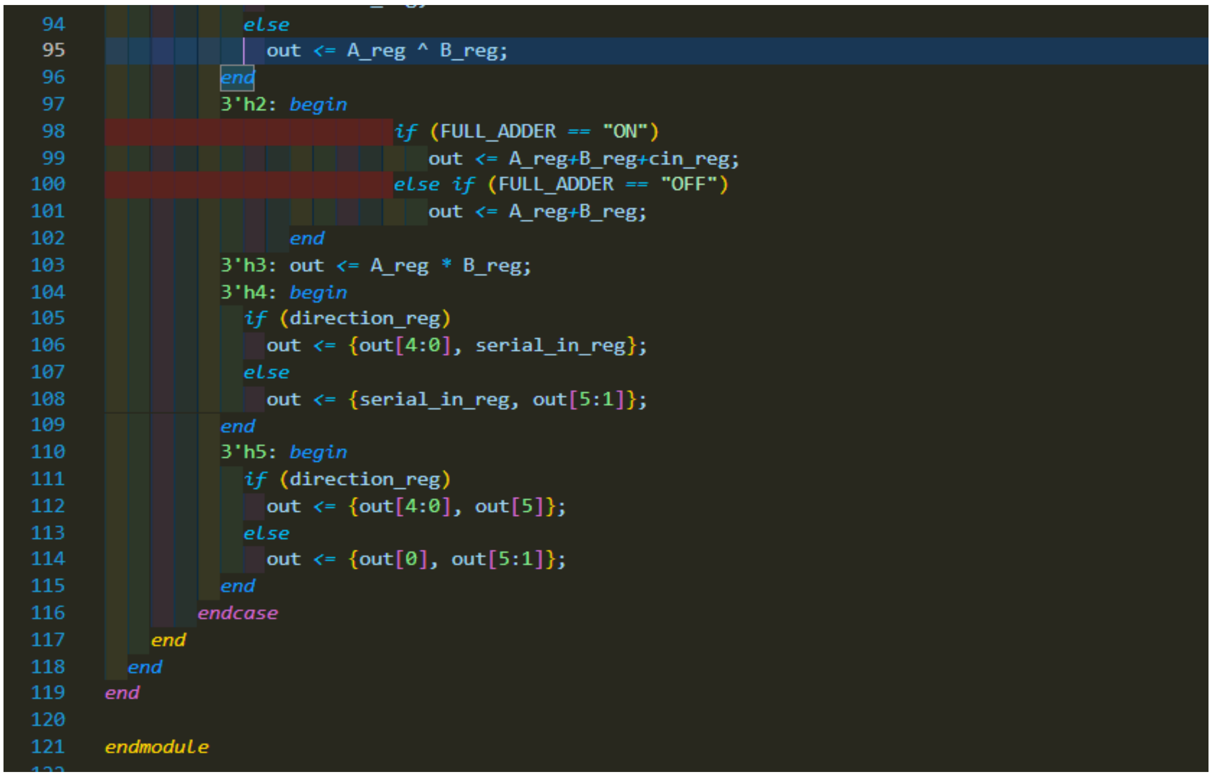
CODE DESIGN :



```verilog
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, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;
output reg [15:0] leds;
output reg signed [5:0] out;

reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg signed cin_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

```

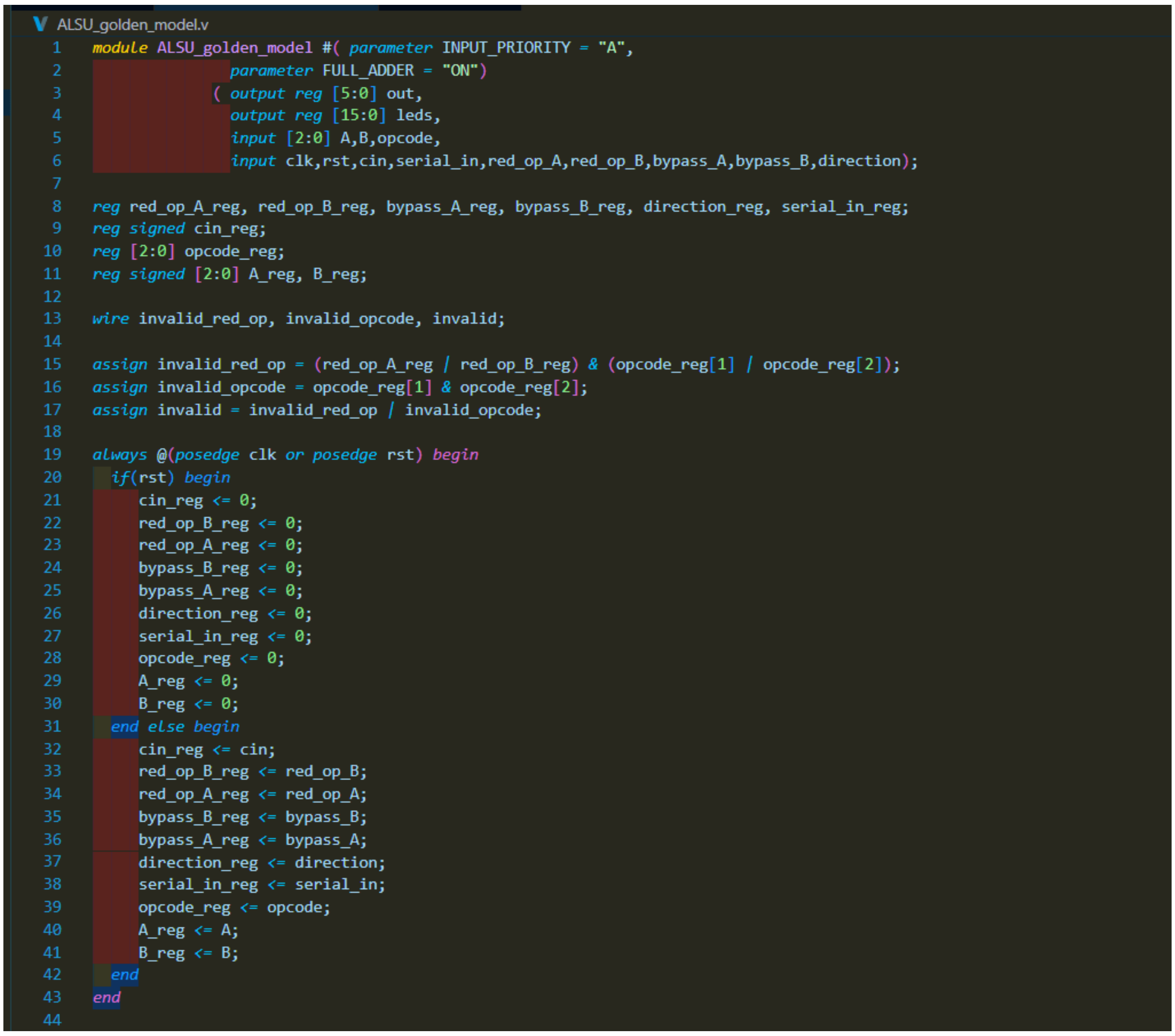


```verilog

//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

//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)
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            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;
            else if (red_op_A_reg)
              out <= ^A_reg;
            else if (red_op_B_reg)
              out <= ^B_reg;
            else
              out <= A_reg ^ B_reg;
```



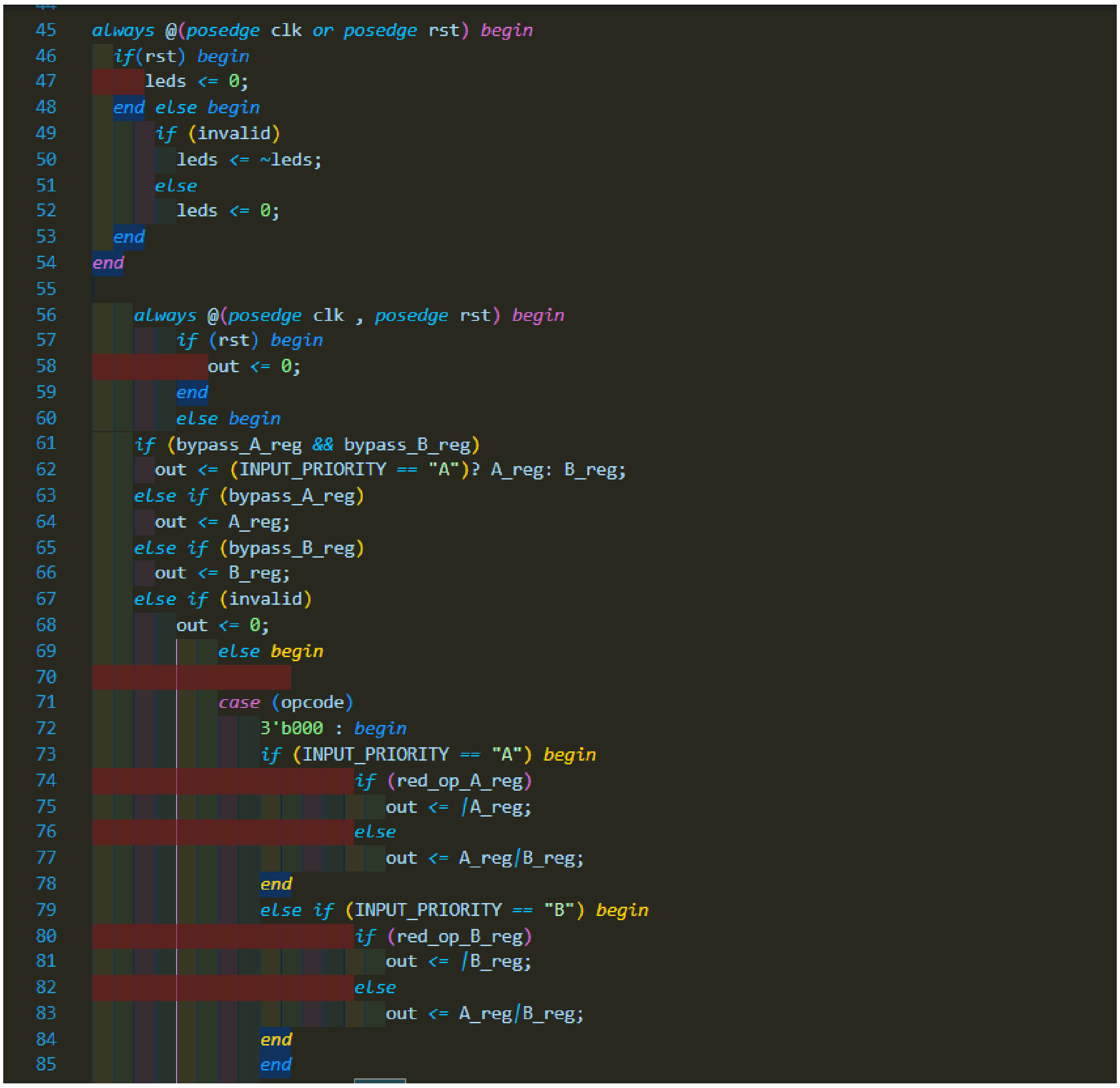
```
      else
       out <= A_reg ^ B_reg;
     end
     3'h2: begin
            if (FULL_ADDER == "ON")
              out <= A_reg+B_reg+cin_reg;
            else if (FULL_ADDER == "OFF")
              out <= A_reg+B_reg;
        end
     3'h3: out <= A_reg * B_reg;
     3'h4: begin
      if (direction_reg)
       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
    endcase
  end
 end
end

endmodule
```

GOLDEN MODEL :

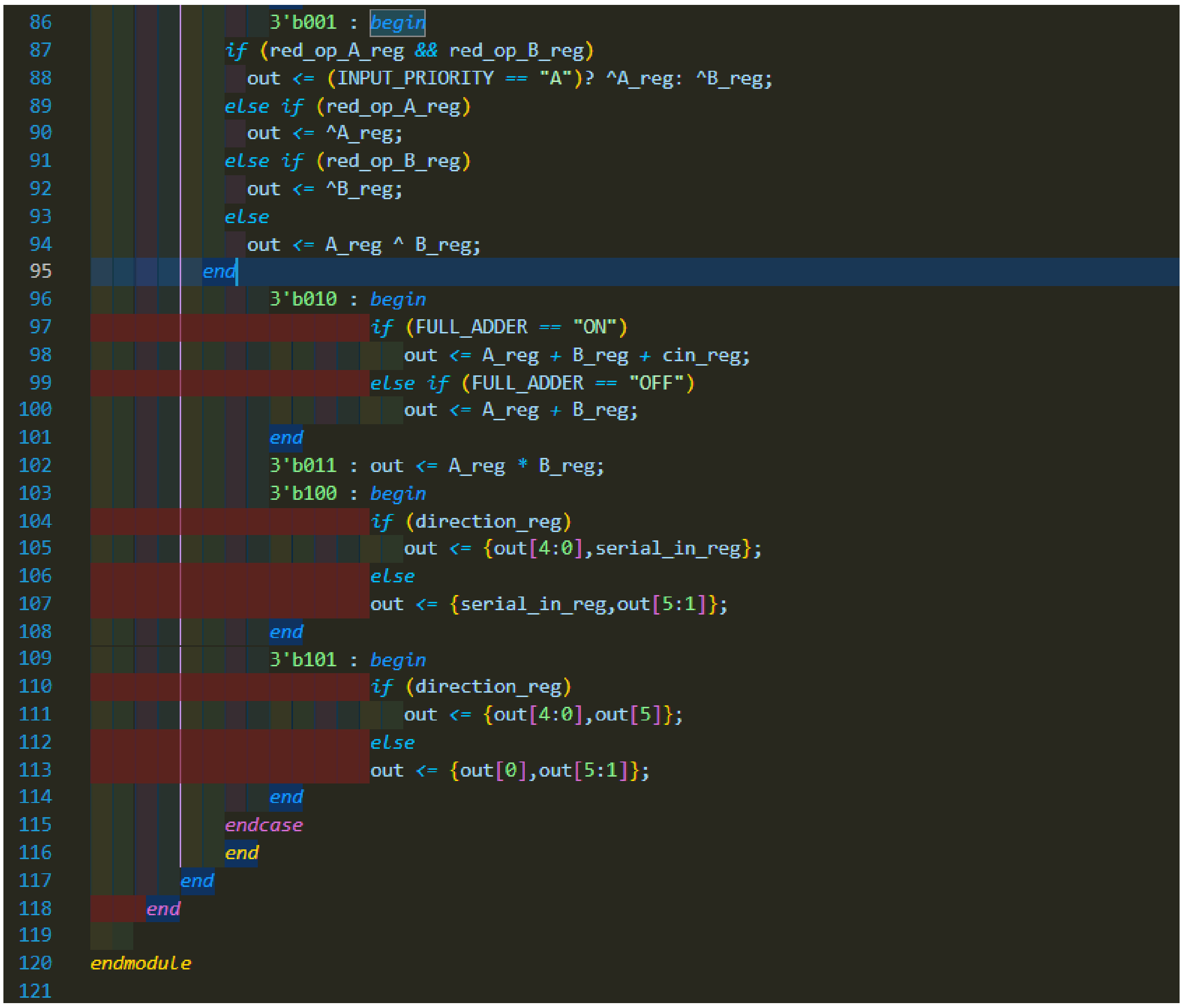


ALSU_golden_model.v

```verilog
module ALSU_golden_model #( parameter INPUT_PRIORITY = "A",
                           parameter FULL_ADDER = "ON")
                         ( output reg [5:0] out,
                           output reg [15:0] leds,
                           input [2:0] A,B,opcode,
                           input clk,rst,cin,serial_in,red_op_A,red_op_B,bypass_A,bypass_B,direction);

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

wire invalid_red_op, invalid_opcode, invalid;

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;

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
```

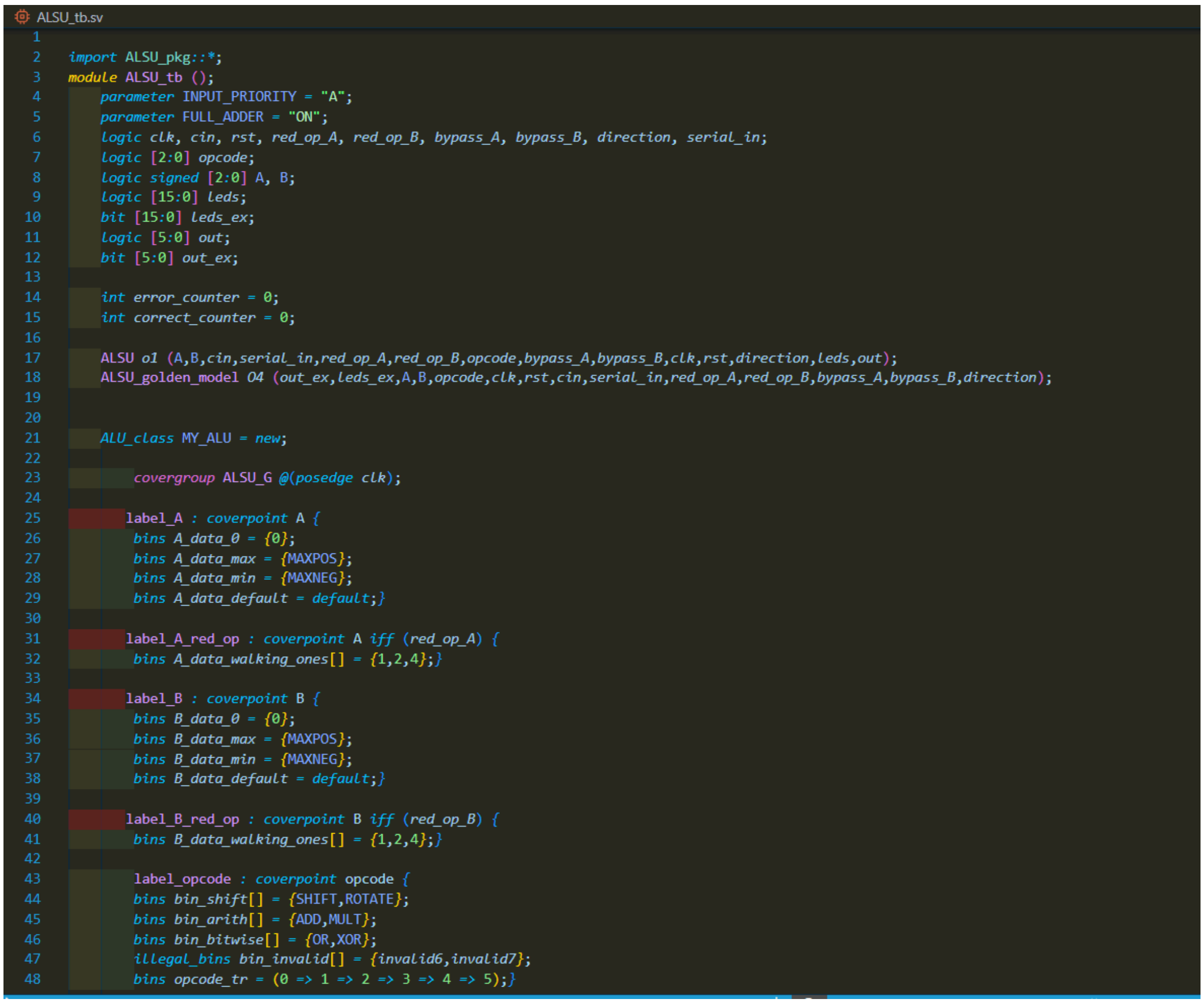


```
always @(posedge clk or posedge rst) begin
  if(rst) begin
     leds <= 0;
  end else begin
      if (invalid)
        leds <= ~leds;
      else
        leds <= 0;
  end
end

    always @(posedge clk , 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)
                3'b000 : begin
                if (INPUT_PRIORITY == "A") begin
                         if (red_op_A_reg)
                            out <= |A_reg;
                         else
                            out <= A_reg|B_reg;
                end
                else if (INPUT_PRIORITY == "B") begin
                         if (red_op_B_reg)
                            out <= |B_reg;
                         else
                            out <= A_reg|B_reg;
                end
                end
```

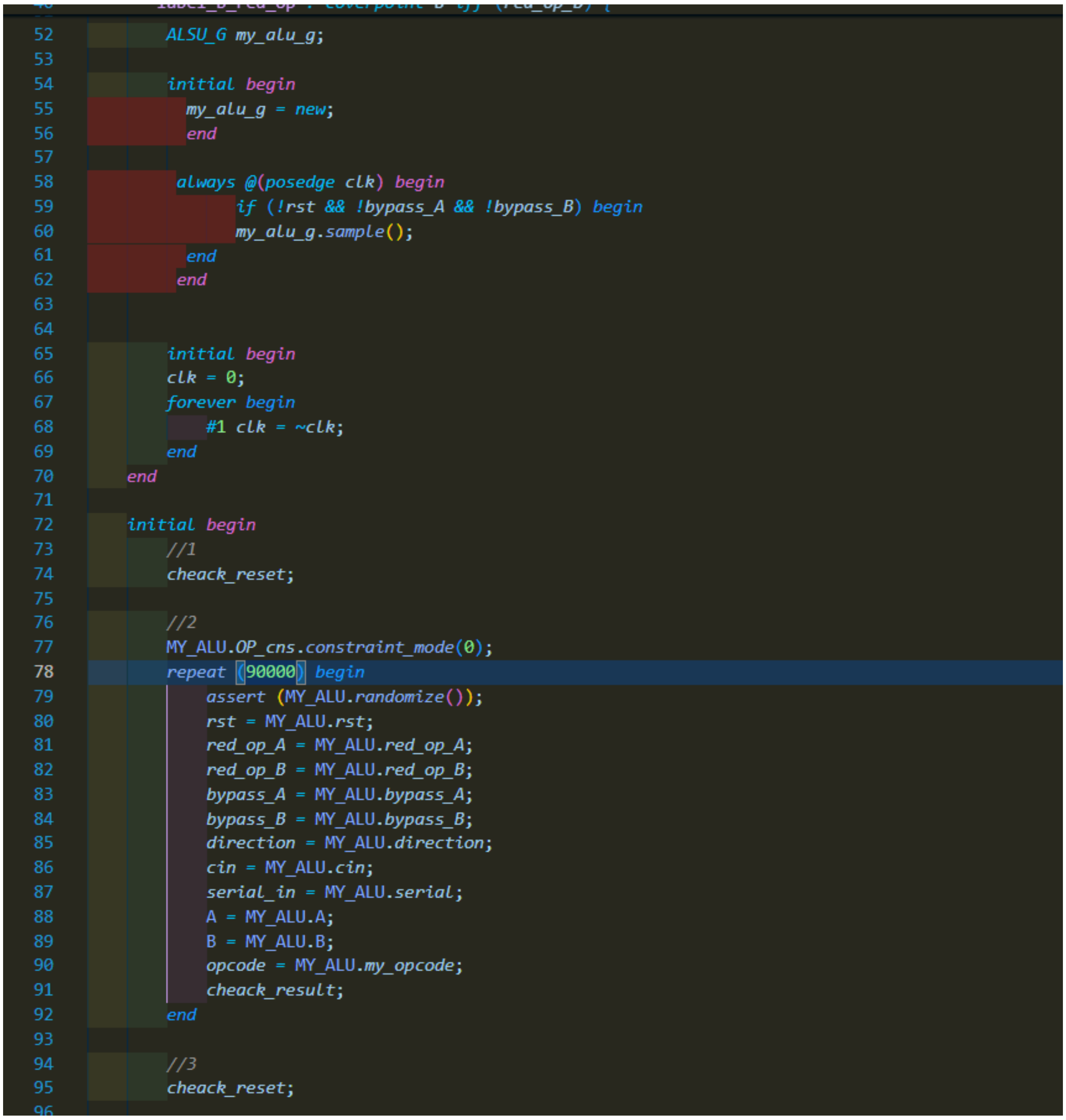


```
                3'b001 : begin
            if (red_op_A_reg && red_op_B_reg)
              out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              out <= ^A_reg;
            else if (red_op_B_reg)
              out <= ^B_reg;
            else
              out <= A_reg ^ B_reg;
          end
                3'b010 : begin
                         if (FULL_ADDER == "ON")
                            out <= A_reg + B_reg + cin_reg;
                         else if (FULL_ADDER == "OFF")
                            out <= A_reg + B_reg;
                end
                3'b011 : out <= A_reg * B_reg;
                3'b100 : begin
                         if (direction_reg)
                            out <= {out[4:0],serial_in_reg};
                         else
                         out <= {serial_in_reg,out[5:1]};
                end
                3'b101 : begin
                         if (direction_reg)
                            out <= {out[4:0],out[5]};
                         else
                         out <= {out[0],out[5:1]};
                end
            endcase
            end
        end
     end

endmodule

```

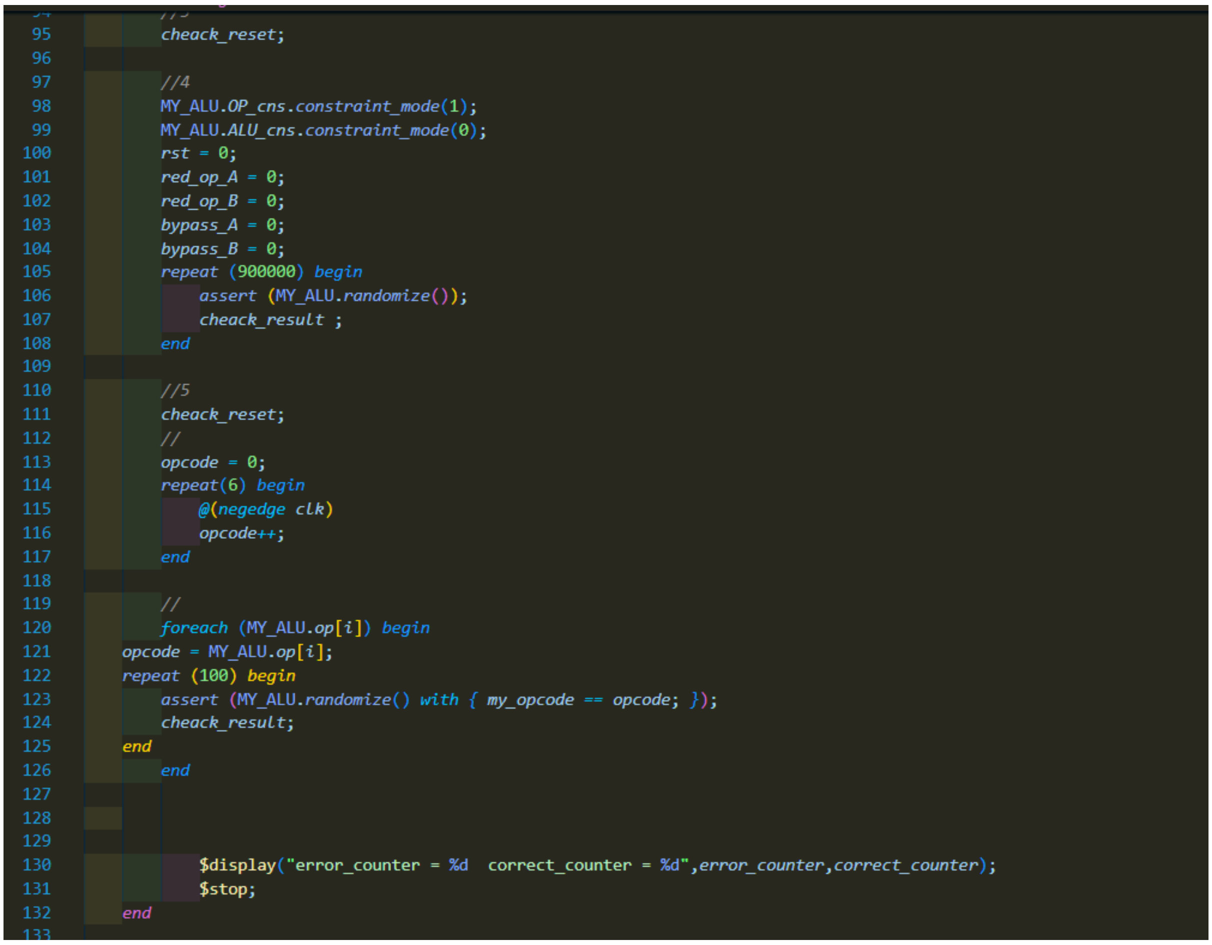
CODE TB :



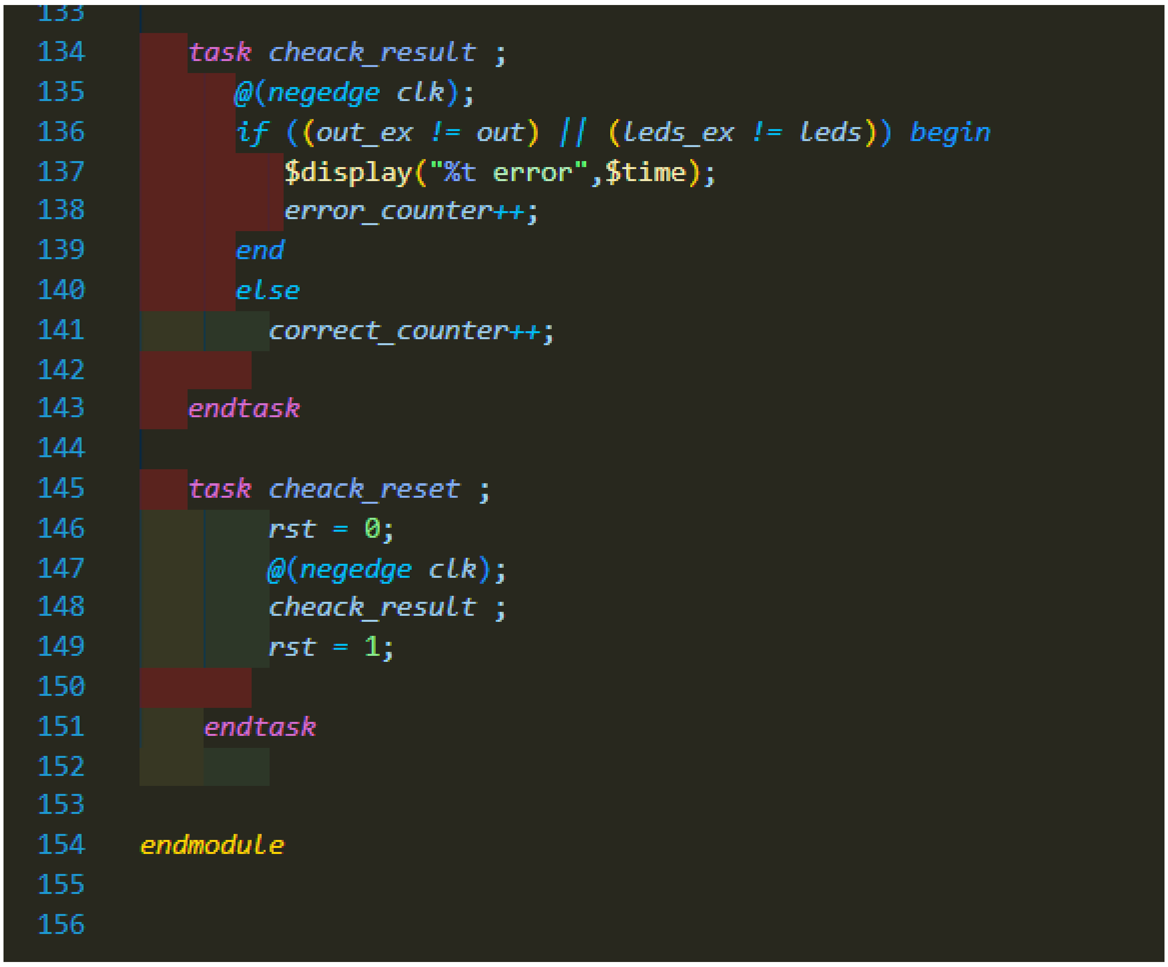
```systemverilog

import ALSU_pkg::*;
module ALSU_tb ();
    parameter INPUT_PRIORITY = "A";
    parameter FULL_ADDER = "ON";
    logic clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    logic [2:0] opcode;
    logic signed [2:0] A, B;
    logic [15:0] leds;
    bit [15:0] leds_ex;
    logic [5:0] out;
    bit [5:0] out_ex;

    int error_counter = 0;
    int correct_counter = 0;

    ALSU o1 (A,B,cin,serial_in,red_op_A,red_op_B,opcode,bypass_A,bypass_B,clk,rst,direction,leds,out);
    ALSU_golden_model O4 (out_ex,leds_ex,A,B,opcode,clk,rst,cin,serial_in,red_op_A,red_op_B,bypass_A,bypass_B,direction);


    ALU_class MY_ALU = new;

        covergroup ALSU_G @(posedge clk);

       label_A : coverpoint A {
        bins A_data_0 = {0};
        bins A_data_max = {MAXPOS};
        bins A_data_min = {MAXNEG};
        bins A_data_default = default;}

       label_A_red_op : coverpoint A iff (red_op_A) {
        bins A_data_walking_ones[] = {1,2,4};}

       label_B : coverpoint B {
        bins B_data_0 = {0};
        bins B_data_max = {MAXPOS};
        bins B_data_min = {MAXNEG};
        bins B_data_default = default;}

       label_B_red_op : coverpoint B iff (red_op_B) {
        bins B_data_walking_ones[] = {1,2,4};}

        label_opcode : coverpoint opcode {
        bins bin_shift[] = {SHIFT,ROTATE};
        bins bin_arith[] = {ADD,MULT};
        bins bin_bitwise[] = {OR,XOR};
        illegal_bins bin_invalid[] = {invalid6,invalid7};
        bins opcode_tr = (0 => 1 => 2 => 3 => 4 => 5);}
```



```
    ALSU_G my_alu_g;

    initial begin
      my_alu_g = new;
      end

     always @(posedge clk) begin
           if (!rst && !bypass_A && !bypass_B) begin
           my_alu_g.sample();
      end
     end


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

initial begin
    //1
    cheack_reset;

    //2
    MY_ALU.OP_cns.constraint_mode(0);
    repeat (90000) begin
        assert (MY_ALU.randomize());
        rst = MY_ALU.rst;
        red_op_A = MY_ALU.red_op_A;
        red_op_B = MY_ALU.red_op_B;
        bypass_A = MY_ALU.bypass_A;
        bypass_B = MY_ALU.bypass_B;
        direction = MY_ALU.direction;
        cin = MY_ALU.cin;
        serial_in = MY_ALU.serial;
        A = MY_ALU.A;
        B = MY_ALU.B;
        opcode = MY_ALU.my_opcode;
        cheack_result;
    end

    //3
    cheack_reset;
```

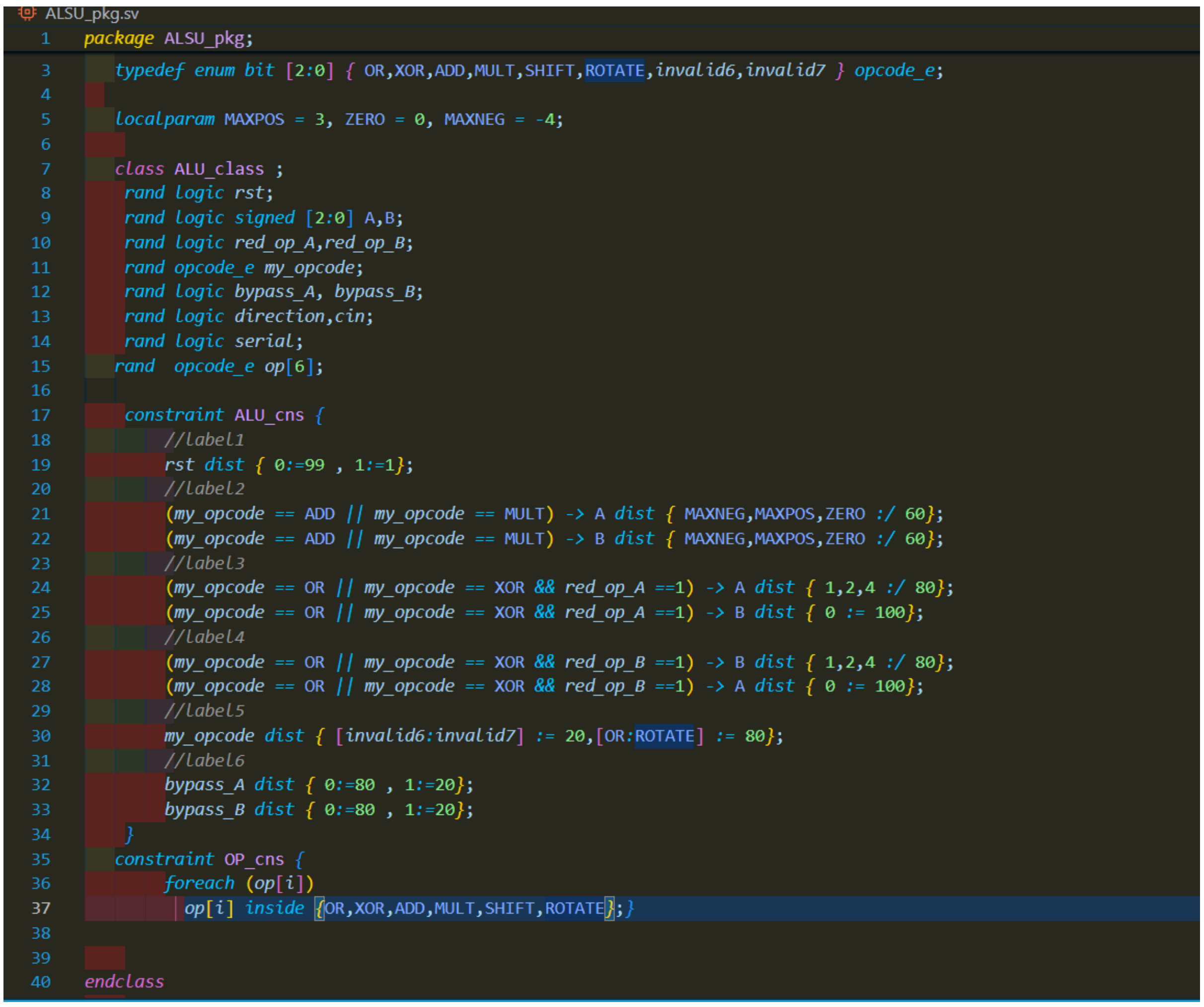


```
        cheack_reset;

        //4
        MY_ALU.OP_cns.constraint_mode(1);
        MY_ALU.ALU_cns.constraint_mode(0);
        rst = 0;
        red_op_A = 0;
        red_op_B = 0;
        bypass_A = 0;
        bypass_B = 0;
        repeat (900000) begin
            assert (MY_ALU.randomize());
            cheack_result ;
        end

        //5
        cheack_reset;
        //
        opcode = 0;
        repeat(6) begin
            @(negedge clk)
            opcode++;
        end

        //
        foreach (MY_ALU.op[i]) begin
    opcode = MY_ALU.op[i];
    repeat (100) begin
        assert (MY_ALU.randomize() with { my_opcode == opcode; });
        cheack_result;
    end
        end



            $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
            $stop;
    end
```



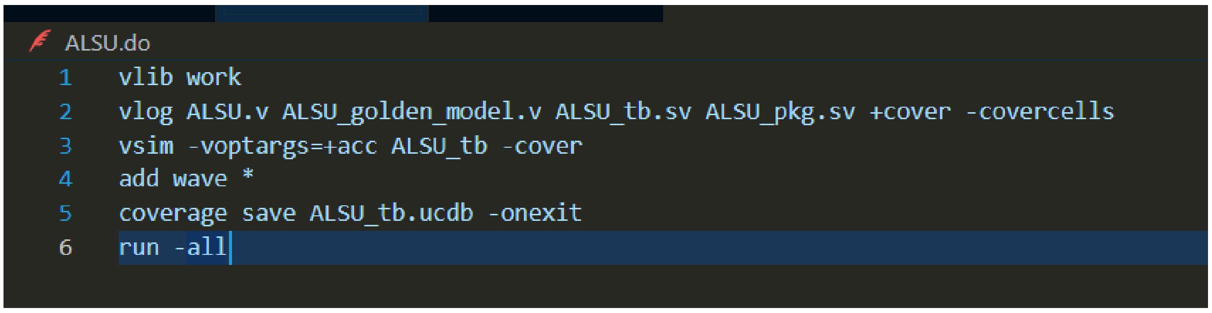
```
  task cheack_result ;
    @(negedge clk);
    if ((out_ex != out) || (leds_ex != leds)) begin
      $display("%t error",$time);
      error_counter++;
    end
    else
      correct_counter++;

  endtask

  task cheack_reset ;
      rst = 0;
      @(negedge clk);
      cheack_result ;
      rst = 1;

    endtask


endmodule


```

CODE PKG :



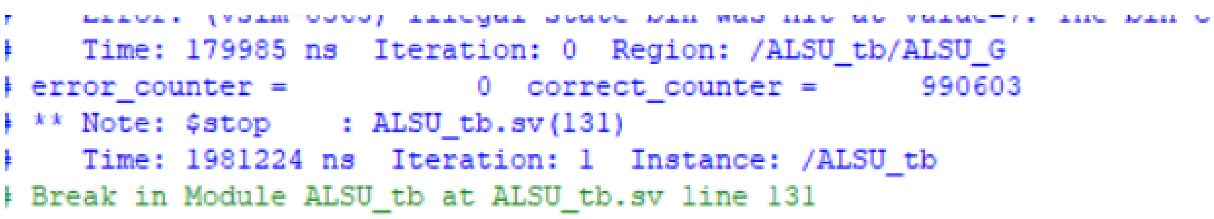
```
ALSU_pkg.sv
package ALSU_pkg;

  typedef enum bit [2:0] { OR,XOR,ADD,MULT,SHIFT,ROTATE,invalid6,invalid7 } opcode_e;

  localparam MAXPOS = 3, ZERO = 0, MAXNEG = -4;

  class ALU_class ;
    rand logic rst;
    rand logic signed [2:0] A,B;
    rand logic red_op_A,red_op_B;
    rand opcode_e my_opcode;
    rand logic bypass_A, bypass_B;
    rand logic direction,cin;
    rand logic serial;
  rand  opcode_e op[6];

    constraint ALU_cns {
        //label1
        rst dist { 0:=99 , 1:=1};
        //label2
        (my_opcode == ADD || my_opcode == MULT) -> A dist { MAXNEG,MAXPOS,ZERO :/ 60};
        (my_opcode == ADD || my_opcode == MULT) -> B dist { MAXNEG,MAXPOS,ZERO :/ 60};
        //label3
        (my_opcode == OR || my_opcode == XOR && red_op_A ==1) -> A dist { 1,2,4 :/ 80};
        (my_opcode == OR || my_opcode == XOR && red_op_A ==1) -> B dist { 0 := 100};
        //label4
        (my_opcode == OR || my_opcode == XOR && red_op_B ==1) -> B dist { 1,2,4 :/ 80};
        (my_opcode == OR || my_opcode == XOR && red_op_B ==1) -> A dist { 0 := 100};
        //label5
        my_opcode dist { [invalid6:invalid7] := 20,[OR:ROTATE] := 80};
        //label6
        bypass_A dist { 0:=80 , 1:=20};
        bypass_B dist { 0:=80 , 1:=20};
    }
  constraint OP_cns {
        foreach (op[i])
          op[i] inside {OR,XOR,ADD,MULT,SHIFT,ROTATE};}


endclass
```

DO FILE :



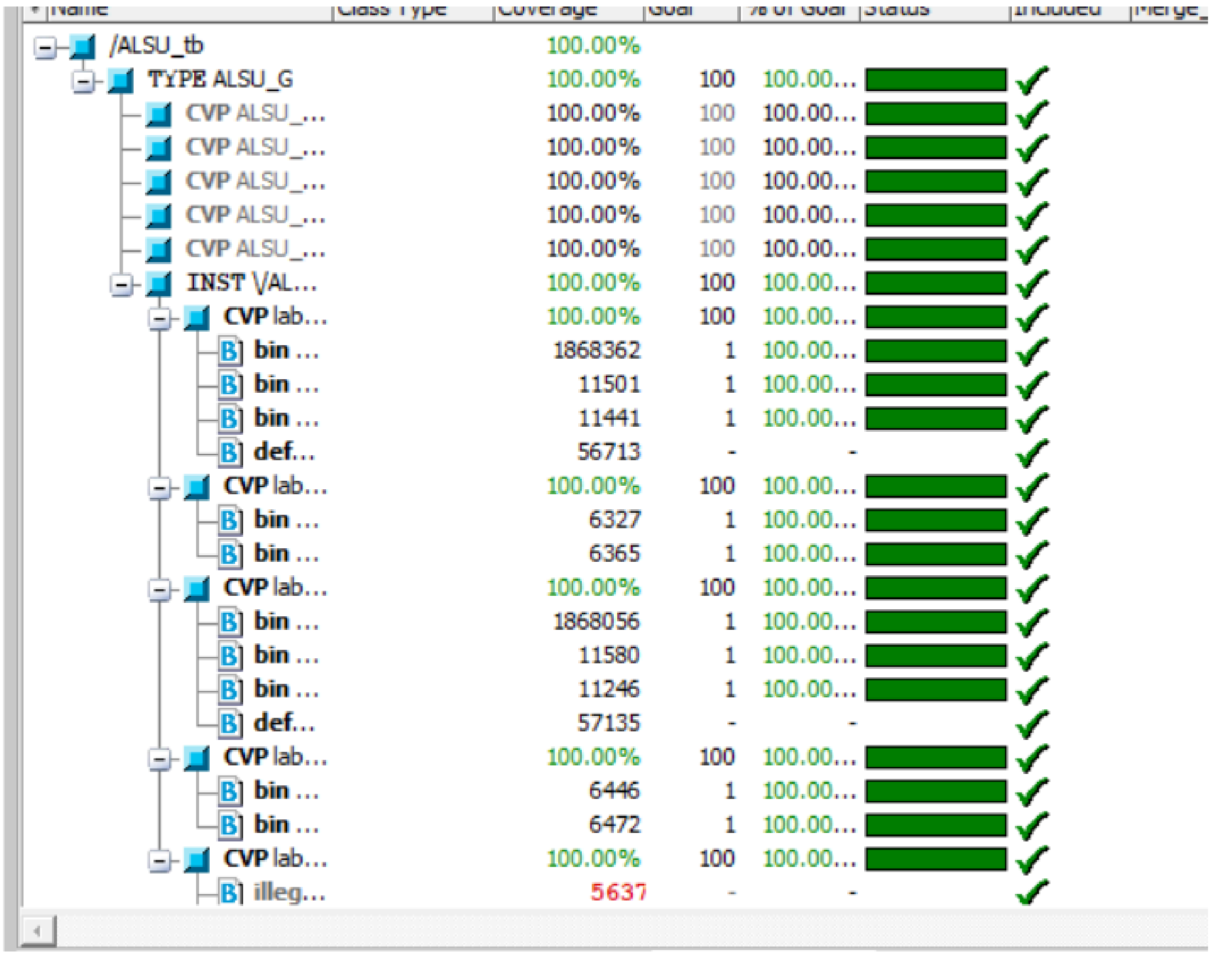
```
ALSU.do
vlib work
vlog ALSU.v ALSU_golden_model.v ALSU_tb.sv ALSU_pkg.sv +cover -covercells
vsim -voptargs=+acc ALSU_tb -cover
add wave *
coverage save ALSU_tb.ucdb -onexit
run -all
```

DISPLAY :



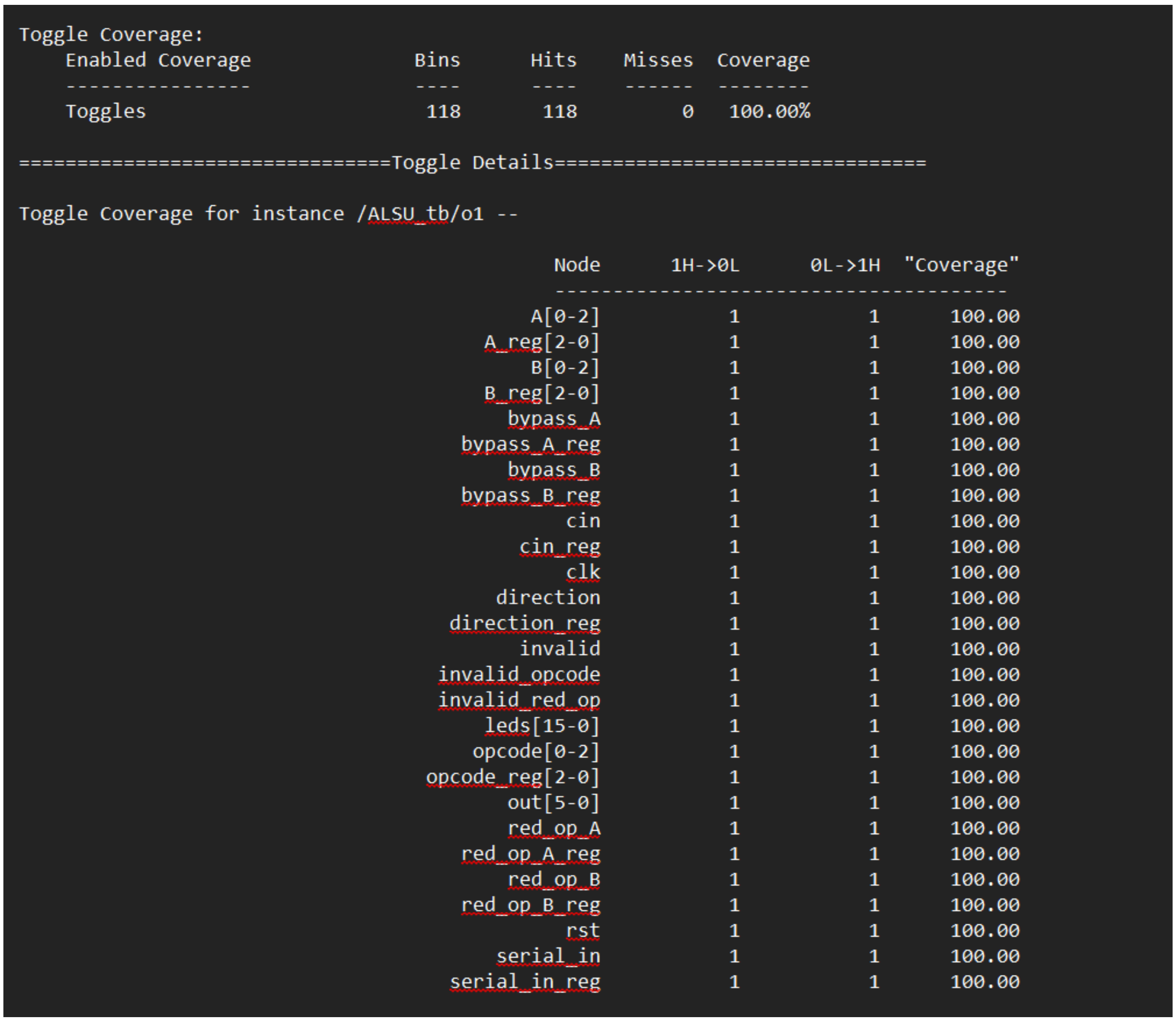
```
# ** Error: (vsim-8565) Illegal state bin was hit at value=7. The bin c
#    Time: 179985 ns  Iteration: 0  Region: /ALSU_tb/ALSU_G
# error_counter =           0  correct_counter =      990603
# ** Note: $stop    : ALSU_tb.sv(131)
#    Time: 1981224 ns  Iteration: 1  Instance: /ALSU_tb
# Break in Module ALSU_tb at ALSU_tb.sv line 131
```

COVERGROUP :



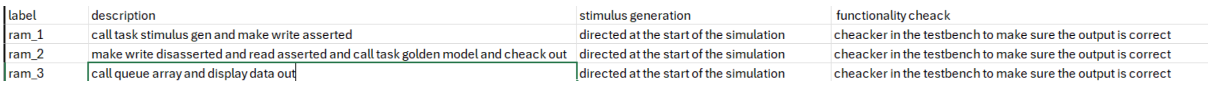
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_
/ALSU_tb		100.00%					
TYPE ALSU_G		100.00%	100	100.00...	🟩	✓	
CVP ALSU_...		100.00%	100	100.00...	🟩	✓	
CVP ALSU_...		100.00%	100	100.00...	🟩	✓	
CVP ALSU_...		100.00%	100	100.00...	🟩	✓	
CVP ALSU_...		100.00%	100	100.00...	🟩	✓	
CVP ALSU_...		100.00%	100	100.00...	🟩	✓	
INST VAL...		100.00%	100	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		1868362	1	100.00...	🟩	✓	
bin ...		11501	1	100.00...	🟩	✓	
bin ...		11441	1	100.00...	🟩	✓	
def...		56713	-	-		✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		6327	1	100.00...	🟩	✓	
bin ...		6365	1	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		1868056	1	100.00...	🟩	✓	
bin ...		11580	1	100.00...	🟩	✓	
bin ...		11246	1	100.00...	🟩	✓	
def...		57135	-	-		✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
bin ...		6446	1	100.00...	🟩	✓	
bin ...		6472	1	100.00...	🟩	✓	
CVP lab...		100.00%	100	100.00...	🟩	✓	
illeg...		5637	-	-		✓	

CODE COVERAGE :


```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        118       118         0   100.00%

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

Toggle Coverage for instance /ALSU_tb/o1 --

                                              Node      1H->0L      0L->1H  "Coverage"
                                              ---------------------------------------
                                            A[0-2]           1           1      100.00
                                        A_reg[2-0]           1           1      100.00
                                            B[0-2]           1           1      100.00
                                        B_reg[2-0]           1           1      100.00
                                          bypass_A           1           1      100.00
                                      bypass_A_reg           1           1      100.00
                                          bypass_B           1           1      100.00
                                      bypass_B_reg           1           1      100.00
                                               cin           1           1      100.00
                                           cin_reg           1           1      100.00
                                               clk           1           1      100.00
                                         direction           1           1      100.00
                                     direction_reg           1           1      100.00
                                           invalid           1           1      100.00
                                    invalid_opcode           1           1      100.00
                                    invalid_red_op           1           1      100.00
                                        leds[15-0]           1           1      100.00
                                       opcode[0-2]           1           1      100.00
                                   opcode_reg[2-0]           1           1      100.00
                                          out[5-0]           1           1      100.00
                                          red_op_A           1           1      100.00
                                      red_op_A_reg           1           1      100.00
                                          red_op_B           1           1      100.00
                                      red_op_B_reg           1           1      100.00
                                               rst           1           1      100.00
                                         serial_in           1           1      100.00
                                     serial_in_reg           1           1      100.00
```

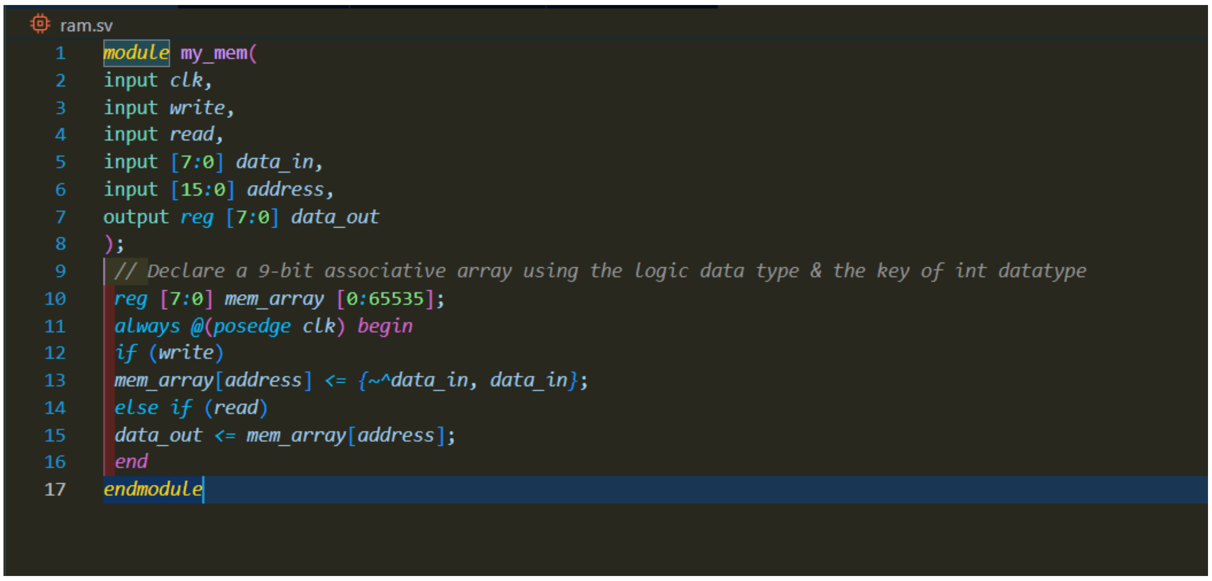
Q4

Verification plan :



label	description	stimulus generation	functionality cheack
ram_1	call task stimulus gen and make write asserted	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct
ram_2	make write disasserted and read asserted and call task golden model and cheack out	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct
ram_3	call queue array and display data out	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct

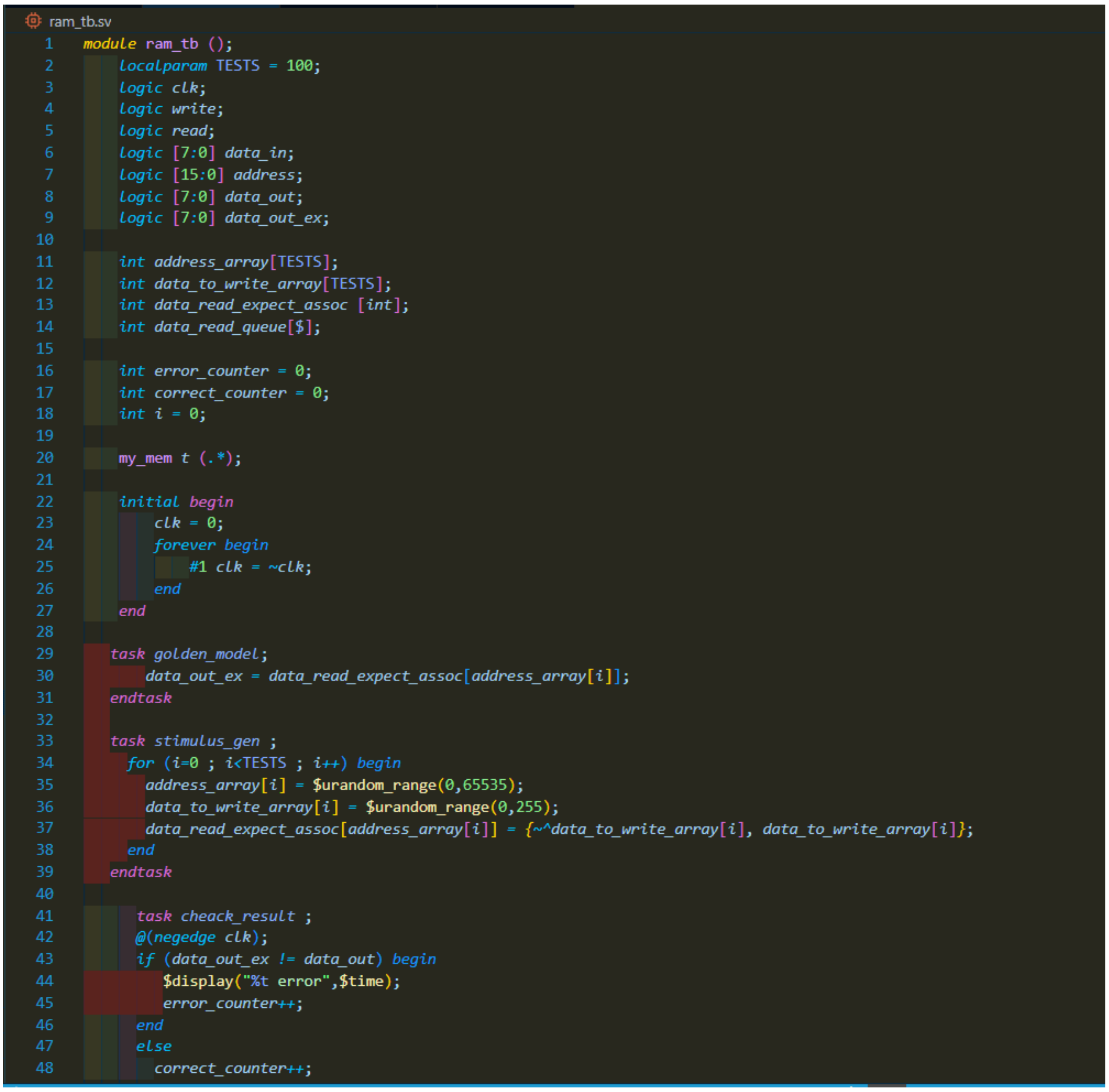
DESIGN :



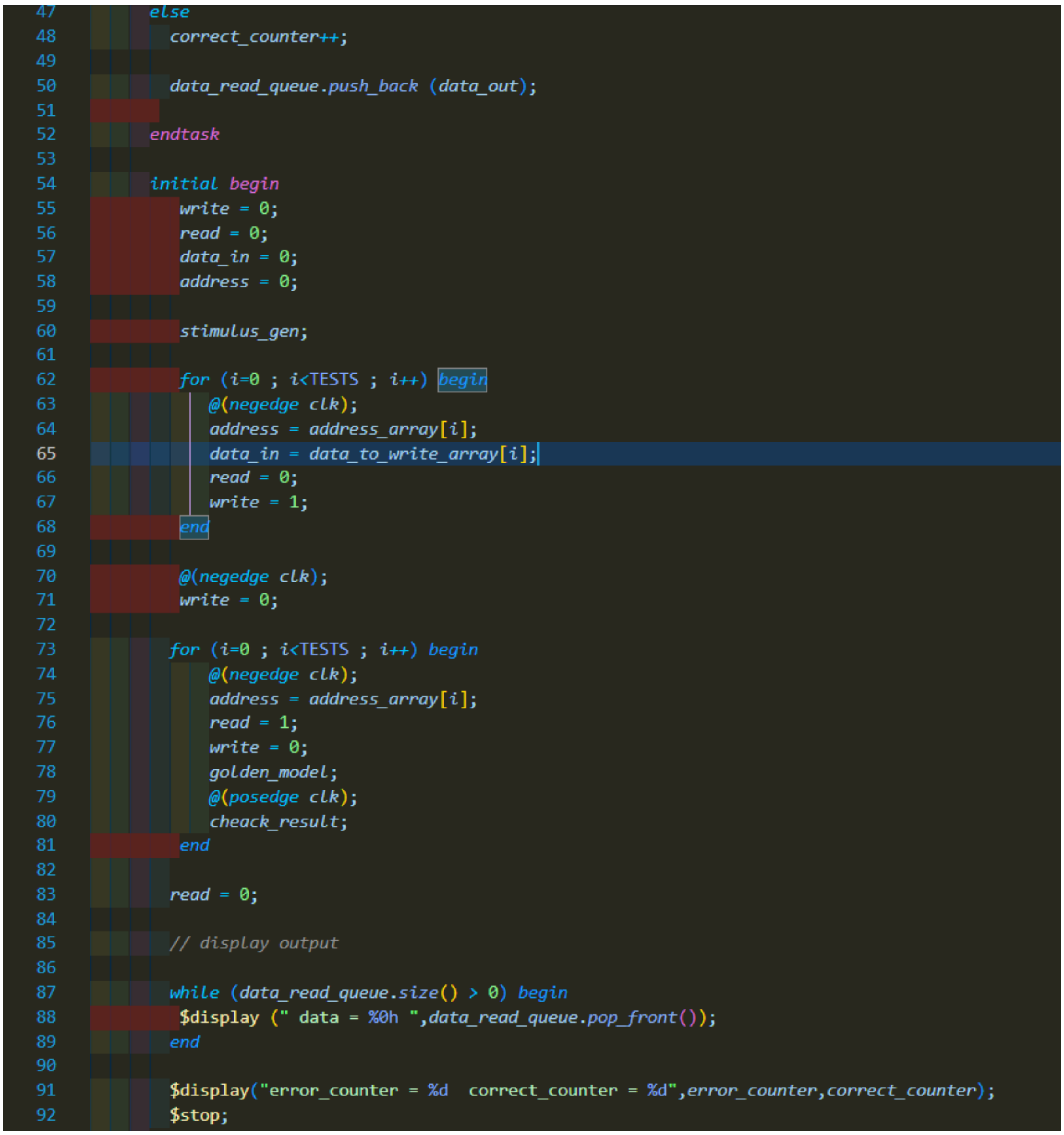
ram.sv

```
module my_mem(
input clk,
input write,
input read,
input [7:0] data_in,
input [15:0] address,
output reg [7:0] data_out
);
 // Declare a 9-bit associative array using the logic data type & the key of int datatype
 reg [7:0] mem_array [0:65535];
 always @(posedge clk) begin
 if (write)
 mem_array[address] <= {~^data_in, data_in};
 else if (read)
 data_out <= mem_array[address];
 end
endmodule
```

CODE TB :

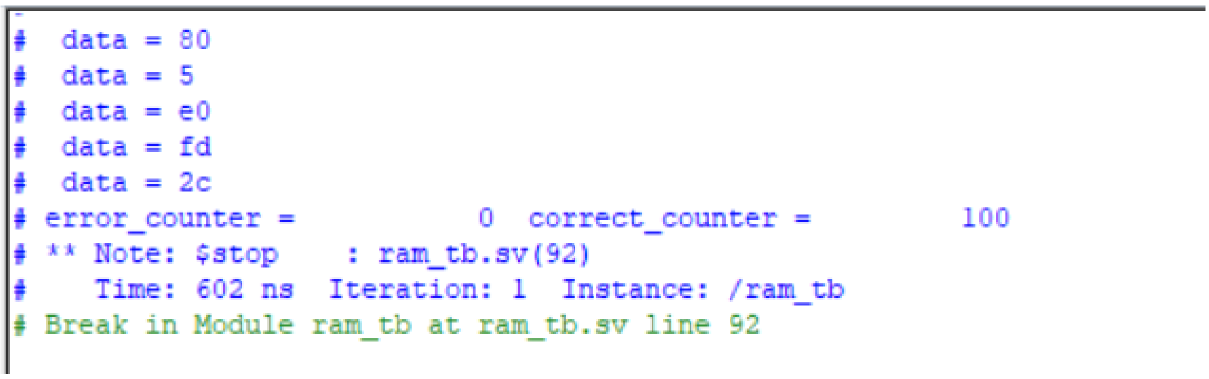

ram_tb.sv

```
module ram_tb ();
    localparam TESTS = 100;
    logic clk;
    logic write;
    logic read;
    logic [7:0] data_in;
    logic [15:0] address;
    logic [7:0] data_out;
    logic [7:0] data_out_ex;

    int address_array[TESTS];
    int data_to_write_array[TESTS];
    int data_read_expect_assoc [int];
    int data_read_queue[$];

    int error_counter = 0;
    int correct_counter = 0;
    int i = 0;

    my_mem t (.*);

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

   task golden_model;
       data_out_ex = data_read_expect_assoc[address_array[i]];
   endtask

   task stimulus_gen ;
     for (i=0 ; i<TESTS ; i++) begin
       address_array[i] = $urandom_range(0,65535);
       data_to_write_array[i] = $urandom_range(0,255);
       data_read_expect_assoc[address_array[i]] = {~^data_to_write_array[i], data_to_write_array[i]};
     end
   endtask

      task cheack_result ;
      @(negedge clk);
      if (data_out_ex != data_out) begin
         $display("%t error",$time);
         error_counter++;
      end
      else
        correct_counter++;
```



```
47      else
48        correct_counter++;
49
50        data_read_queue.push_back (data_out);
51
52      endtask
53
54      initial begin
55         write = 0;
56         read = 0;
57         data_in = 0;
58         address = 0;
59
60         stimulus_gen;
61
62         for (i=0 ; i<TESTS ; i++) begin
63            @(negedge clk);
64            address = address_array[i];
65            data_in = data_to_write_array[i];
66            read = 0;
67            write = 1;
68         end
69
70         @(negedge clk);
71         write = 0;
72
73        for (i=0 ; i<TESTS ; i++) begin
74            @(negedge clk);
75            address = address_array[i];
76            read = 1;
77            write = 0;
78            golden_model;
79            @(posedge clk);
80            cheack_result;
81         end
82
83        read = 0;
84
85        // display output
86
87        while (data_read_queue.size() > 0) begin
88         $display (" data = %0h ",data_read_queue.pop_front());
89        end
90
91        $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
92        $stop;
```

DISPLAY :



```
#   data = 80
#   data = 5
#   data = e0
#   data = fd
#   data = 2c
# error_counter =           0   correct_counter =         100
# ** Note: $stop    : ram_tb.sv(92)
#    Time: 602 ns  Iteration: 1  Instance: /ram_tb
# Break in Module ram_tb at ram_tb.sv line 92
```