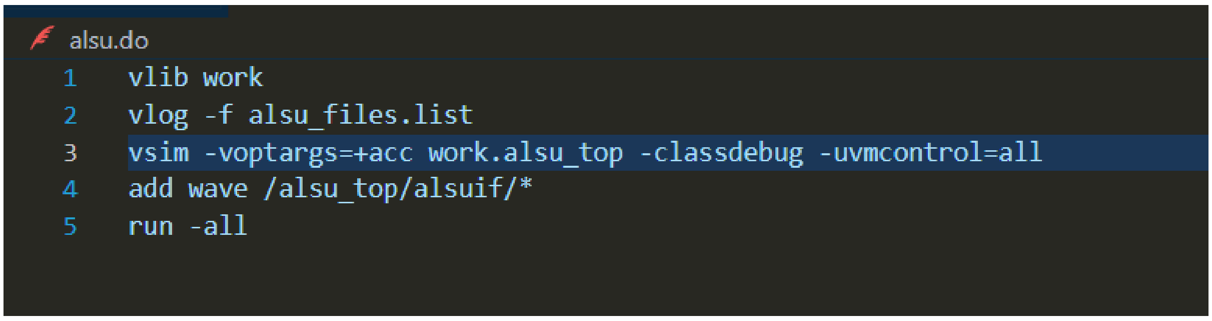
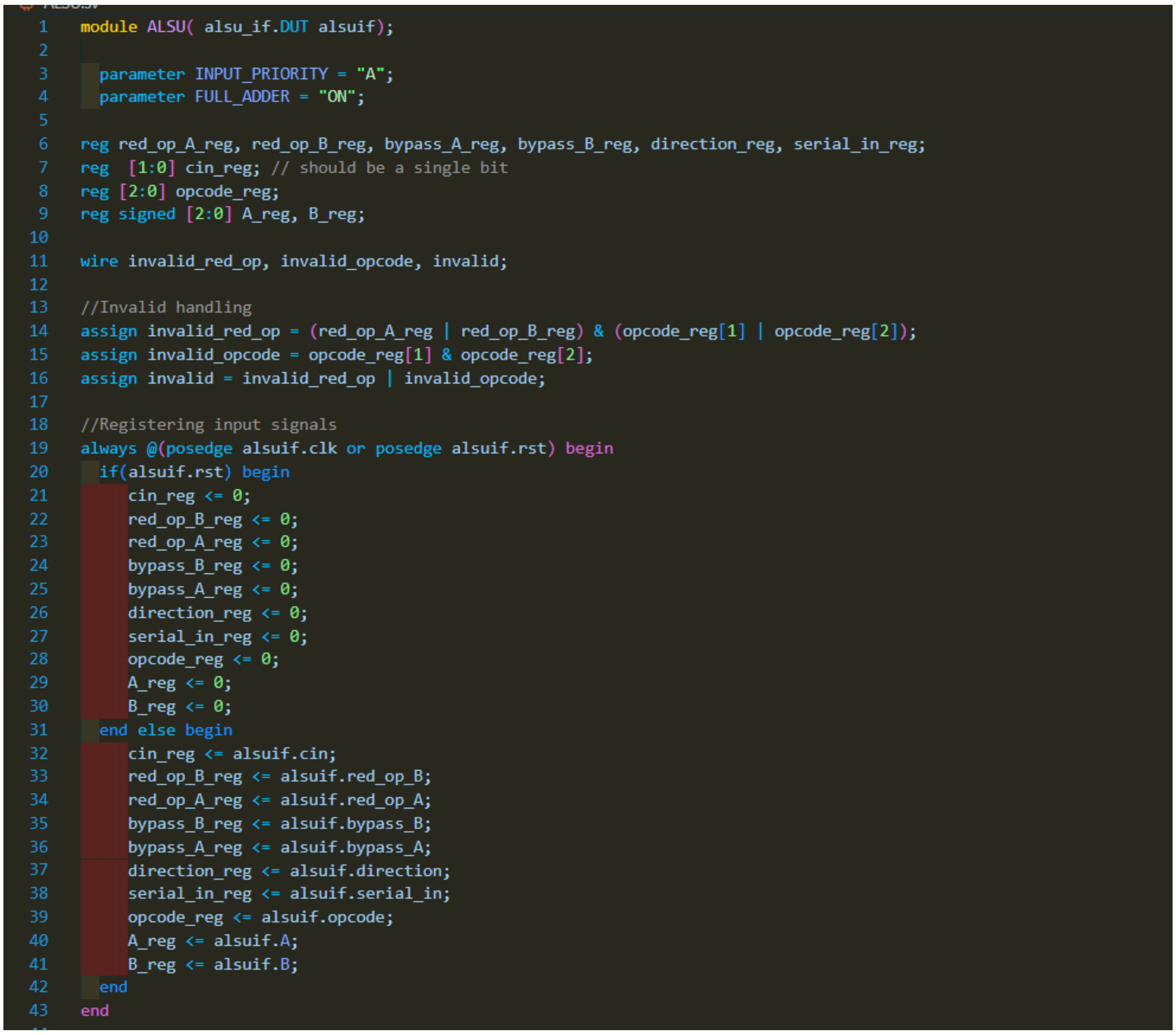
Assignment 6
Verification For ALSU Using UVM

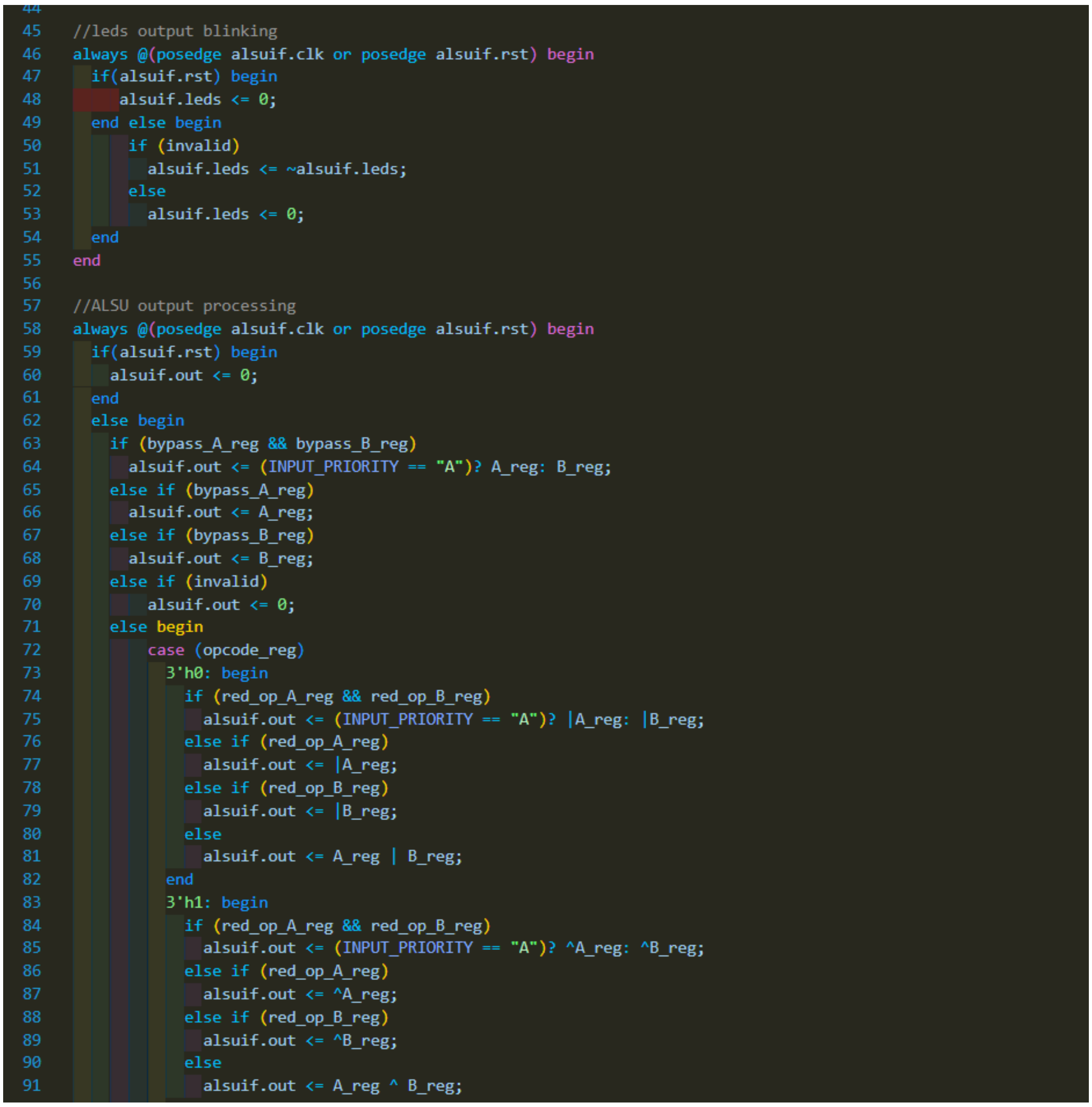


alsu.do

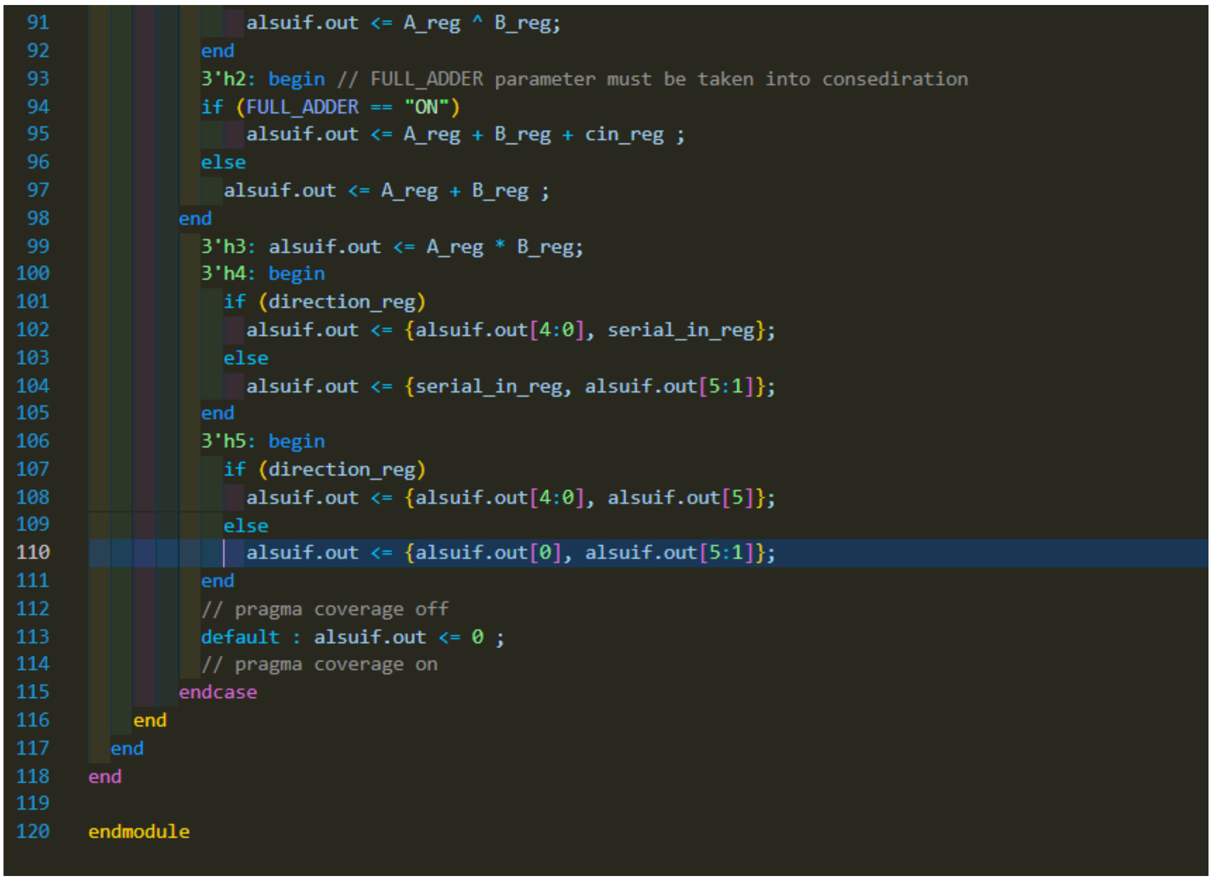
```
vlib work
vlog -f alsu_files.list
vsim -voptargs=+acc work.alsu_top -classdebug -uvmcontrol=all
add wave /alsu_top/alsuif/*
run -all
```



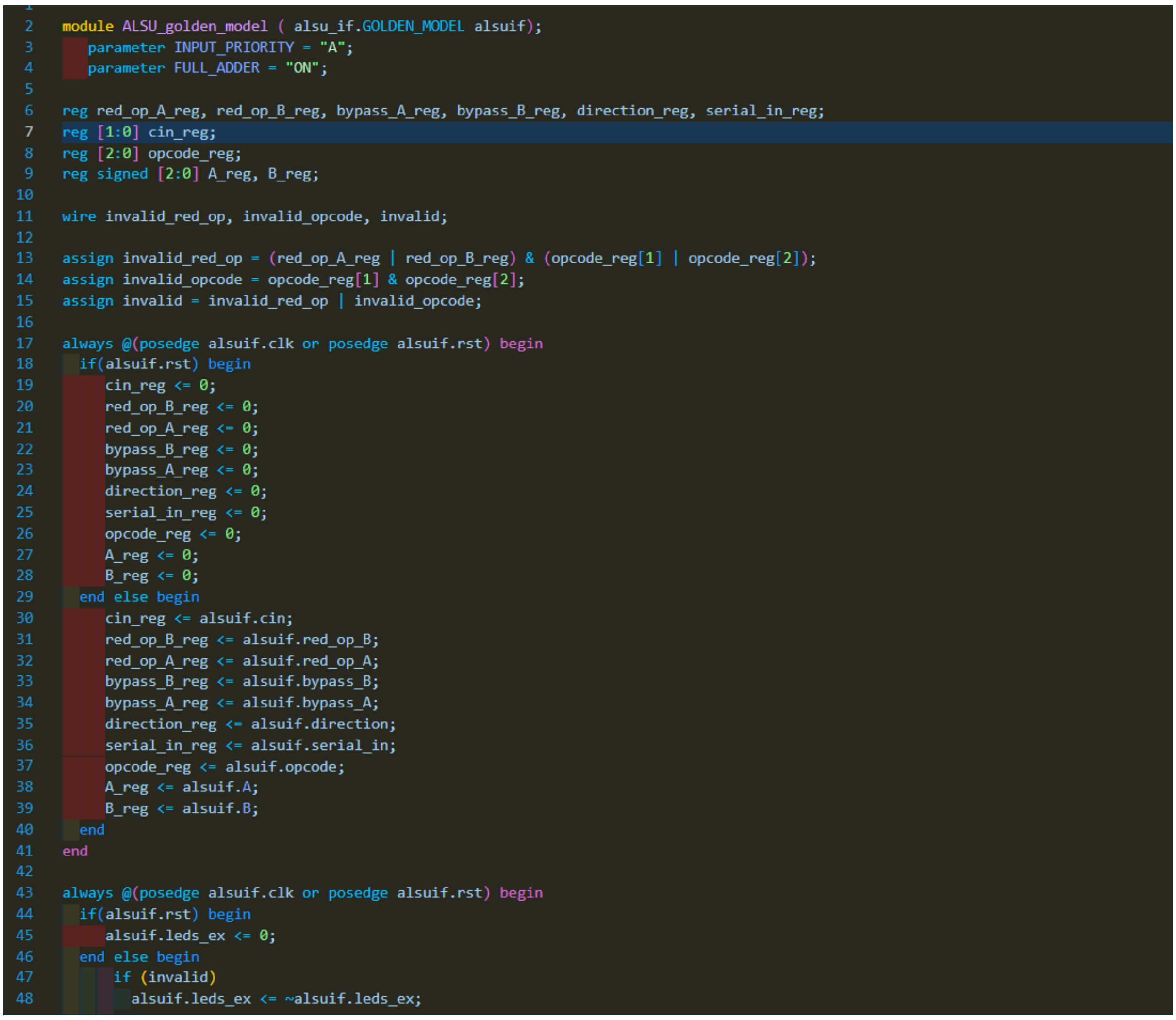
```
module ALSU( alsu_if.DUT alsuif);

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

reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg  [1:0] cin_reg; // should be a single bit
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 alsuif.clk or posedge alsuif.rst) begin
  if(alsuif.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 <= alsuif.cin;
     red_op_B_reg <= alsuif.red_op_B;
     red_op_A_reg <= alsuif.red_op_A;
     bypass_B_reg <= alsuif.bypass_B;
     bypass_A_reg <= alsuif.bypass_A;
     direction_reg <= alsuif.direction;
     serial_in_reg <= alsuif.serial_in;
     opcode_reg <= alsuif.opcode;
     A_reg <= alsuif.A;
     B_reg <= alsuif.B;
  end
end
```



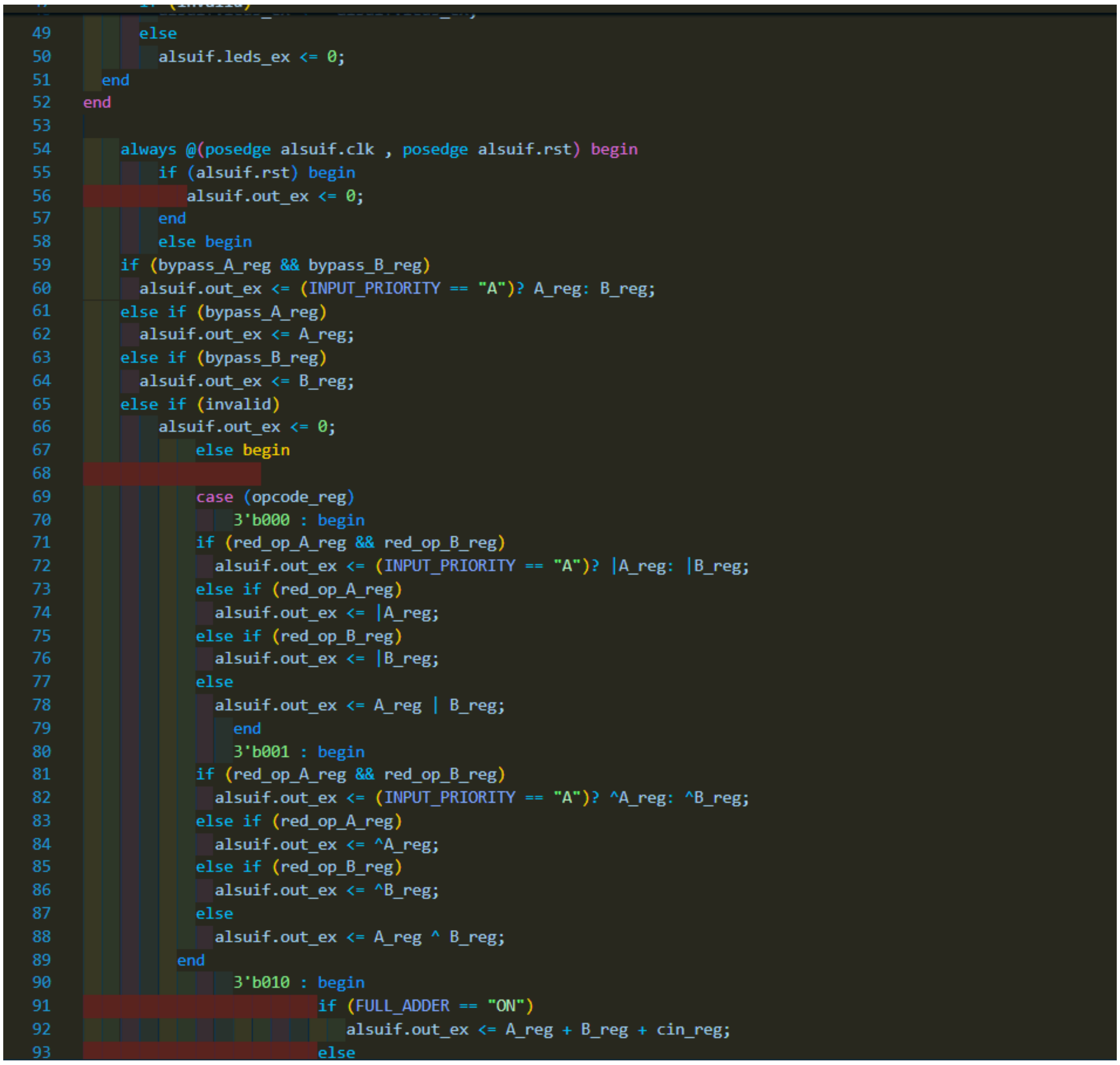
```
//leds output blinking
always @(posedge alsuif.clk or posedge alsuif.rst) begin
  if(alsuif.rst) begin
     alsuif.leds <= 0;
  end else begin
      if (invalid)
        alsuif.leds <= ~alsuif.leds;
      else
        alsuif.leds <= 0;
  end
end

//ALSU output processing
always @(posedge alsuif.clk or posedge alsuif.rst) begin
  if(alsuif.rst) begin
    alsuif.out <= 0;
  end
  else begin
    if (bypass_A_reg && bypass_B_reg)
      alsuif.out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    else if (bypass_A_reg)
      alsuif.out <= A_reg;
    else if (bypass_B_reg)
      alsuif.out <= B_reg;
    else if (invalid)
        alsuif.out <= 0;
    else begin
        case (opcode_reg)
          3'h0: begin
            if (red_op_A_reg && red_op_B_reg)
              alsuif.out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            else if (red_op_A_reg)
              alsuif.out <= |A_reg;
            else if (red_op_B_reg)
              alsuif.out <= |B_reg;
            else
              alsuif.out <= A_reg | B_reg;
          end
          3'h1: begin
            if (red_op_A_reg && red_op_B_reg)
              alsuif.out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              alsuif.out <= ^A_reg;
            else if (red_op_B_reg)
              alsuif.out <= ^B_reg;
            else
              alsuif.out <= A_reg ^ B_reg;
```



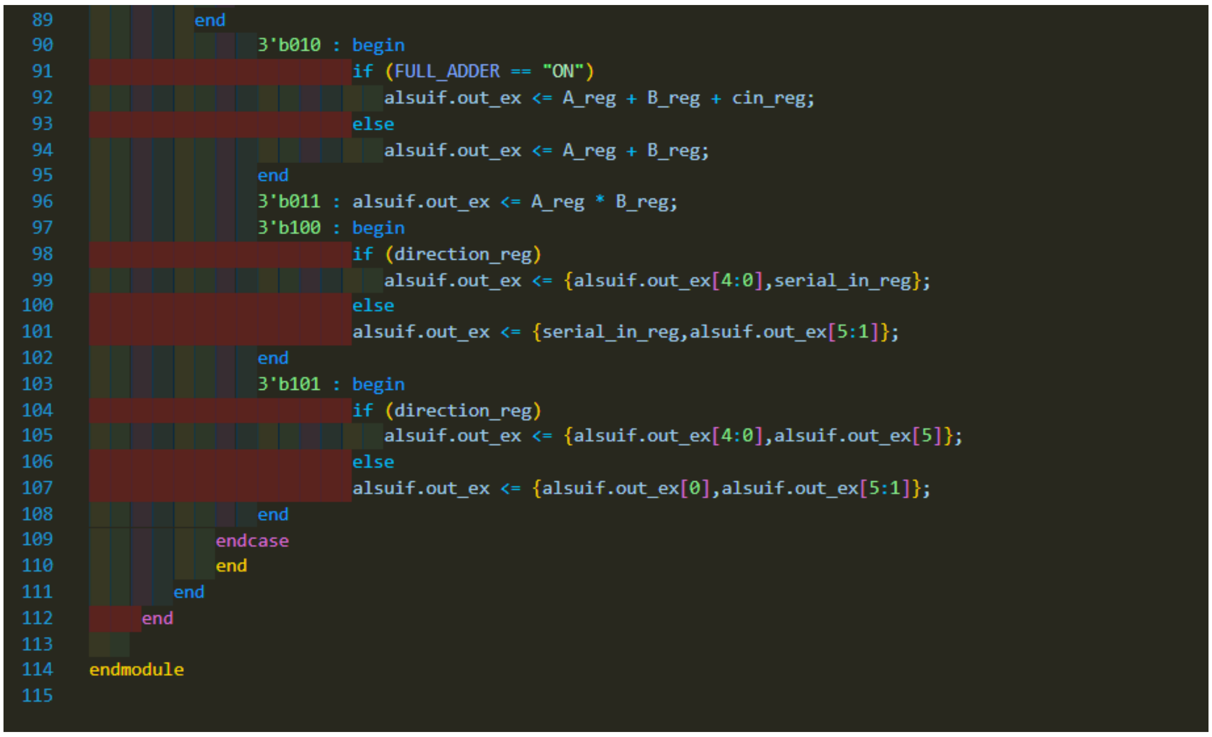
```
              alsuif.out <= A_reg ^ B_reg;
          end
          3'h2: begin // FULL_ADDER parameter must be taken into consediration
          if (FULL_ADDER == "ON")
              alsuif.out <= A_reg + B_reg + cin_reg ;
          else
            alsuif.out <= A_reg + B_reg ;
        end
          3'h3: alsuif.out <= A_reg * B_reg;
          3'h4: begin
            if (direction_reg)
              alsuif.out <= {alsuif.out[4:0], serial_in_reg};
            else
              alsuif.out <= {serial_in_reg, alsuif.out[5:1]};
          end
          3'h5: begin
            if (direction_reg)
              alsuif.out <= {alsuif.out[4:0], alsuif.out[5]};
            else
              alsuif.out <= {alsuif.out[0], alsuif.out[5:1]};
          end
          // pragma coverage off
          default : alsuif.out <= 0 ;
          // pragma coverage on
        endcase
    end
  end
end

endmodule
```



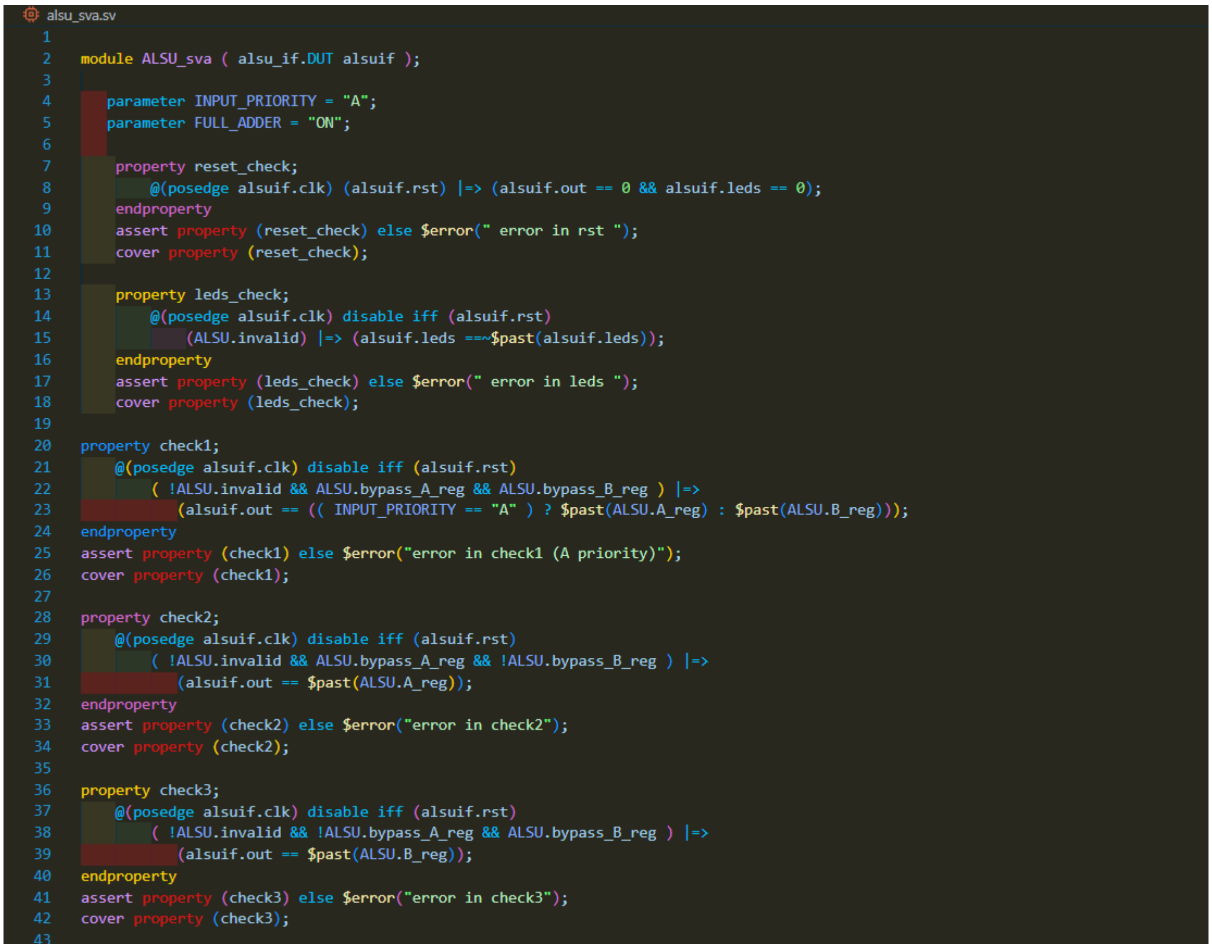
```verilog

module ALSU_golden_model ( alsu_if.GOLDEN_MODEL alsuif);
    parameter INPUT_PRIORITY = "A";
    parameter FULL_ADDER = "ON";

reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [1:0] 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 alsuif.clk or posedge alsuif.rst) begin
  if(alsuif.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 <= alsuif.cin;
     red_op_B_reg <= alsuif.red_op_B;
     red_op_A_reg <= alsuif.red_op_A;
     bypass_B_reg <= alsuif.bypass_B;
     bypass_A_reg <= alsuif.bypass_A;
     direction_reg <= alsuif.direction;
     serial_in_reg <= alsuif.serial_in;
     opcode_reg <= alsuif.opcode;
     A_reg <= alsuif.A;
     B_reg <= alsuif.B;
  end
end

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



```
      else
        alsuif.leds_ex <= 0;
  end
end

    always @(posedge alsuif.clk , posedge alsuif.rst) begin
        if (alsuif.rst) begin
           alsuif.out_ex <= 0;
        end
        else begin
    if (bypass_A_reg && bypass_B_reg)
      alsuif.out_ex <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    else if (bypass_A_reg)
      alsuif.out_ex <= A_reg;
    else if (bypass_B_reg)
      alsuif.out_ex <= B_reg;
    else if (invalid)
        alsuif.out_ex <= 0;
            else begin

            case (opcode_reg)
                3'b000 : begin
            if (red_op_A_reg && red_op_B_reg)
              alsuif.out_ex <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            else if (red_op_A_reg)
              alsuif.out_ex <= |A_reg;
            else if (red_op_B_reg)
              alsuif.out_ex <= |B_reg;
            else
              alsuif.out_ex <= A_reg | B_reg;
                end
                3'b001 : begin
            if (red_op_A_reg && red_op_B_reg)
              alsuif.out_ex <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
            else if (red_op_A_reg)
              alsuif.out_ex <= ^A_reg;
            else if (red_op_B_reg)
              alsuif.out_ex <= ^B_reg;
            else
              alsuif.out_ex <= A_reg ^ B_reg;
          end
                3'b010 : begin
                         if (FULL_ADDER == "ON")
                            alsuif.out_ex <= A_reg + B_reg + cin_reg;
                         else
```

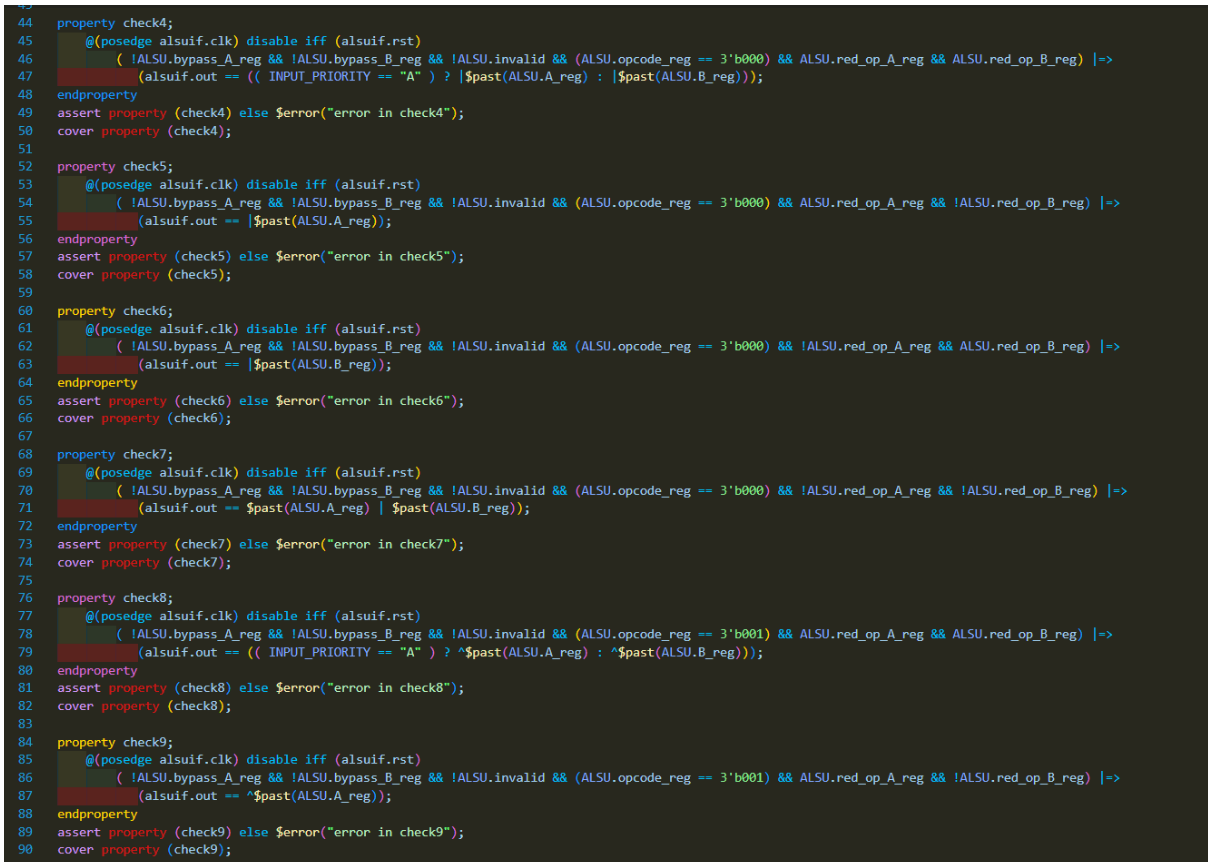


```
          end
                3'b010 : begin
                         if (FULL_ADDER == "ON")
                            alsuif.out_ex <= A_reg + B_reg + cin_reg;
                         else
                            alsuif.out_ex <= A_reg + B_reg;
                end
                3'b011 : alsuif.out_ex <= A_reg * B_reg;
                3'b100 : begin
                         if (direction_reg)
                            alsuif.out_ex <= {alsuif.out_ex[4:0],serial_in_reg};
                         else
                         alsuif.out_ex <= {serial_in_reg,alsuif.out_ex[5:1]};
                end
                3'b101 : begin
                         if (direction_reg)
                            alsuif.out_ex <= {alsuif.out_ex[4:0],alsuif.out_ex[5]};
                         else
                         alsuif.out_ex <= {alsuif.out_ex[0],alsuif.out_ex[5:1]};
                end
            endcase
            end
        end
     end

endmodule
```

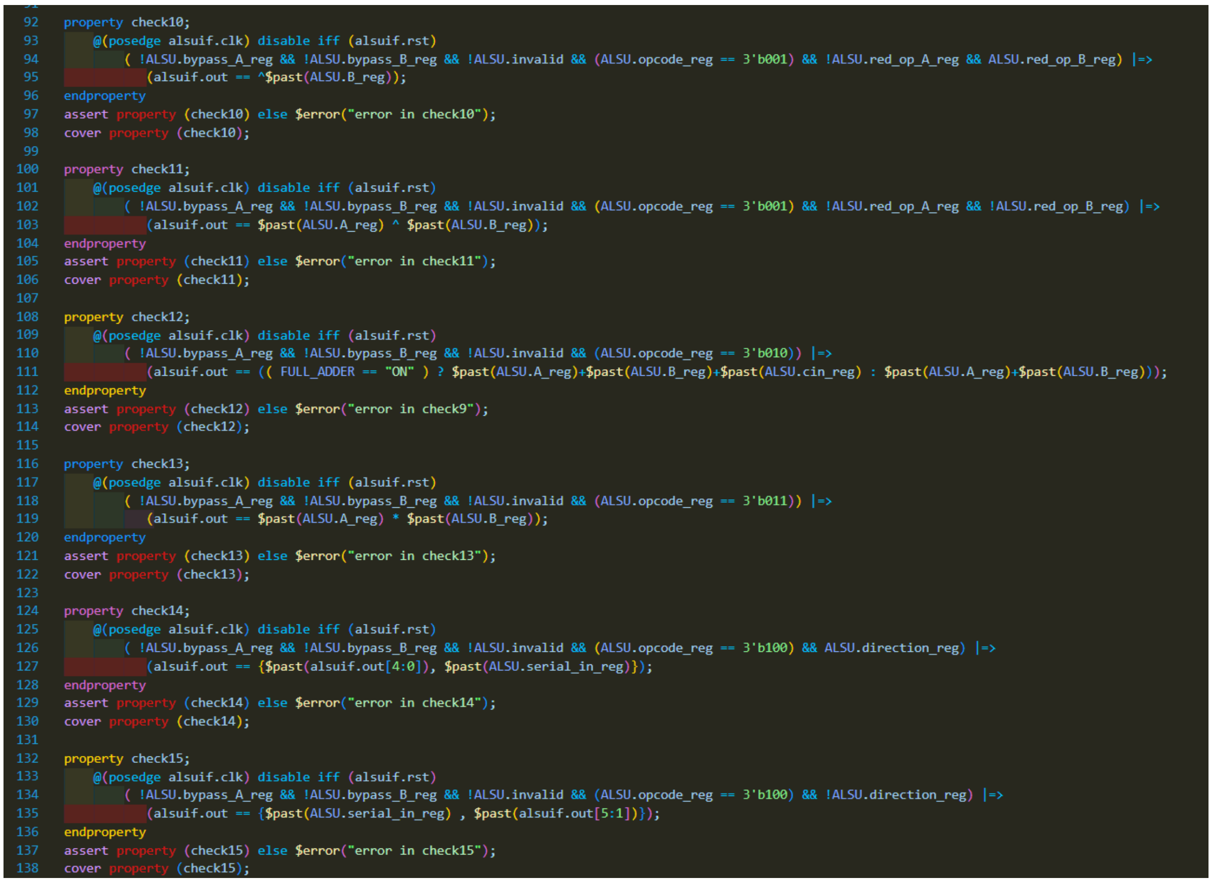


alsu_sva.sv

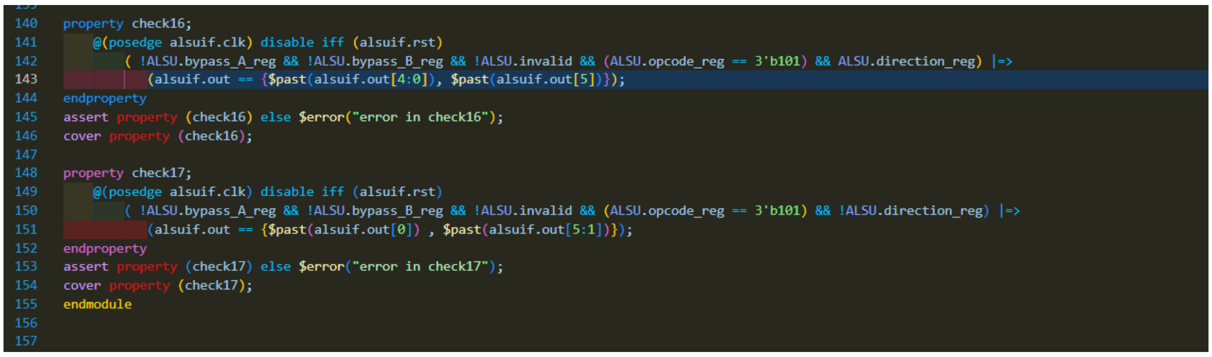
```

module ALSU_sva ( alsu_if.DUT alsuif );

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

    property reset_check;
        @(posedge alsuif.clk) (alsuif.rst) |=> (alsuif.out == 0 && alsuif.leds == 0);
    endproperty
    assert property (reset_check) else $error(" error in rst ");
    cover property (reset_check);

    property leds_check;
        @(posedge alsuif.clk) disable iff (alsuif.rst)
            (ALSU.invalid) |=> (alsuif.leds ===~$past(alsuif.leds));
    endproperty
    assert property (leds_check) else $error(" error in leds ");
    cover property (leds_check);

property check1;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.invalid && ALSU.bypass_A_reg && ALSU.bypass_B_reg ) |=>
           (alsuif.out == (( INPUT_PRIORITY == "A" ) ? $past(ALSU.A_reg) : $past(ALSU.B_reg)));
endproperty
assert property (check1) else $error("error in check1 (A priority)");
cover property (check1);

property check2;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.invalid && ALSU.bypass_A_reg && !ALSU.bypass_B_reg ) |=>
           (alsuif.out == $past(ALSU.A_reg));
endproperty
assert property (check2) else $error("error in check2");
cover property (check2);

property check3;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.invalid && !ALSU.bypass_A_reg && ALSU.bypass_B_reg ) |=>
           (alsuif.out == $past(ALSU.B_reg));
endproperty
assert property (check3) else $error("error in check3");
cover property (check3);
```



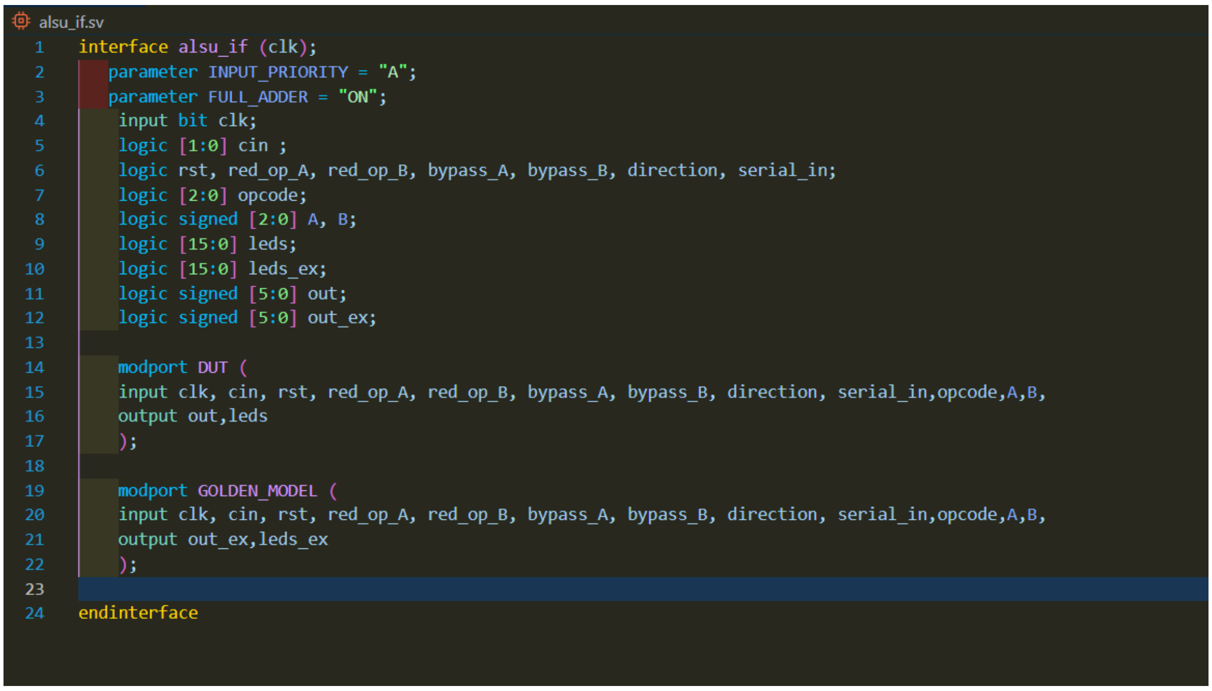
```
property check4;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b000) && ALSU.red_op_A_reg && ALSU.red_op_B_reg) |=>
            (alsuif.out == (( INPUT_PRIORITY == "A" ) ? |$past(ALSU.A_reg) : |$past(ALSU.B_reg)));
endproperty
assert property (check4) else $error("error in check4");
cover property (check4);

property check5;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b000) && ALSU.red_op_A_reg && !ALSU.red_op_B_reg) |=>
            (alsuif.out == |$past(ALSU.A_reg));
endproperty
assert property (check5) else $error("error in check5");
cover property (check5);

property check6;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b000) && !ALSU.red_op_A_reg && ALSU.red_op_B_reg) |=>
            (alsuif.out == |$past(ALSU.B_reg));
endproperty
assert property (check6) else $error("error in check6");
cover property (check6);

property check7;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b000) && !ALSU.red_op_A_reg && !ALSU.red_op_B_reg) |=>
            (alsuif.out == $past(ALSU.A_reg) | $past(ALSU.B_reg));
endproperty
assert property (check7) else $error("error in check7");
cover property (check7);

property check8;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b001) && ALSU.red_op_A_reg && ALSU.red_op_B_reg) |=>
            (alsuif.out == (( INPUT_PRIORITY == "A" ) ? ^$past(ALSU.A_reg) : ^$past(ALSU.B_reg)));
endproperty
assert property (check8) else $error("error in check8");
cover property (check8);

property check9;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b001) && ALSU.red_op_A_reg && !ALSU.red_op_B_reg) |=>
            (alsuif.out == ^$past(ALSU.A_reg));
endproperty
assert property (check9) else $error("error in check9");
cover property (check9);
```



```
property check10;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b001) && !ALSU.red_op_A_reg && ALSU.red_op_B_reg) |=>
            (alsuif.out == ^$past(ALSU.B_reg));
endproperty
assert property (check10) else $error("error in check10");
cover property (check10);

property check11;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b001) && !ALSU.red_op_A_reg && !ALSU.red_op_B_reg) |=>
            (alsuif.out == $past(ALSU.A_reg) ^ $past(ALSU.B_reg));
endproperty
assert property (check11) else $error("error in check11");
cover property (check11);

property check12;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b010)) |=>
            (alsuif.out == (( FULL_ADDER == "ON" ) ? $past(ALSU.A_reg)+$past(ALSU.B_reg)+$past(ALSU.cin_reg) : $past(ALSU.A_reg)+$past(ALSU.B_reg)));
endproperty
assert property (check12) else $error("error in check9");
cover property (check12);

property check13;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b011)) |=>
            (alsuif.out == $past(ALSU.A_reg) * $past(ALSU.B_reg));
endproperty
assert property (check13) else $error("error in check13");
cover property (check13);

property check14;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b100) && ALSU.direction_reg) |=>
            (alsuif.out == {$past(alsuif.out[4:0]), $past(ALSU.serial_in_reg)});
endproperty
assert property (check14) else $error("error in check14");
cover property (check14);

property check15;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b100) && !ALSU.direction_reg) |=>
            (alsuif.out == {$past(ALSU.serial_in_reg) , $past(alsuif.out[5:1])});
endproperty
assert property (check15) else $error("error in check15");
cover property (check15);
```

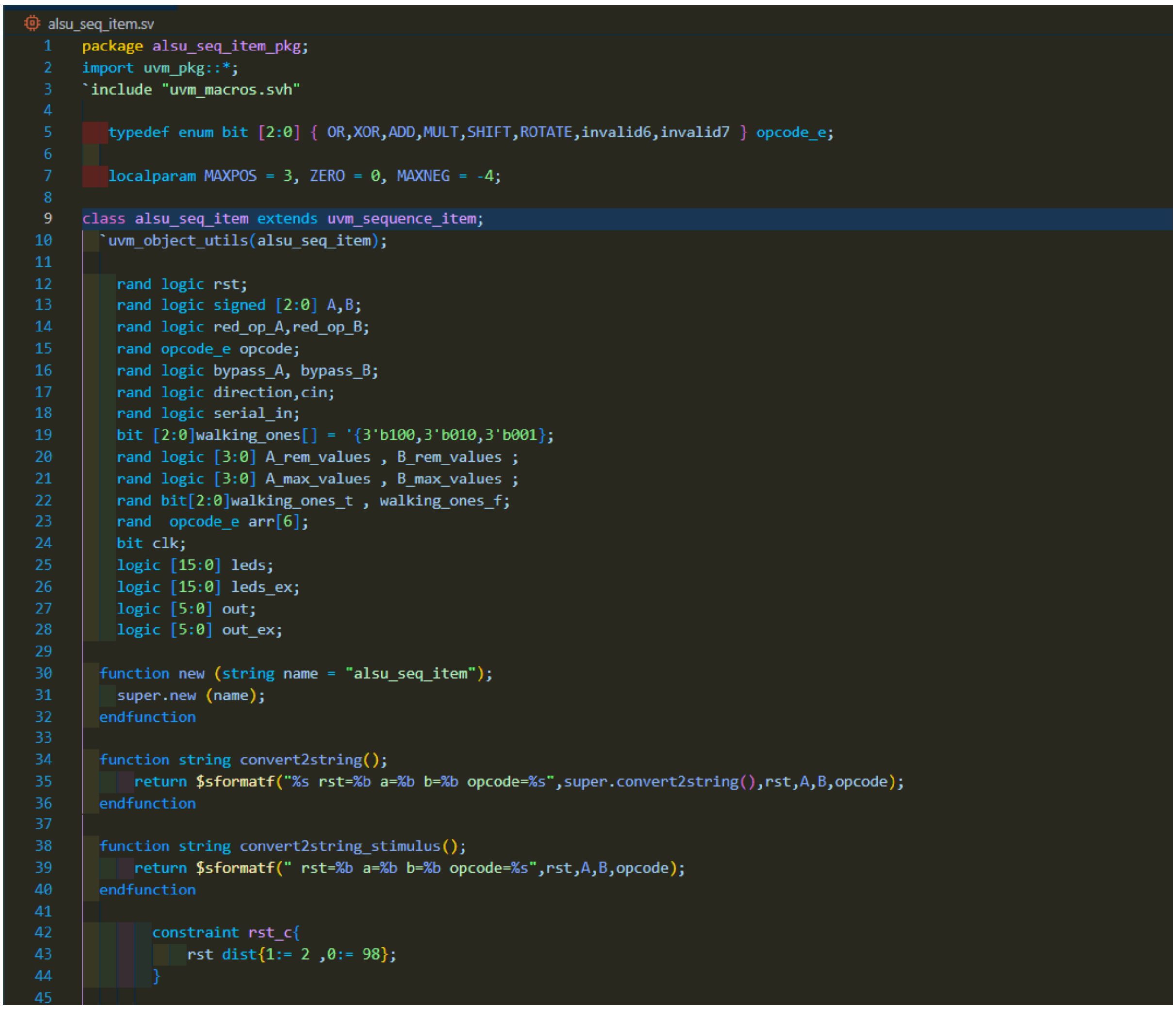


```
property check16;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b101) && ALSU.direction_reg) |=>
           (alsuif.out == {$past(alsuif.out[4:0]), $past(alsuif.out[5])});
endproperty
assert property (check16) else $error("error in check16");
cover property (check16);

property check17;
    @(posedge alsuif.clk) disable iff (alsuif.rst)
        ( !ALSU.bypass_A_reg && !ALSU.bypass_B_reg && !ALSU.invalid && (ALSU.opcode_reg == 3'b101) && !ALSU.direction_reg) |=>
           (alsuif.out == {$past(alsuif.out[0]) , $past(alsuif.out[5:1])});
endproperty
assert property (check17) else $error("error in check17");
cover property (check17);
endmodule


```

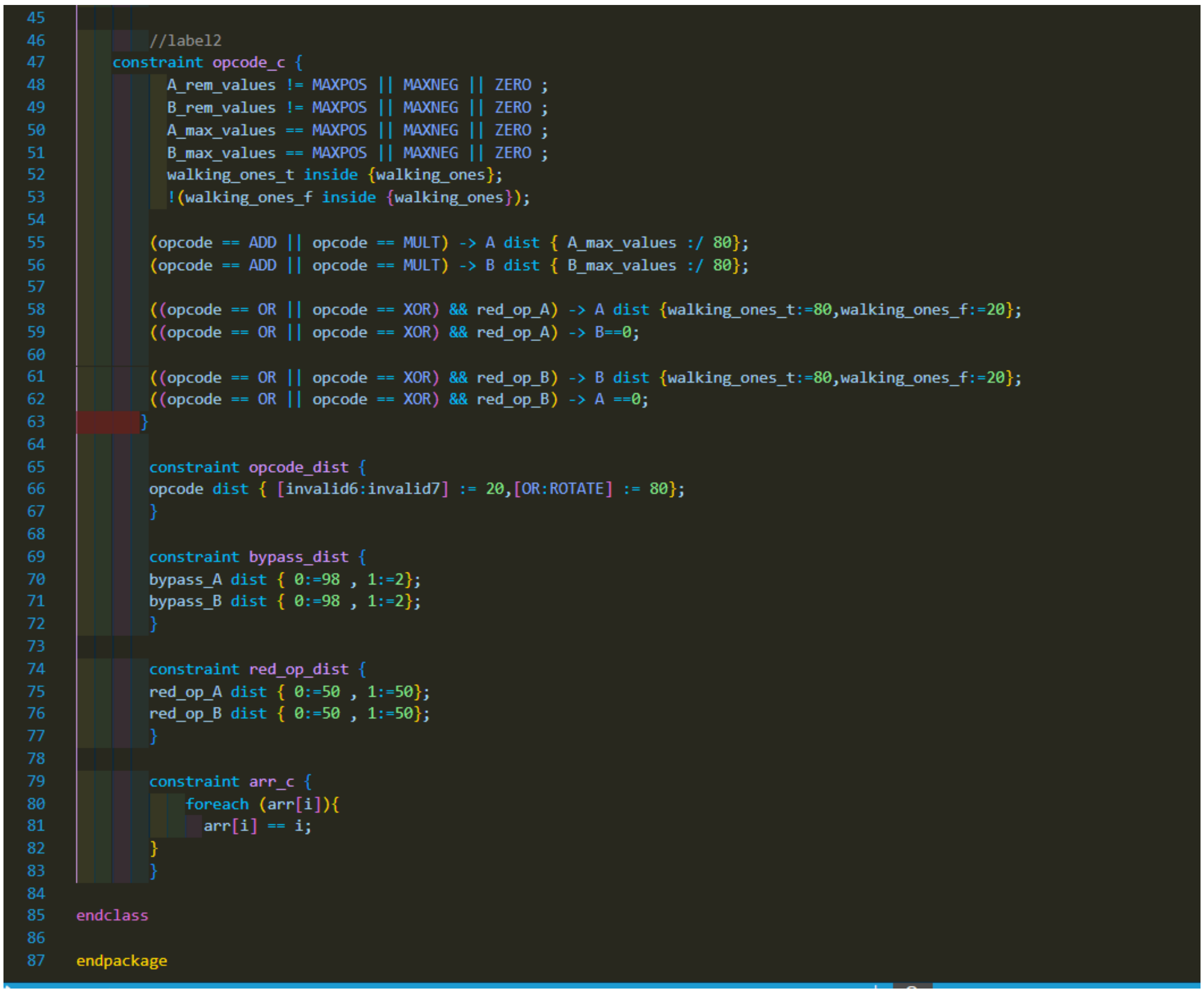


alsu_if.sv

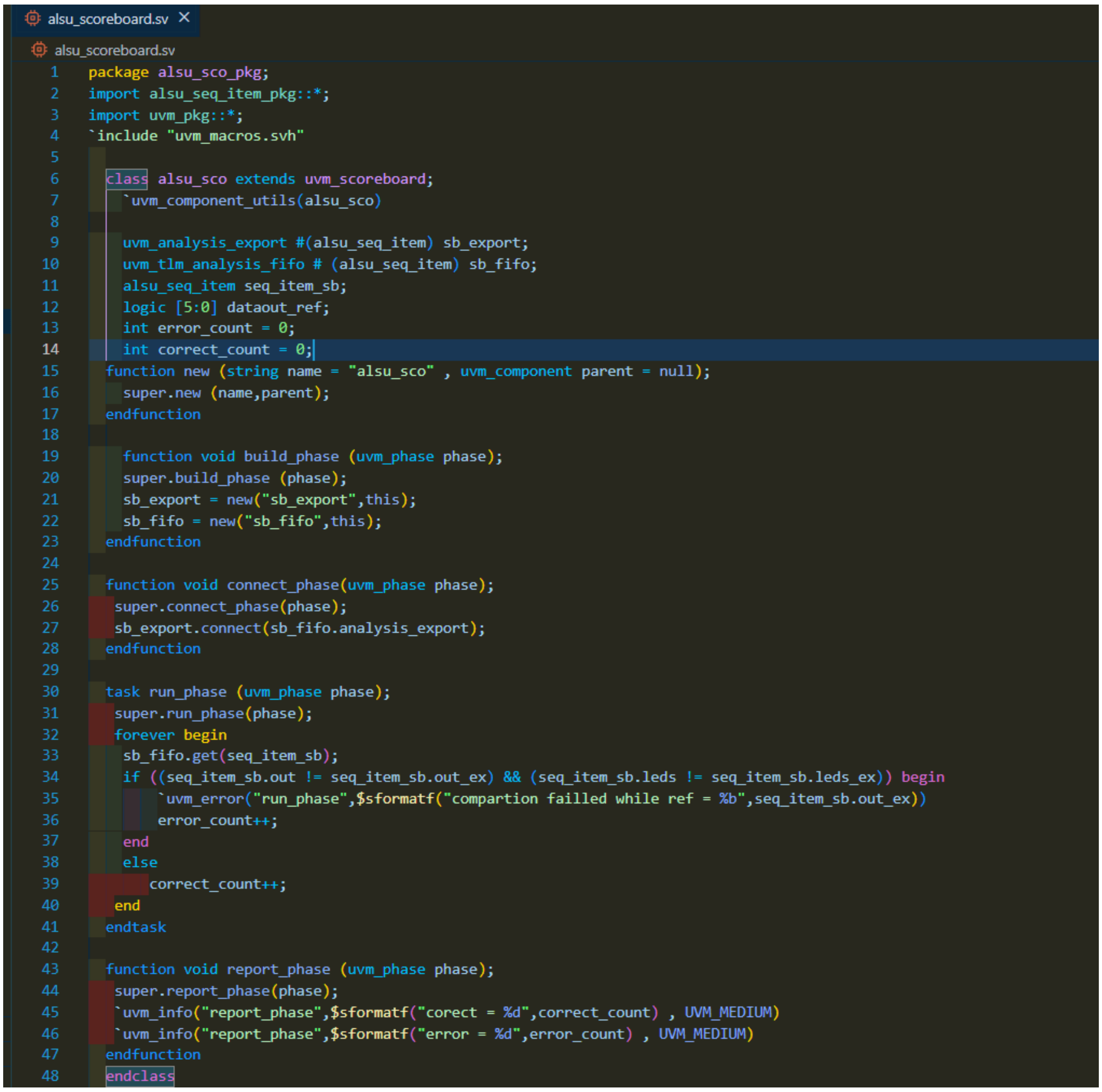
```
interface alsu_if (clk);
  parameter INPUT_PRIORITY = "A";
  parameter FULL_ADDER = "ON";
    input bit clk;
    logic [1:0] cin ;
    logic rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    logic [2:0] opcode;
    logic signed [2:0] A, B;
    logic [15:0] leds;
    logic [15:0] leds_ex;
    logic signed [5:0] out;
    logic signed [5:0] out_ex;

    modport DUT (
    input clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in,opcode,A,B,
    output out,leds
    );

    modport GOLDEN_MODEL (
    input clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in,opcode,A,B,
    output out_ex,leds_ex
    );

endinterface
```



```systemverilog
package alsu_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

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

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

class alsu_seq_item extends uvm_sequence_item;
  `uvm_object_utils(alsu_seq_item);

    rand logic rst;
    rand logic signed [2:0] A,B;
    rand logic red_op_A,red_op_B;
    rand opcode_e opcode;
    rand logic bypass_A, bypass_B;
    rand logic direction,cin;
    rand logic serial_in;
    bit [2:0]walking_ones[] = '{3'b100,3'b010,3'b001};
    rand logic [3:0] A_rem_values , B_rem_values ;
    rand logic [3:0] A_max_values , B_max_values ;
    rand bit[2:0]walking_ones_t , walking_ones_f;
    rand  opcode_e arr[6];
    bit clk;
    logic [15:0] leds;
    logic [15:0] leds_ex;
    logic [5:0] out;
    logic [5:0] out_ex;

  function new (string name = "alsu_seq_item");
    super.new (name);
  endfunction

  function string convert2string();
      return $sformatf("%s rst=%b a=%b b=%b opcode=%s",super.convert2string(),rst,A,B,opcode);
  endfunction

  function string convert2string_stimulus();
      return $sformatf(" rst=%b a=%b b=%b opcode=%s",rst,A,B,opcode);
  endfunction

        constraint rst_c{
            rst dist{1:= 2 ,0:= 98};
        }

```



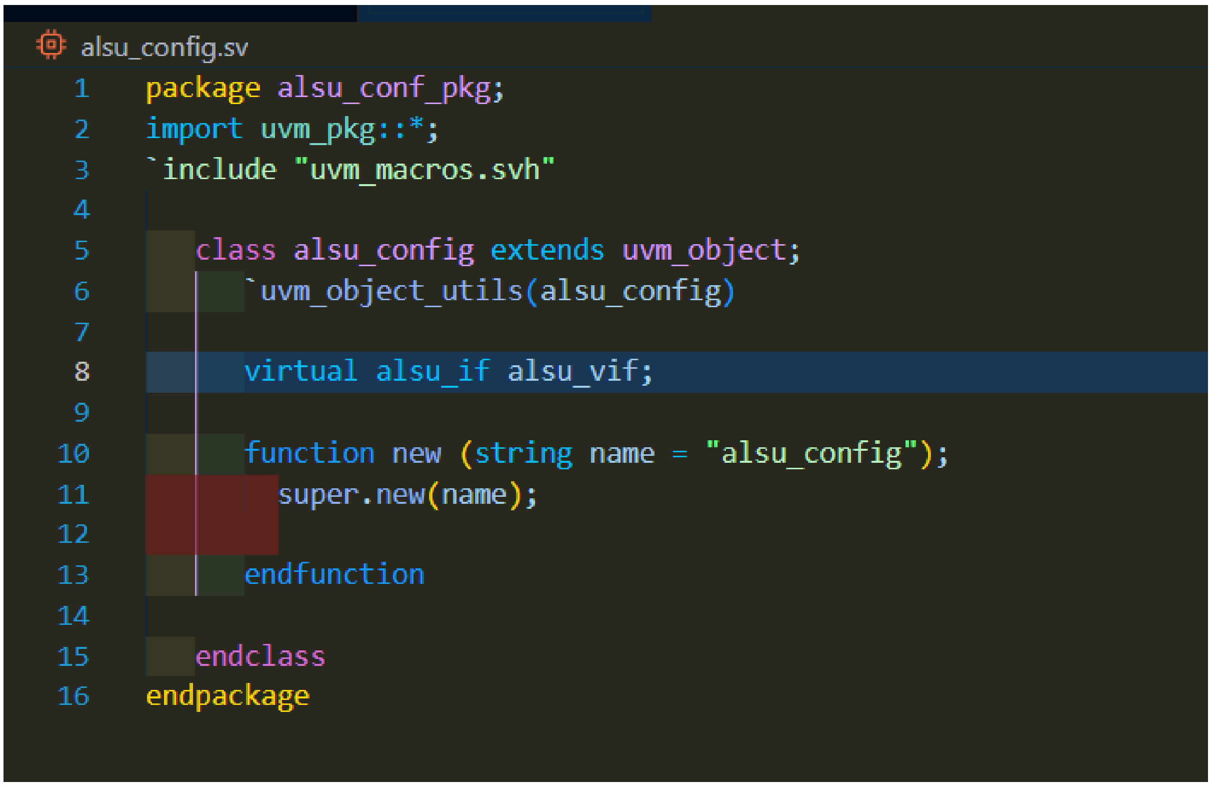
```

        //label2
    constraint opcode_c {
          A_rem_values != MAXPOS || MAXNEG || ZERO ;
          B_rem_values != MAXPOS || MAXNEG || ZERO ;
          A_max_values == MAXPOS || MAXNEG || ZERO ;
          B_max_values == MAXPOS || MAXNEG || ZERO ;
          walking_ones_t inside {walking_ones};
          !(walking_ones_f inside {walking_ones});

        (opcode == ADD || opcode == MULT) -> A dist { A_max_values :/ 80};
        (opcode == ADD || opcode == MULT) -> B dist { B_max_values :/ 80};

        ((opcode == OR || opcode == XOR) && red_op_A) -> A dist {walking_ones_t:=80,walking_ones_f:=20};
        ((opcode == OR || opcode == XOR) && red_op_A) -> B==0;

        ((opcode == OR || opcode == XOR) && red_op_B) -> B dist {walking_ones_t:=80,walking_ones_f:=20};
        ((opcode == OR || opcode == XOR) && red_op_B) -> A ==0;
       }

        constraint opcode_dist {
        opcode dist { [invalid6:invalid7] := 20,[OR:ROTATE] := 80};
        }

        constraint bypass_dist {
        bypass_A dist { 0:=98 , 1:=2};
        bypass_B dist { 0:=98 , 1:=2};
        }

        constraint red_op_dist {
        red_op_A dist { 0:=50 , 1:=50};
        red_op_B dist { 0:=50 , 1:=50};
        }

        constraint arr_c {
            foreach (arr[i]){
              arr[i] == i;
        }
        }

endclass

endpackage
```



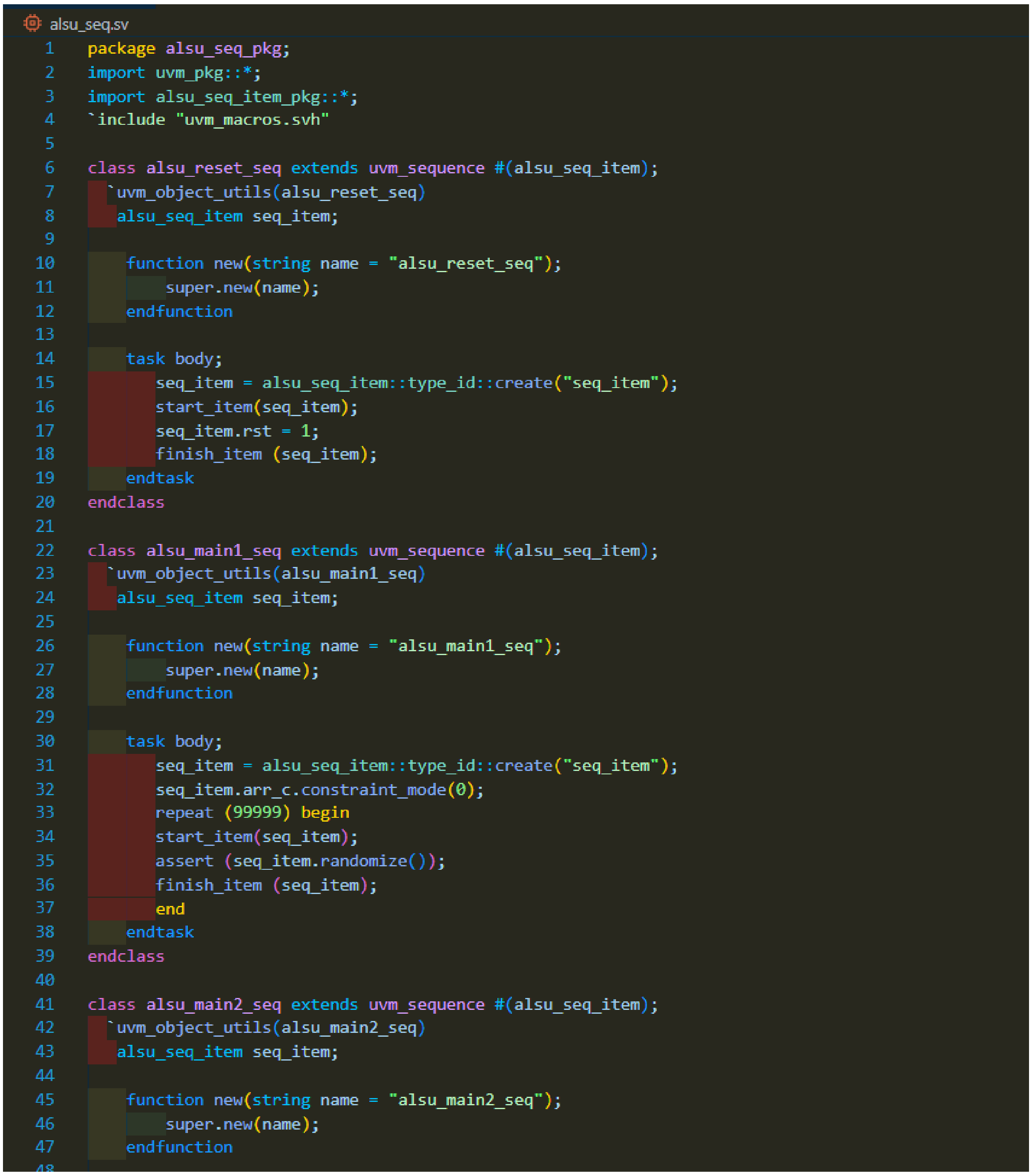
alsu_scoreboard.sv ×

alsu_scoreboard.sv

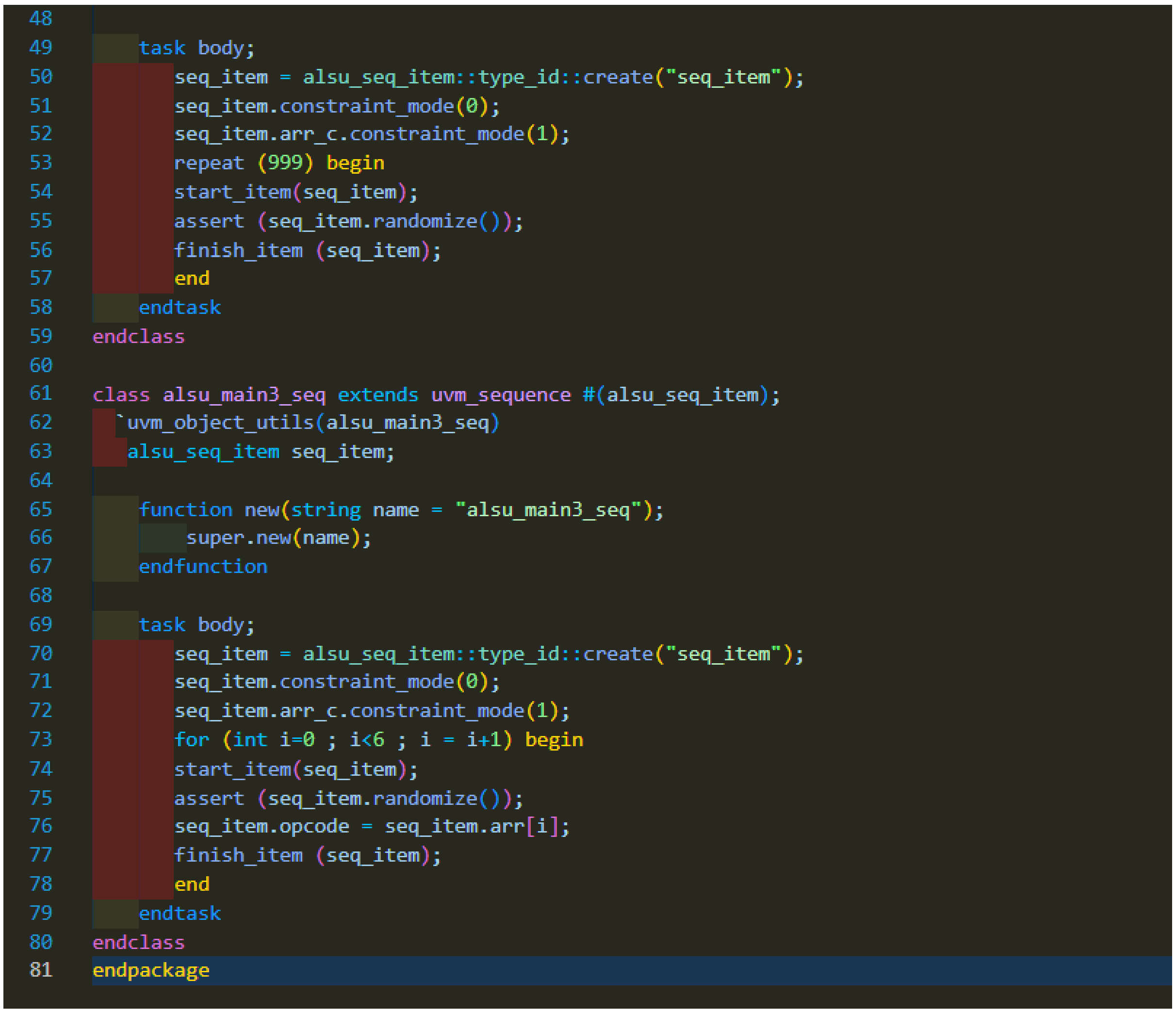
```
package alsu_sco_pkg;
import alsu_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

  class alsu_sco extends uvm_scoreboard;
    `uvm_component_utils(alsu_sco)

    uvm_analysis_export #(alsu_seq_item) sb_export;
    uvm_tlm_analysis_fifo # (alsu_seq_item) sb_fifo;
    alsu_seq_item seq_item_sb;
    logic [5:0] dataout_ref;
    int error_count = 0;
    int correct_count = 0;
  function new (string name = "alsu_sco" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

    function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    sb_export = new("sb_export",this);
    sb_fifo = new("sb_fifo",this);
  endfunction

  function void connect_phase(uvm_phase phase);
   super.connect_phase(phase);
   sb_export.connect(sb_fifo.analysis_export);
  endfunction

  task run_phase (uvm_phase phase);
   super.run_phase(phase);
   forever begin
    sb_fifo.get(seq_item_sb);
    if ((seq_item_sb.out != seq_item_sb.out_ex) && (seq_item_sb.leds != seq_item_sb.leds_ex)) begin
        `uvm_error("run_phase",$sformatf("compartion failled while ref = %b",seq_item_sb.out_ex))
        error_count++;
    end
    else
       correct_count++;
   end
  endtask

  function void report_phase (uvm_phase phase);
   super.report_phase(phase);
   `uvm_info("report_phase",$sformatf("corect = %d",correct_count) , UVM_MEDIUM)
   `uvm_info("report_phase",$sformatf("error = %d",error_count) , UVM_MEDIUM)
  endfunction
  endclass
```


alsu_config.sv

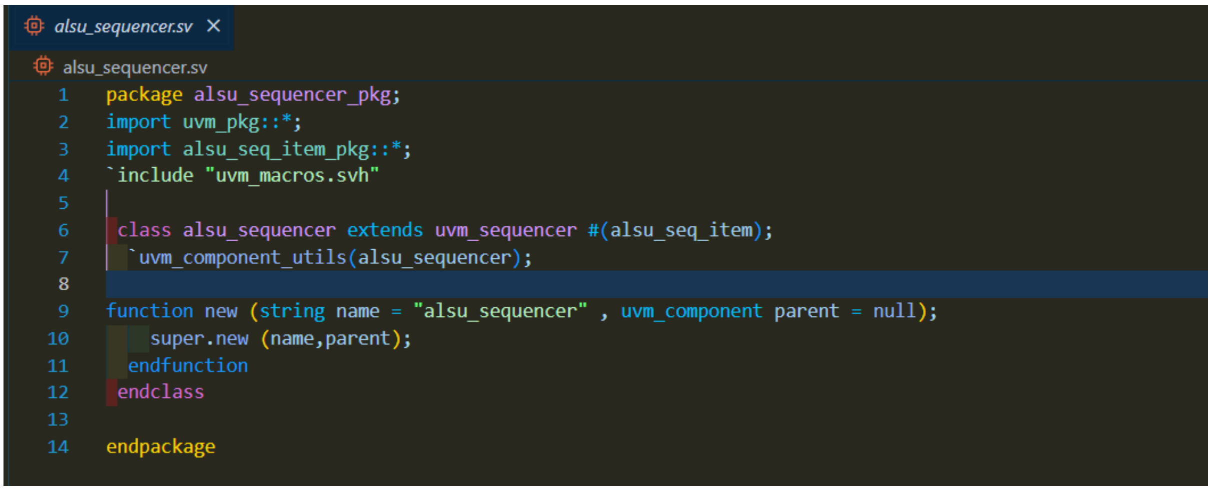
```
package alsu_conf_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_config extends uvm_object;
    `uvm_object_utils(alsu_config)

    virtual alsu_if alsu_vif;

    function new (string name = "alsu_config");
      super.new(name);

    endfunction

endclass
endpackage
```


alsu_seq.sv

```systemverilog
package alsu_seq_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
`include "uvm_macros.svh"

class alsu_reset_seq extends uvm_sequence #(alsu_seq_item);
  `uvm_object_utils(alsu_reset_seq)
  alsu_seq_item seq_item;

    function new(string name = "alsu_reset_seq");
        super.new(name);
    endfunction

    task body;
       seq_item = alsu_seq_item::type_id::create("seq_item");
       start_item(seq_item);
       seq_item.rst = 1;
       finish_item (seq_item);
    endtask
endclass

class alsu_main1_seq extends uvm_sequence #(alsu_seq_item);
  `uvm_object_utils(alsu_main1_seq)
  alsu_seq_item seq_item;

    function new(string name = "alsu_main1_seq");
        super.new(name);
    endfunction

    task body;
       seq_item = alsu_seq_item::type_id::create("seq_item");
       seq_item.arr_c.constraint_mode(0);
       repeat (99999) begin
       start_item(seq_item);
       assert (seq_item.randomize());
       finish_item (seq_item);
       end
    endtask
endclass

class alsu_main2_seq extends uvm_sequence #(alsu_seq_item);
  `uvm_object_utils(alsu_main2_seq)
  alsu_seq_item seq_item;

    function new(string name = "alsu_main2_seq");
        super.new(name);
    endfunction
```



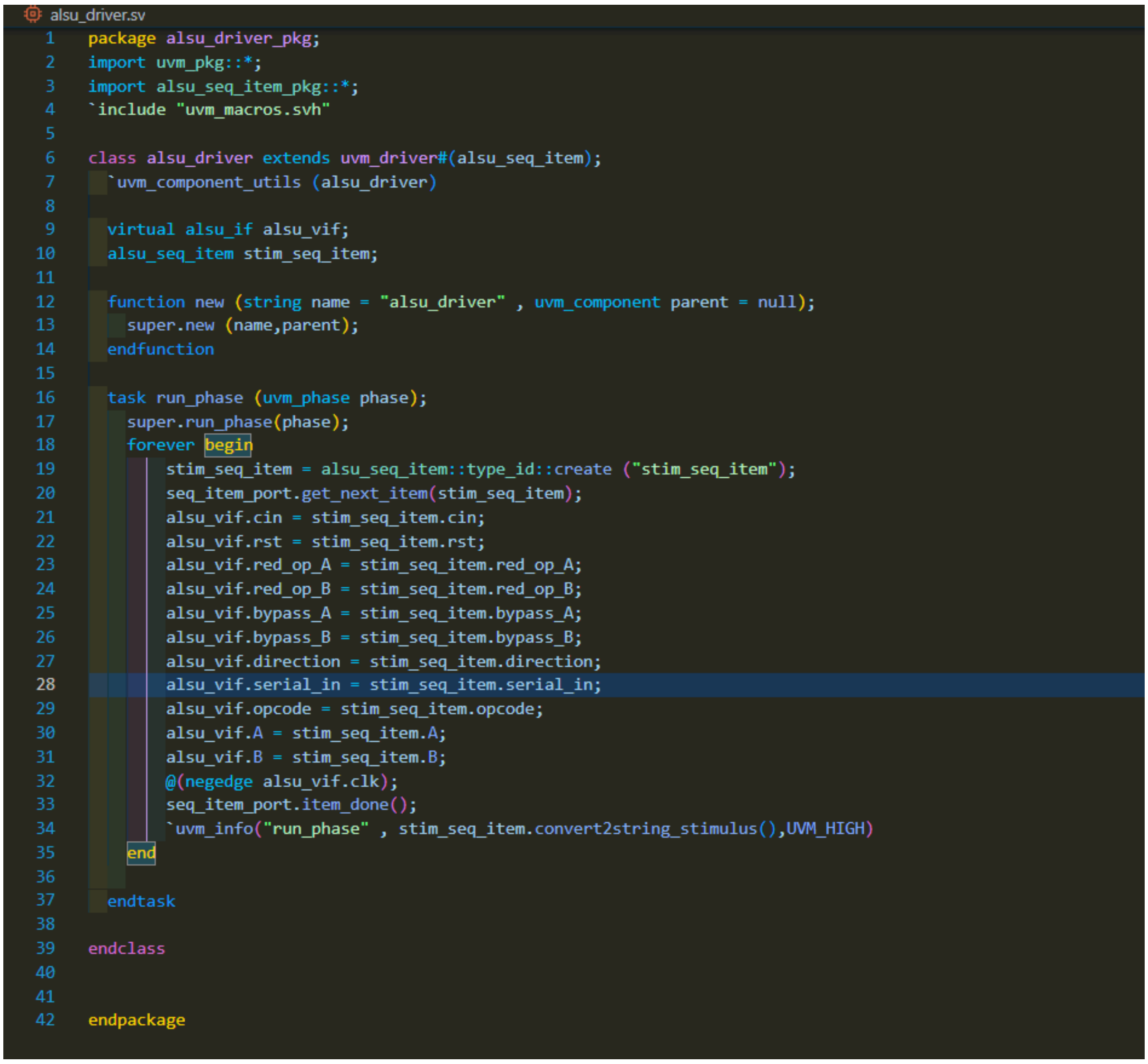
```

    task body;
        seq_item = alsu_seq_item::type_id::create("seq_item");
        seq_item.constraint_mode(0);
        seq_item.arr_c.constraint_mode(1);
        repeat (999) begin
        start_item(seq_item);
        assert (seq_item.randomize());
        finish_item (seq_item);
        end
    endtask
endclass

class alsu_main3_seq extends uvm_sequence #(alsu_seq_item);
  `uvm_object_utils(alsu_main3_seq)
   alsu_seq_item seq_item;

    function new(string name = "alsu_main3_seq");
        super.new(name);
    endfunction

    task body;
        seq_item = alsu_seq_item::type_id::create("seq_item");
        seq_item.constraint_mode(0);
        seq_item.arr_c.constraint_mode(1);
        for (int i=0 ; i<6 ; i = i+1) begin
        start_item(seq_item);
        assert (seq_item.randomize());
        seq_item.opcode = seq_item.arr[i];
        finish_item (seq_item);
        end
    endtask
endclass
endpackage
```



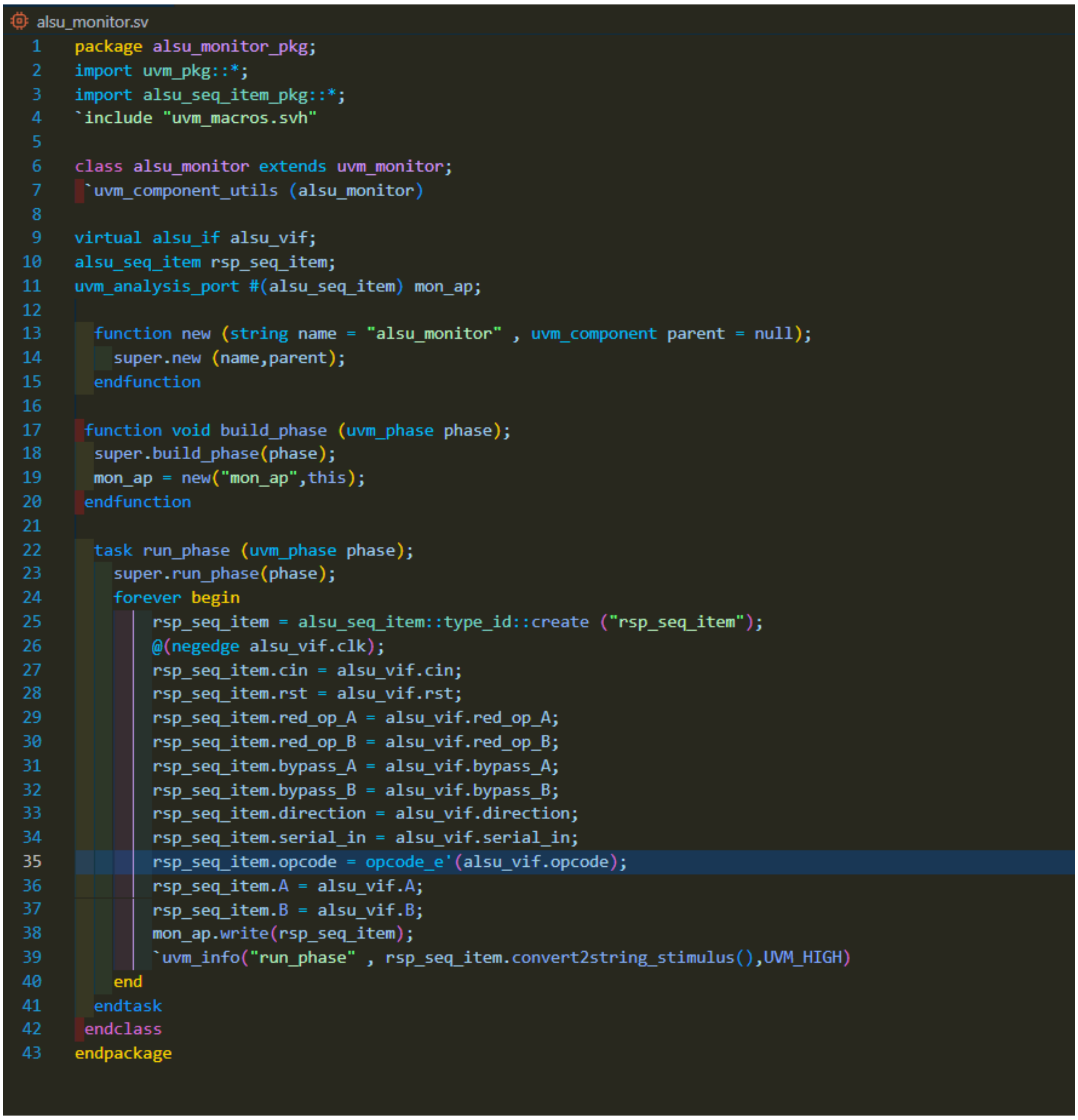
alsu_sequencer.sv

alsu_sequencer.sv

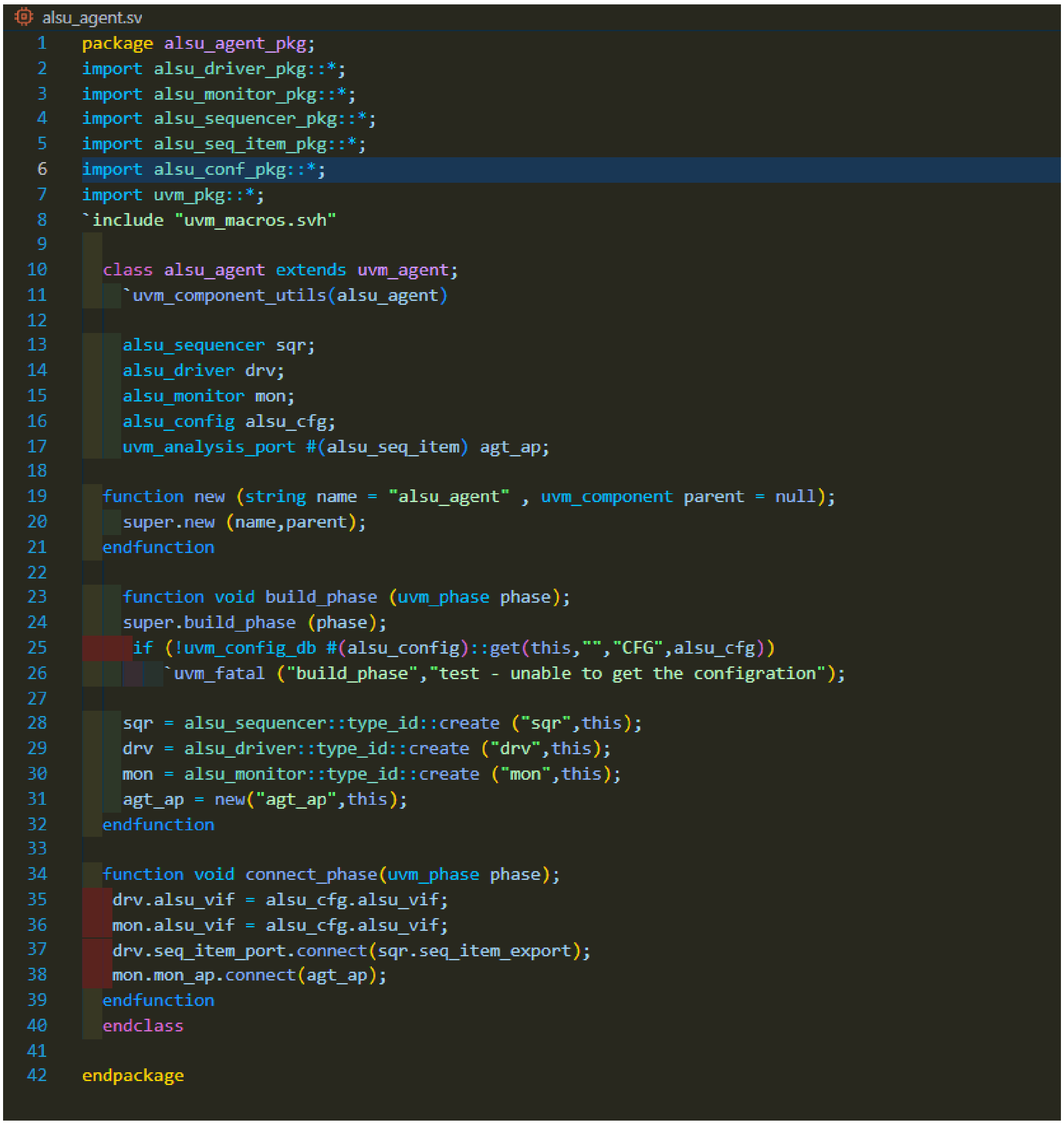
```
package alsu_sequencer_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
`include "uvm_macros.svh"

 class alsu_sequencer extends uvm_sequencer #(alsu_seq_item);
   `uvm_component_utils(alsu_sequencer);

function new (string name = "alsu_sequencer" , uvm_component parent = null);
     super.new (name,parent);
  endfunction
 endclass

endpackage
```



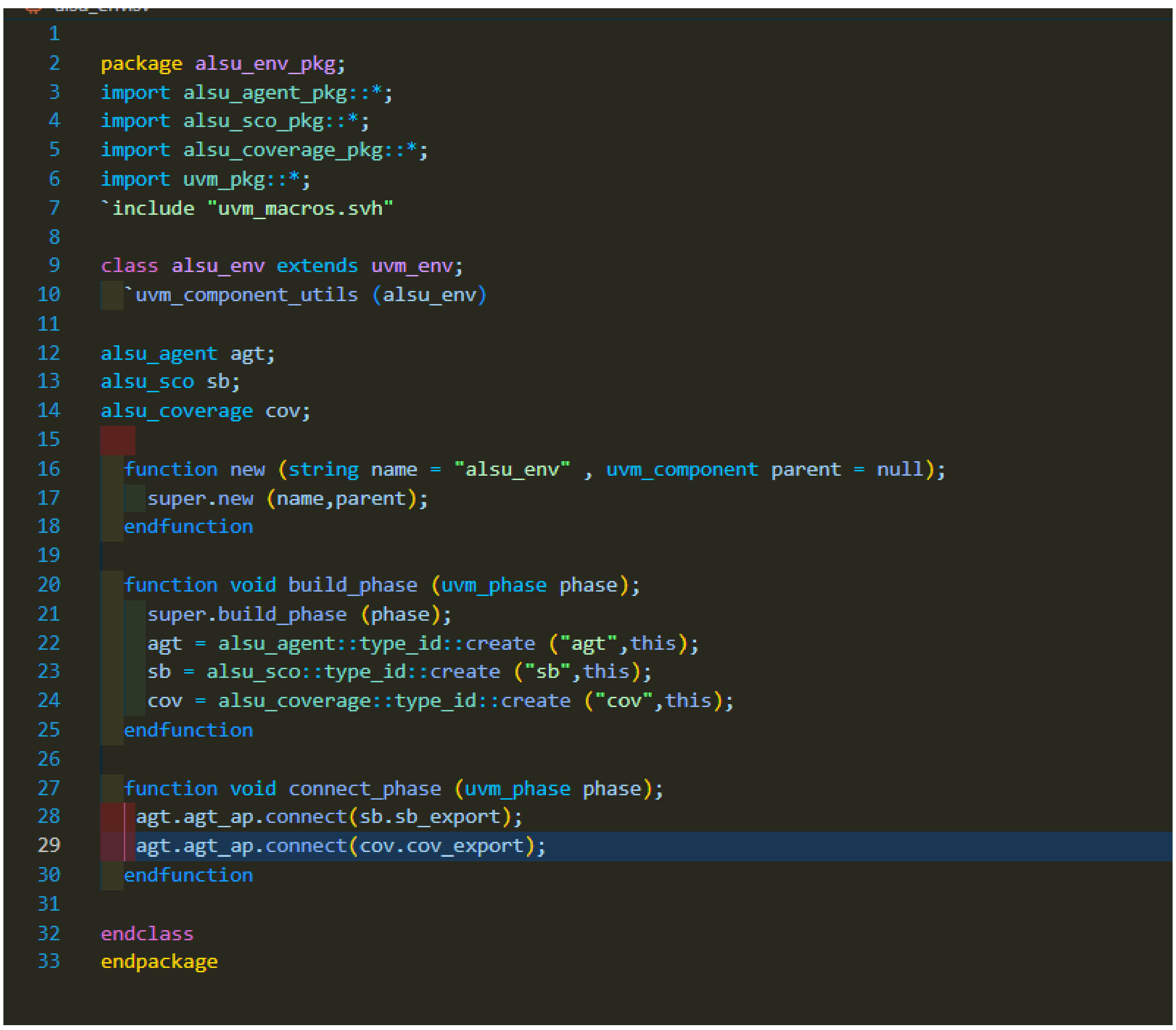
```
package alsu_driver_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
`include "uvm_macros.svh"

class alsu_driver extends uvm_driver#(alsu_seq_item);
  `uvm_component_utils (alsu_driver)

  virtual alsu_if alsu_vif;
  alsu_seq_item stim_seq_item;

  function new (string name = "alsu_driver" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
      stim_seq_item = alsu_seq_item::type_id::create ("stim_seq_item");
      seq_item_port.get_next_item(stim_seq_item);
      alsu_vif.cin = stim_seq_item.cin;
      alsu_vif.rst = stim_seq_item.rst;
      alsu_vif.red_op_A = stim_seq_item.red_op_A;
      alsu_vif.red_op_B = stim_seq_item.red_op_B;
      alsu_vif.bypass_A = stim_seq_item.bypass_A;
      alsu_vif.bypass_B = stim_seq_item.bypass_B;
      alsu_vif.direction = stim_seq_item.direction;
      alsu_vif.serial_in = stim_seq_item.serial_in;
      alsu_vif.opcode = stim_seq_item.opcode;
      alsu_vif.A = stim_seq_item.A;
      alsu_vif.B = stim_seq_item.B;
      @(negedge alsu_vif.clk);
      seq_item_port.item_done();
      `uvm_info("run_phase" , stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end

  endtask

endclass


endpackage
```


alsu_monitor.sv

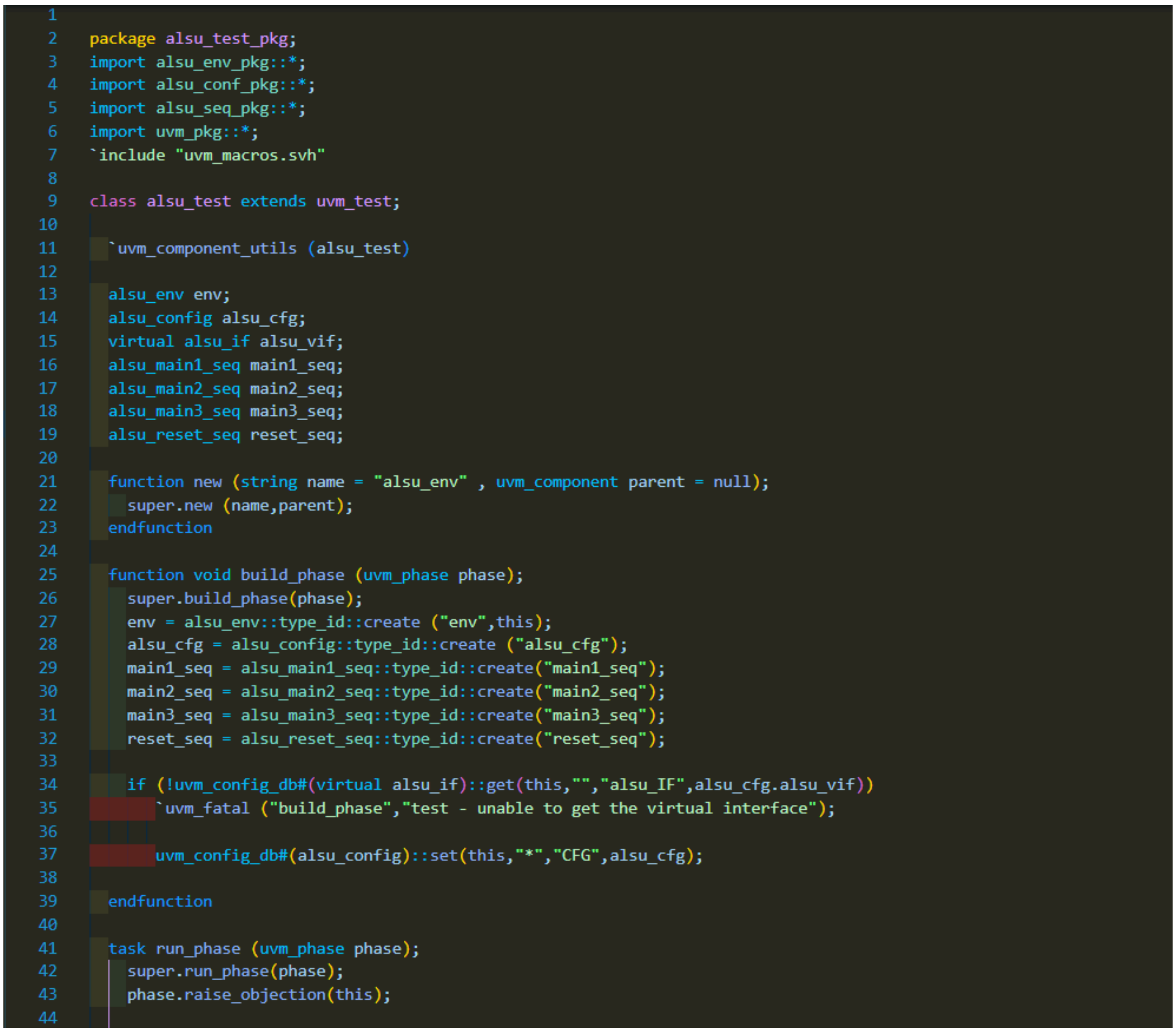
```
package alsu_monitor_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
`include "uvm_macros.svh"

class alsu_monitor extends uvm_monitor;
 `uvm_component_utils (alsu_monitor)

virtual alsu_if alsu_vif;
alsu_seq_item rsp_seq_item;
uvm_analysis_port #(alsu_seq_item) mon_ap;

  function new (string name = "alsu_monitor" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

 function void build_phase (uvm_phase phase);
  super.build_phase(phase);
  mon_ap = new("mon_ap",this);
 endfunction

  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
        rsp_seq_item = alsu_seq_item::type_id::create ("rsp_seq_item");
        @(negedge alsu_vif.clk);
        rsp_seq_item.cin = alsu_vif.cin;
        rsp_seq_item.rst = alsu_vif.rst;
        rsp_seq_item.red_op_A = alsu_vif.red_op_A;
        rsp_seq_item.red_op_B = alsu_vif.red_op_B;
        rsp_seq_item.bypass_A = alsu_vif.bypass_A;
        rsp_seq_item.bypass_B = alsu_vif.bypass_B;
        rsp_seq_item.direction = alsu_vif.direction;
        rsp_seq_item.serial_in = alsu_vif.serial_in;
        rsp_seq_item.opcode = opcode_e'(alsu_vif.opcode);
        rsp_seq_item.A = alsu_vif.A;
        rsp_seq_item.B = alsu_vif.B;
        mon_ap.write(rsp_seq_item);
        `uvm_info("run_phase" , rsp_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
 endclass
endpackage
```


alsu_agent.sv

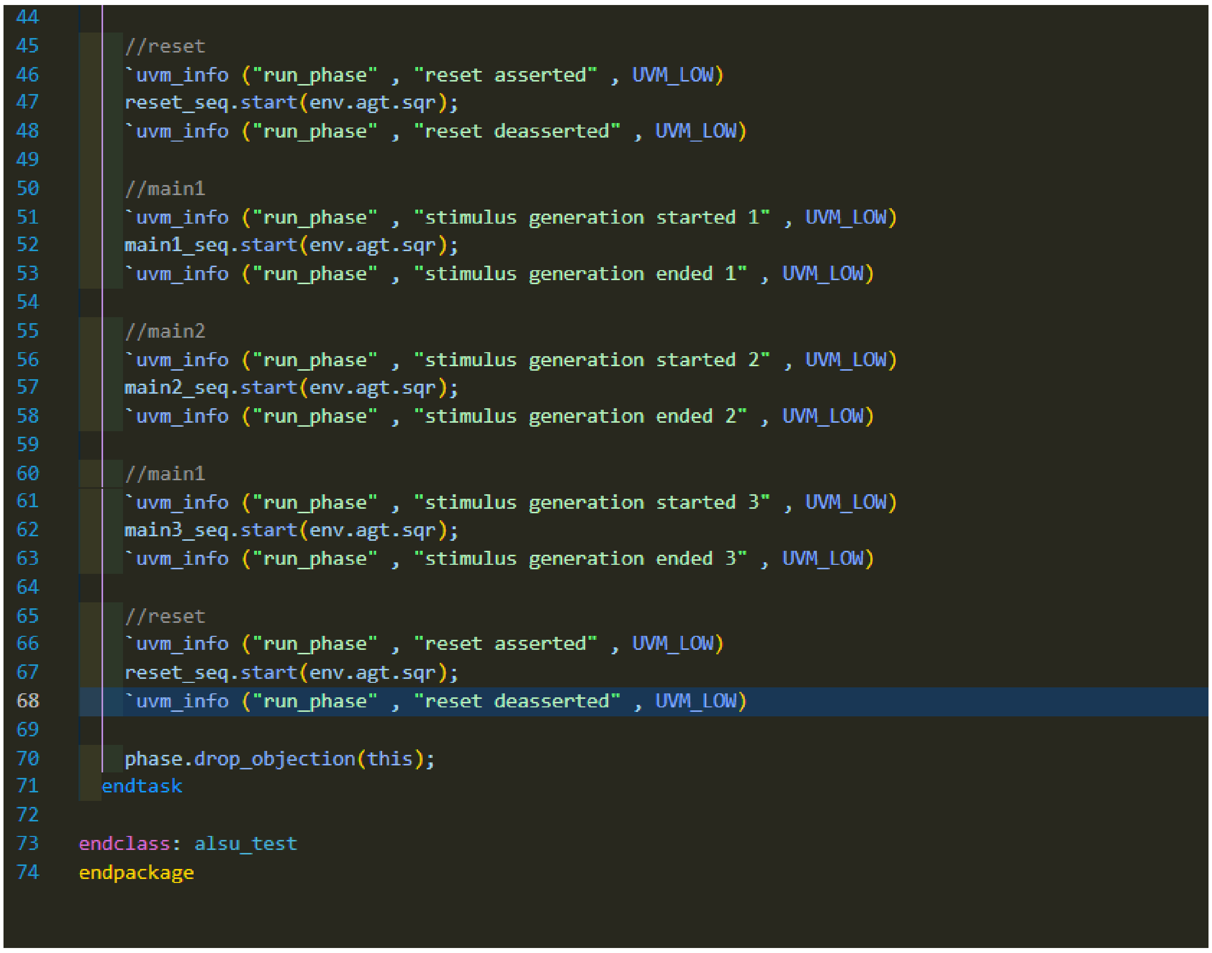
```systemverilog
package alsu_agent_pkg;
import alsu_driver_pkg::*;
import alsu_monitor_pkg::*;
import alsu_sequencer_pkg::*;
import alsu_seq_item_pkg::*;
import alsu_conf_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

  class alsu_agent extends uvm_agent;
    `uvm_component_utils(alsu_agent)

    alsu_sequencer sqr;
    alsu_driver drv;
    alsu_monitor mon;
    alsu_config alsu_cfg;
    uvm_analysis_port #(alsu_seq_item) agt_ap;

  function new (string name = "alsu_agent" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

    function void build_phase (uvm_phase phase);
    super.build_phase (phase);
     if (!uvm_config_db #(alsu_config)::get(this,"","CFG",alsu_cfg))
        `uvm_fatal ("build_phase","test - unable to get the configration");

    sqr = alsu_sequencer::type_id::create ("sqr",this);
    drv = alsu_driver::type_id::create ("drv",this);
    mon = alsu_monitor::type_id::create ("mon",this);
    agt_ap = new("agt_ap",this);
  endfunction

  function void connect_phase(uvm_phase phase);
   drv.alsu_vif = alsu_cfg.alsu_vif;
   mon.alsu_vif = alsu_cfg.alsu_vif;
   drv.seq_item_port.connect(sqr.seq_item_export);
   mon.mon_ap.connect(agt_ap);
  endfunction
  endclass

endpackage
```



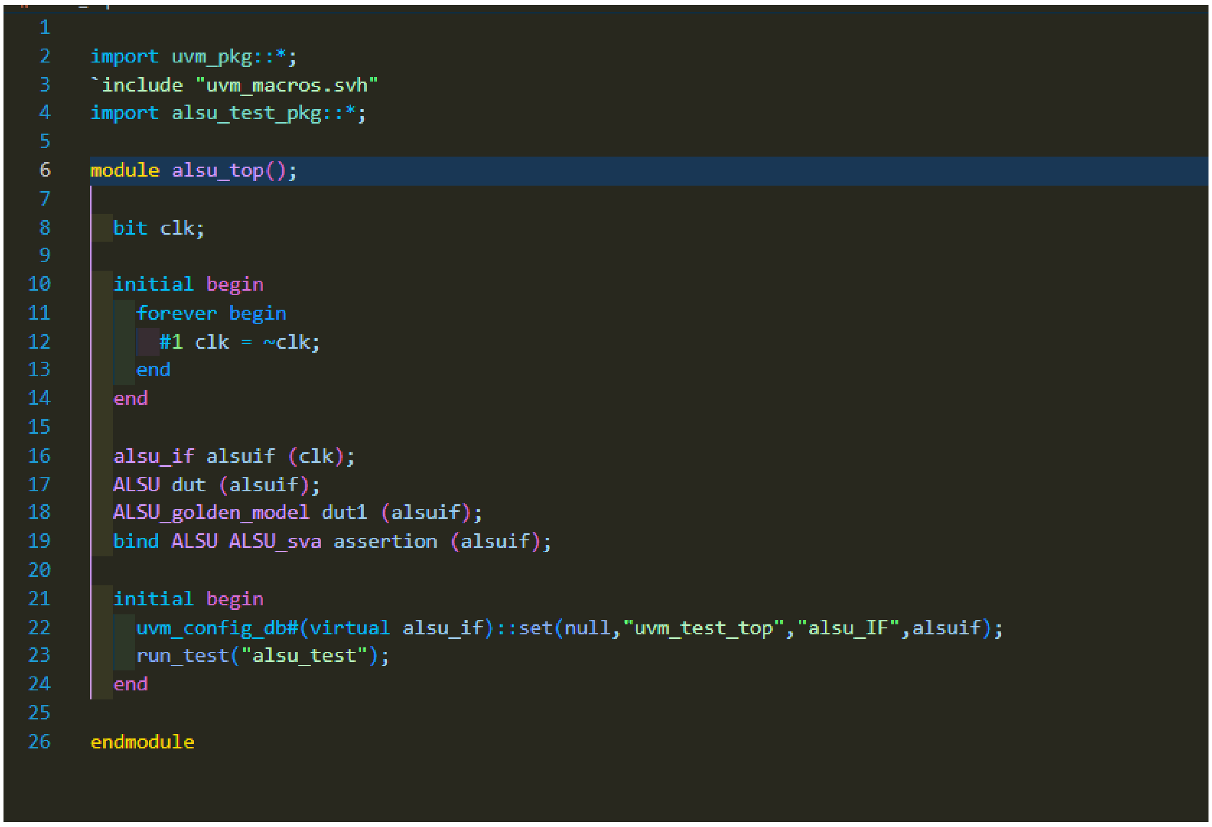
```

package alsu_env_pkg;
import alsu_agent_pkg::*;
import alsu_sco_pkg::*;
import alsu_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_env extends uvm_env;
  `uvm_component_utils (alsu_env)

alsu_agent agt;
alsu_sco sb;
alsu_coverage cov;

  function new (string name = "alsu_env" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

  function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    agt = alsu_agent::type_id::create ("agt",this);
    sb = alsu_sco::type_id::create ("sb",this);
    cov = alsu_coverage::type_id::create ("cov",this);
  endfunction

  function void connect_phase (uvm_phase phase);
   agt.agt_ap.connect(sb.sb_export);
   agt.agt_ap.connect(cov.cov_export);
  endfunction

endclass
endpackage
```



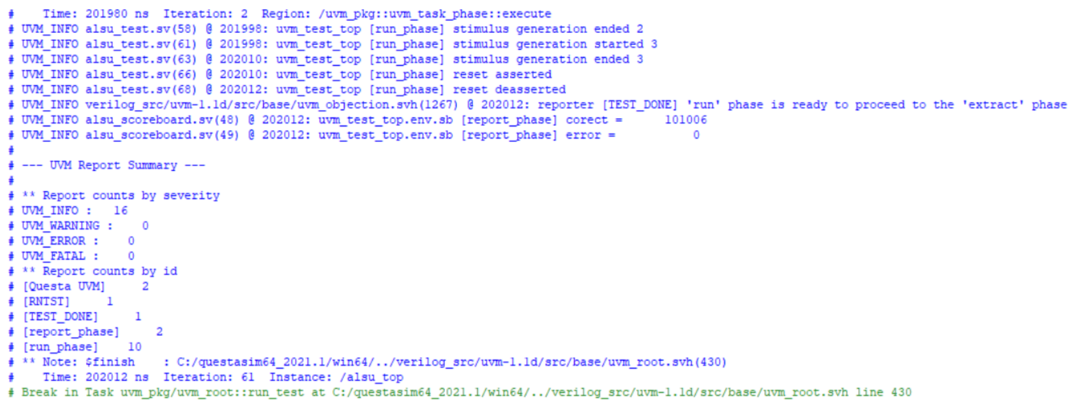
```

package alsu_test_pkg;
import alsu_env_pkg::*;
import alsu_conf_pkg::*;
import alsu_seq_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_test extends uvm_test;

  `uvm_component_utils (alsu_test)

  alsu_env env;
  alsu_config alsu_cfg;
  virtual alsu_if alsu_vif;
  alsu_main1_seq main1_seq;
  alsu_main2_seq main2_seq;
  alsu_main3_seq main3_seq;
  alsu_reset_seq reset_seq;

  function new (string name = "alsu_env" , uvm_component parent = null);
    super.new (name,parent);
  endfunction

  function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    env = alsu_env::type_id::create ("env",this);
    alsu_cfg = alsu_config::type_id::create ("alsu_cfg");
    main1_seq = alsu_main1_seq::type_id::create("main1_seq");
    main2_seq = alsu_main2_seq::type_id::create("main2_seq");
    main3_seq = alsu_main3_seq::type_id::create("main3_seq");
    reset_seq = alsu_reset_seq::type_id::create("reset_seq");

    if (!uvm_config_db#(virtual alsu_if)::get(this,"","alsu_IF",alsu_cfg.alsu_vif))
       `uvm_fatal ("build_phase","test - unable to get the virtual interface");

       uvm_config_db#(alsu_config)::set(this,"*","CFG",alsu_cfg);

  endfunction

  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);

```



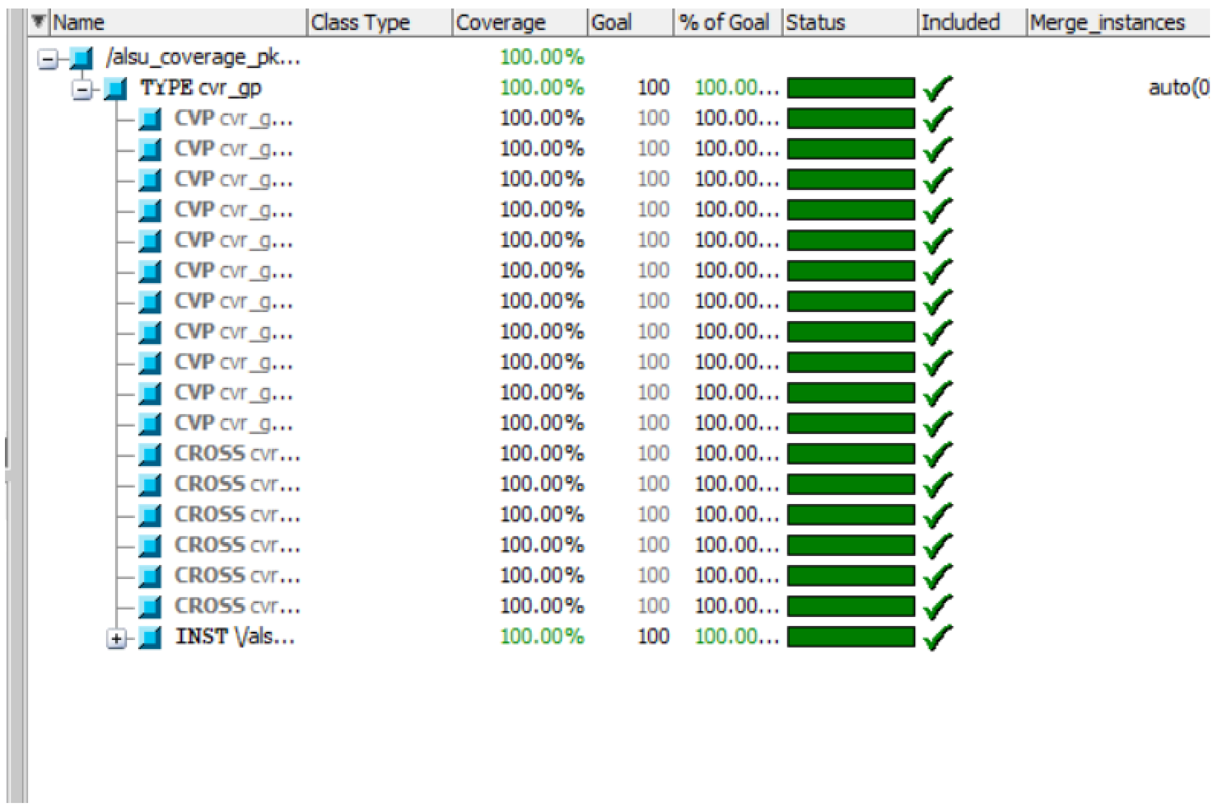
```

    //reset
    `uvm_info ("run_phase" , "reset asserted" , UVM_LOW)
    reset_seq.start(env.agt.sqr);
    `uvm_info ("run_phase" , "reset deasserted" , UVM_LOW)

    //main1
    `uvm_info ("run_phase" , "stimulus generation started 1" , UVM_LOW)
    main1_seq.start(env.agt.sqr);
    `uvm_info ("run_phase" , "stimulus generation ended 1" , UVM_LOW)

    //main2
    `uvm_info ("run_phase" , "stimulus generation started 2" , UVM_LOW)
    main2_seq.start(env.agt.sqr);
    `uvm_info ("run_phase" , "stimulus generation ended 2" , UVM_LOW)

    //main1
    `uvm_info ("run_phase" , "stimulus generation started 3" , UVM_LOW)
    main3_seq.start(env.agt.sqr);
    `uvm_info ("run_phase" , "stimulus generation ended 3" , UVM_LOW)

    //reset
    `uvm_info ("run_phase" , "reset asserted" , UVM_LOW)
    reset_seq.start(env.agt.sqr);
    `uvm_info ("run_phase" , "reset deasserted" , UVM_LOW)

    phase.drop_objection(this);
  endtask

endclass: alsu_test
endpackage
```



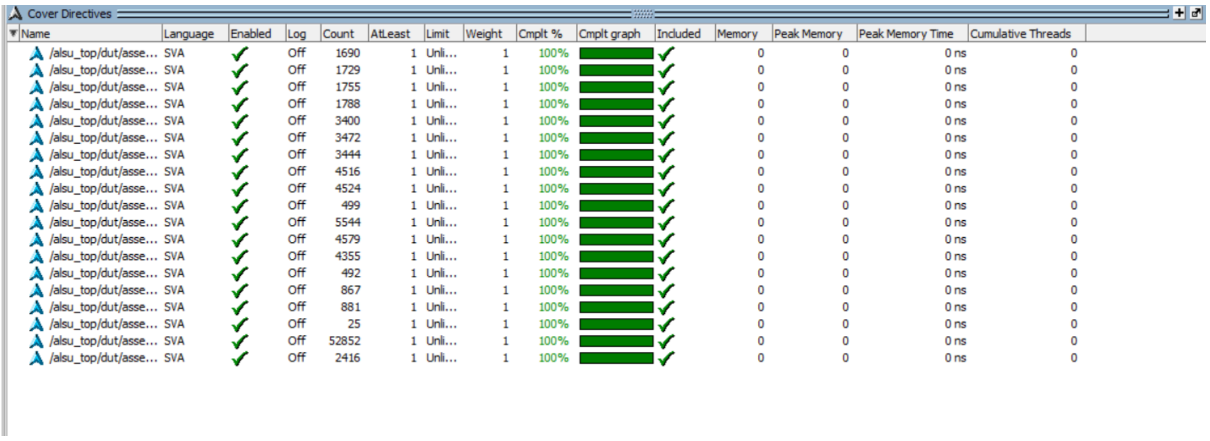
```

import uvm_pkg::*;
`include "uvm_macros.svh"
import alsu_test_pkg::*;

module alsu_top();

  bit clk;

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

  alsu_if alsuif (clk);
  ALSU dut (alsuif);
  ALSU_golden_model dut1 (alsuif);
  bind ALSU ALSU_sva assertion (alsuif);

  initial begin
    uvm_config_db#(virtual alsu_if)::set(null,"uvm_test_top","alsu_IF",alsuif);
    run_test("alsu_test");
  end

endmodule
```



```
#    Time: 201980 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
# UVM_INFO alsu_test.sv(58) @ 201998: uvm_test_top [run_phase] stimulus generation ended 2
# UVM_INFO alsu_test.sv(61) @ 201998: uvm_test_top [run_phase] stimulus generation started 3
# UVM_INFO alsu_test.sv(63) @ 202010: uvm_test_top [run_phase] stimulus generation ended 3
# UVM_INFO alsu_test.sv(66) @ 202010: uvm_test_top [run_phase] reset asserted
# UVM_INFO alsu_test.sv(68) @ 202012: uvm_test_top [run_phase] reset deasserted
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 202012: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO alsu_scoreboard.sv(48) @ 202012: uvm_test_top.env.sb [report_phase] corect =      101006
# UVM_INFO alsu_scoreboard.sv(49) @ 202012: uvm_test_top.env.sb [report_phase] error =           0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]    10
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 202012 ns  Iteration: 61  Instance: /alsu_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

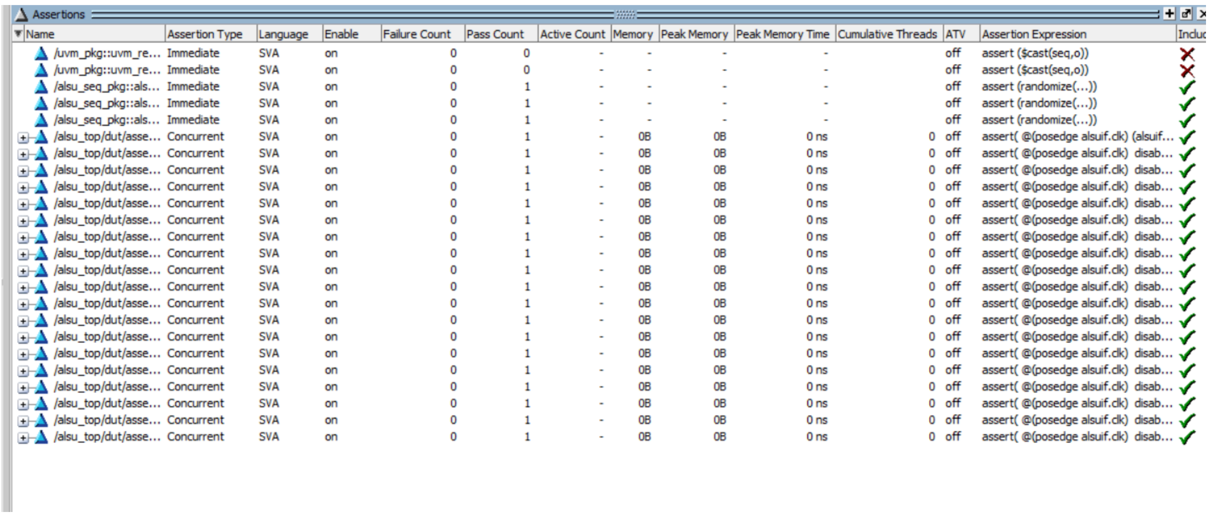


Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances
/alsu_coverage_pk...		100.00%					
TYPE cvr_gp		100.00%	100	100.00...	🟩	✓	auto(0
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CVP cvr_g...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
CROSS cvr...		100.00%	100	100.00...	🟩	✓	
INST \/als...		100.00%	100	100.00...	🟩	✓	



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/alsu_top/dut/asse...	SVA	✓	Off	1690	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	1729	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	1755	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	1788	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	3400	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	3472	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	3444	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	4516	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	4524	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	499	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	5544	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	4579	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	4355	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	492	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	867	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	881	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	25	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	52852	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/alsu_top/dut/asse...	SVA	✓	Off	2416	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Incluc
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/alsu_seq_pkg::als...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/alsu_seq_pkg::als...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/alsu_seq_pkg::als...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) (alsuif...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓
/alsu_top/dut/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge alsuif.clk) disab...	✓