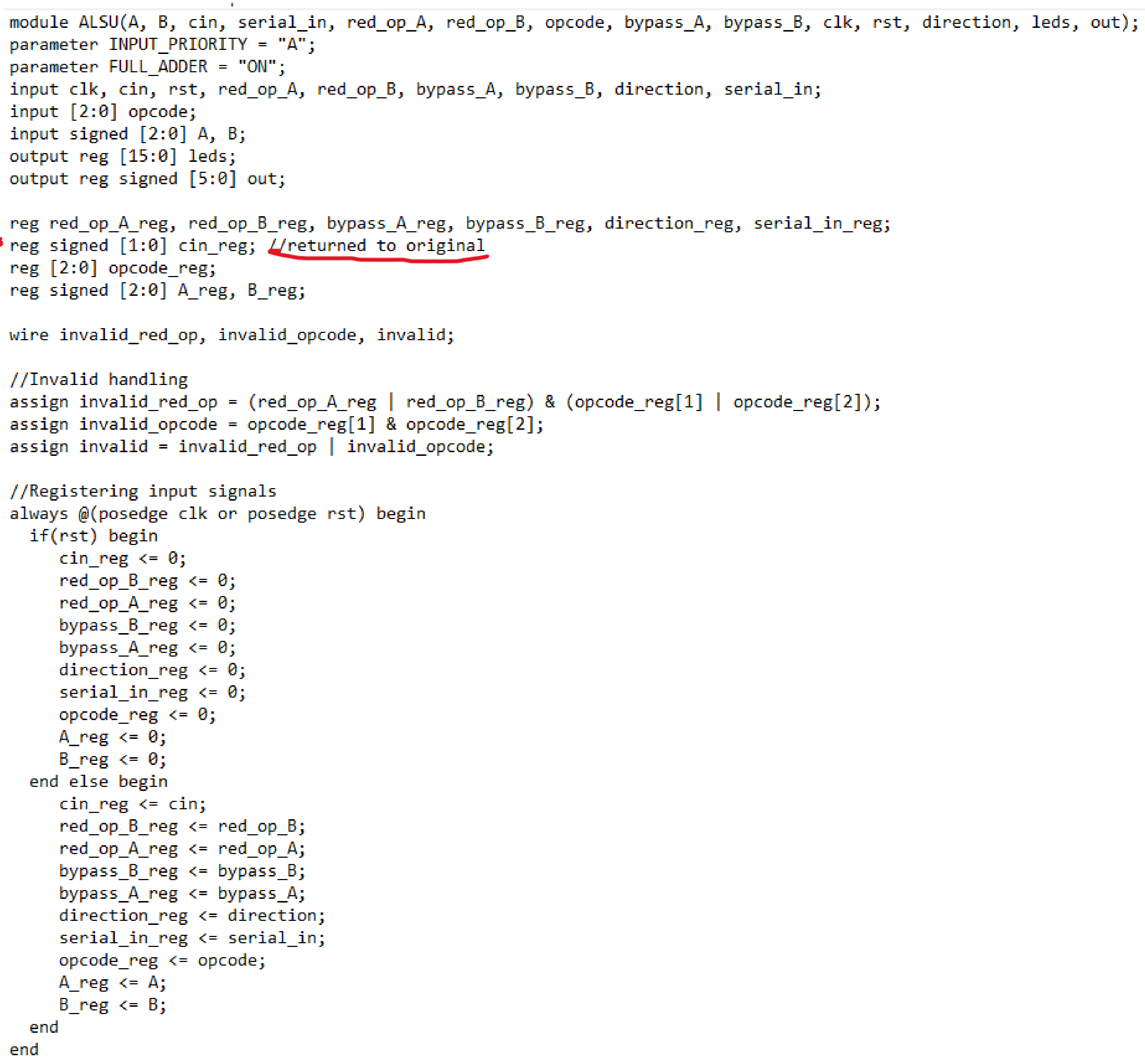
Name: Yousef Wael Mahmoud Alkattan

Part3)

1. Top module

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

module alsu_top;

  bit clk;
  alsu_if intf(clk);
  ALSU dut (
        .clk(clk),
        .rst(intf.rst),
        .cin(intf.cin),
        .red_op_A(intf.red_op_A),
        .red_op_B(intf.red_op_B),
        .bypass_A(intf.bypass_A),
        .bypass_B(intf.bypass_B),
        .direction(intf.direction),
        .serial_in(intf.serial_in),
        .opcode(intf.opcode),
        .A(intf.A),
        .B(intf.B),
        .leds(intf.leds),
        .out(intf.out)
);

alsu_assertions sva ( .intf(intf.assertions) );

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

  initial begin
    uvm_config_db#(virtual alsu_if)::set(null, "*", "alsu_vif", intf);
    run_test("alsu_test");
  end

endmodule
```

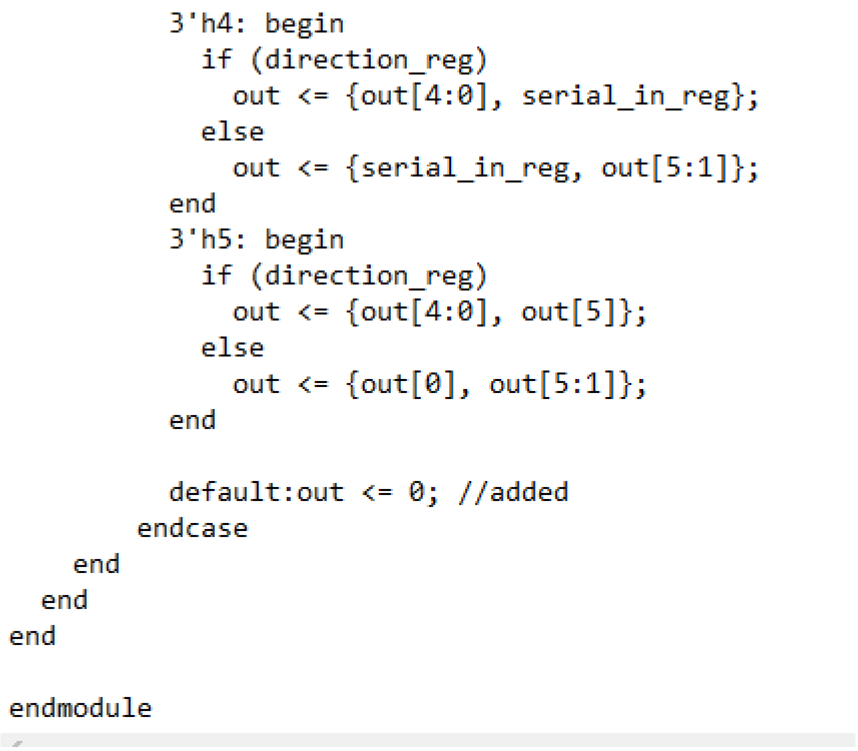
2. Design



```
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, 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 [1:0] cin_reg; //returned to original
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
```

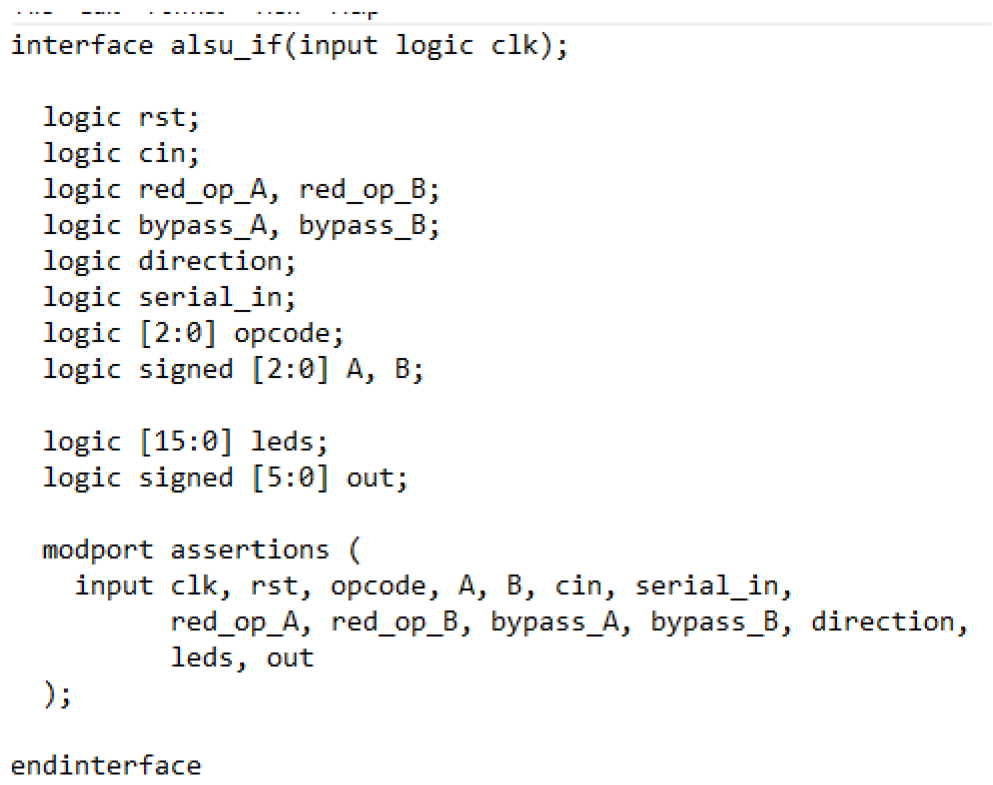
```
//leds output blinking
always @(posedge clk or posedge rst) begin
  if(rst) begin
     leds <= 0;
  end else begin
      if (invalid)
        leds <= ~leds;
      else
        leds <= 0;
  end
end

//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)
          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;
          end
          3'h2: out <= A_reg + B_reg + cin_reg; //fixed
          3'h3: out <= A_reg * B_reg;
```

was wrong in past assignments



```
          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; //added
        endcase
    end
  end
end

endmodule
```

3. Interface

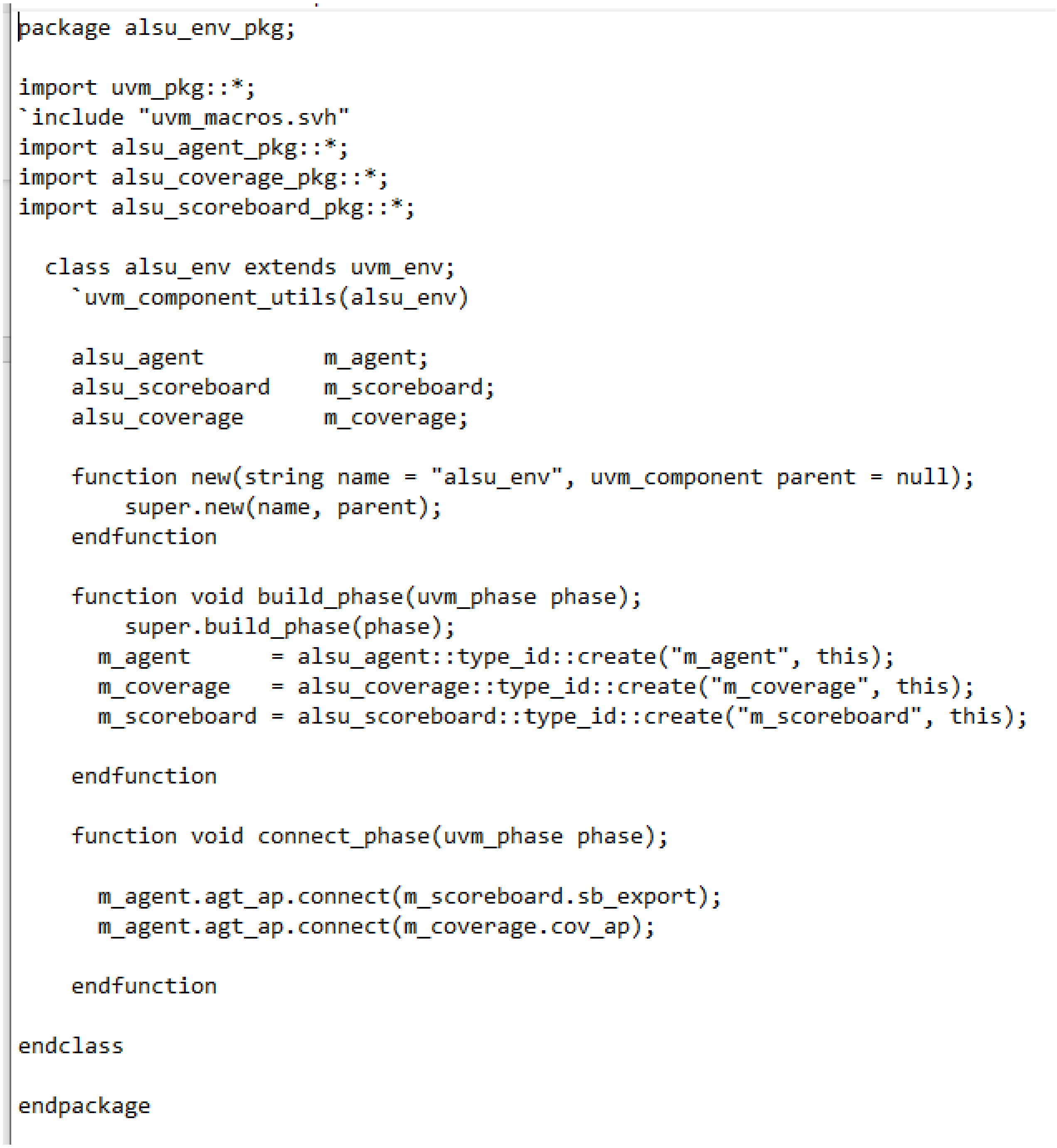


```
interface alsu_if(input logic clk);

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

  logic [15:0] leds;
  logic signed [5:0] out;

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

endinterface
```

4. Test

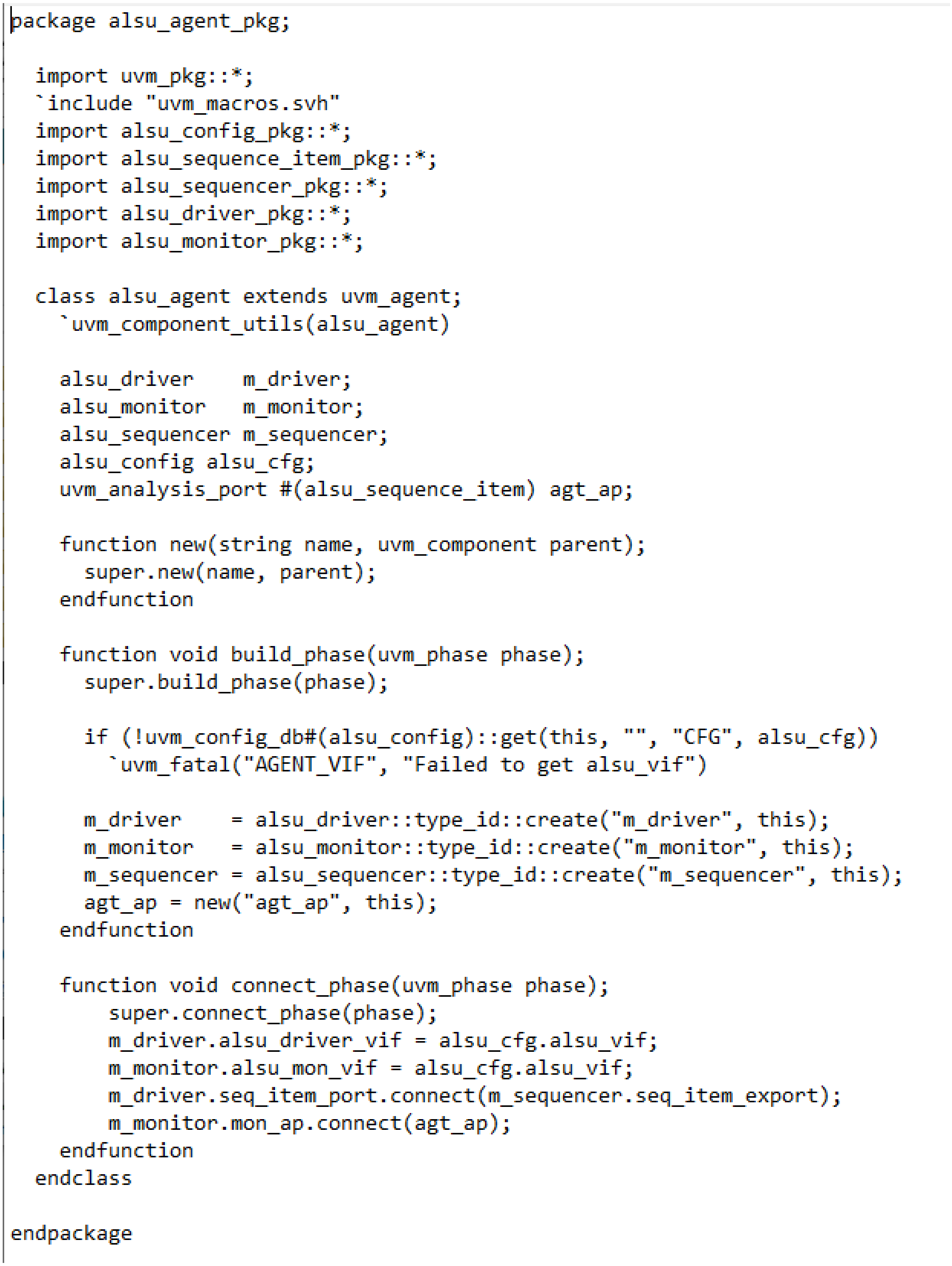
```
package alsu_test_pkg;

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

import alsu_env_pkg::*;
import alsu_config_pkg::*;
import alsu_sequence_pkg::*;
import alsu_reset_sequence_pkg::*;

class alsu_test extends uvm_test;
   `uvm_component_utils(alsu_test)

    alsu_env env;
    alsu_config alsu_cfg;
    alsu_sequence seq1;
    alsu_reset_sequence seq2;

    function new(string name = "alsu_test", 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");
        seq1 = alsu_sequence::type_id::create("seq1");
        seq2 = alsu_reset_sequence::type_id::create("seq2");

        if (!uvm_config_db#(virtual alsu_if)::get(this, "", "alsu_vif", alsu_cfg.alsu_vif))
        `uvm_fatal("VIF_GET", "Failed to get virtual interface in alsu_test")

        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);
        `uvm_info("ALSU_TEST", "Inside the ALSU test", UVM_MEDIUM)

        seq2.start(env.m_agent.m_sequencer);
        seq1.start(env.m_agent.m_sequencer);

        phase.drop_objection(this);
    endtask

endclass

endpackage
```

5. Env



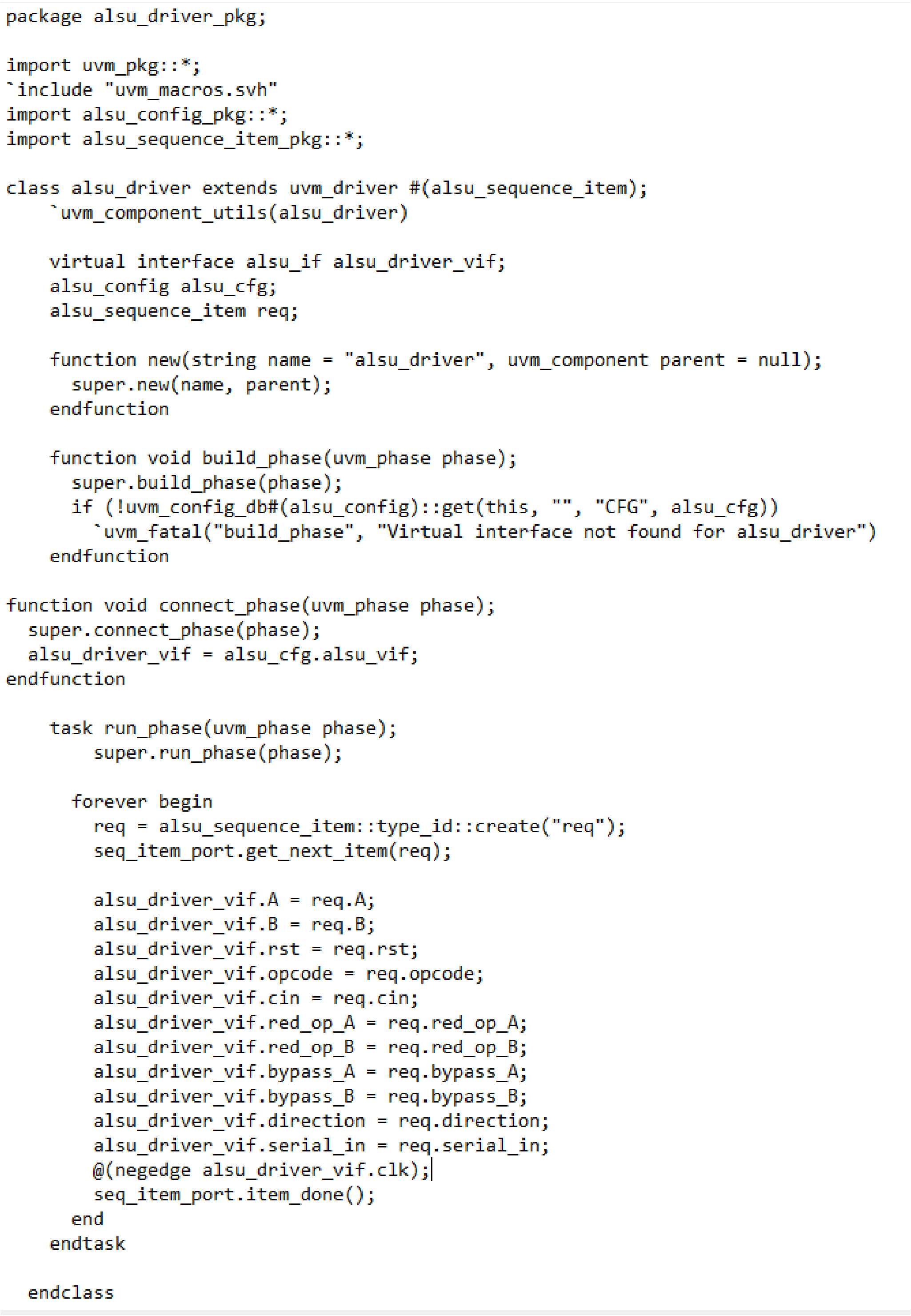
```
package alsu_env_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import alsu_agent_pkg::*;
import alsu_coverage_pkg::*;
import alsu_scoreboard_pkg::*;

  class alsu_env extends uvm_env;
    `uvm_component_utils(alsu_env)

    alsu_agent         m_agent;
    alsu_scoreboard    m_scoreboard;
    alsu_coverage      m_coverage;

    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);
      m_agent      = alsu_agent::type_id::create("m_agent", this);
      m_coverage   = alsu_coverage::type_id::create("m_coverage", this);
      m_scoreboard = alsu_scoreboard::type_id::create("m_scoreboard", this);

    endfunction

    function void connect_phase(uvm_phase phase);

      m_agent.agt_ap.connect(m_scoreboard.sb_export);
      m_agent.agt_ap.connect(m_coverage.cov_ap);

    endfunction

endclass

endpackage
```

6. Agent



```
package alsu_agent_pkg;

  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import alsu_config_pkg::*;
  import alsu_sequence_item_pkg::*;
  import alsu_sequencer_pkg::*;
  import alsu_driver_pkg::*;
  import alsu_monitor_pkg::*;

  class alsu_agent extends uvm_agent;
    `uvm_component_utils(alsu_agent)

    alsu_driver    m_driver;
    alsu_monitor   m_monitor;
    alsu_sequencer m_sequencer;
    alsu_config alsu_cfg;
    uvm_analysis_port #(alsu_sequence_item) agt_ap;

    function new(string name, uvm_component parent);
      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("AGENT_VIF", "Failed to get alsu_vif")

      m_driver    = alsu_driver::type_id::create("m_driver", this);
      m_monitor   = alsu_monitor::type_id::create("m_monitor", this);
      m_sequencer = alsu_sequencer::type_id::create("m_sequencer", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        m_driver.alsu_driver_vif = alsu_cfg.alsu_vif;
        m_monitor.alsu_mon_vif = alsu_cfg.alsu_vif;
        m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
        m_monitor.mon_ap.connect(agt_ap);
    endfunction
  endclass

endpackage
```

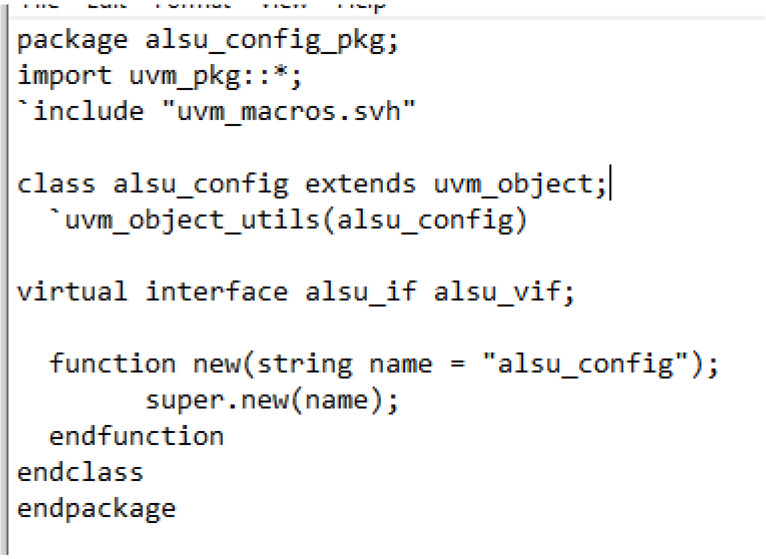
7. Driver



```
package alsu_driver_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import alsu_config_pkg::*;
import alsu_sequence_item_pkg::*;

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

    virtual interface alsu_if alsu_driver_vif;
    alsu_config alsu_cfg;
    alsu_sequence_item req;

    function new(string name = "alsu_driver", 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", "Virtual interface not found for alsu_driver")
    endfunction

function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  alsu_driver_vif = alsu_cfg.alsu_vif;
endfunction

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

      forever begin
        req = alsu_sequence_item::type_id::create("req");
        seq_item_port.get_next_item(req);

        alsu_driver_vif.A = req.A;
        alsu_driver_vif.B = req.B;
        alsu_driver_vif.rst = req.rst;
        alsu_driver_vif.opcode = req.opcode;
        alsu_driver_vif.cin = req.cin;
        alsu_driver_vif.red_op_A = req.red_op_A;
        alsu_driver_vif.red_op_B = req.red_op_B;
        alsu_driver_vif.bypass_A = req.bypass_A;
        alsu_driver_vif.bypass_B = req.bypass_B;
        alsu_driver_vif.direction = req.direction;
        alsu_driver_vif.serial_in = req.serial_in;
        @(negedge alsu_driver_vif.clk);
        seq_item_port.item_done();
      end
    endtask

  endclass
```

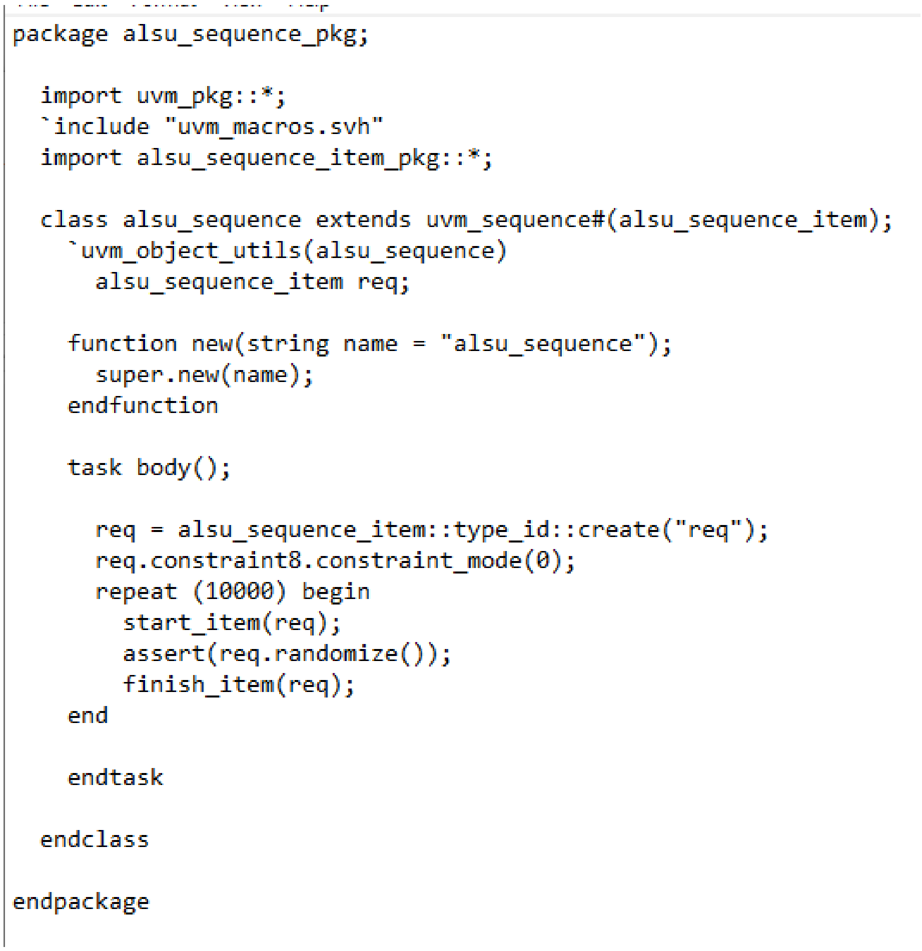
```
endpackage
```

8. Config



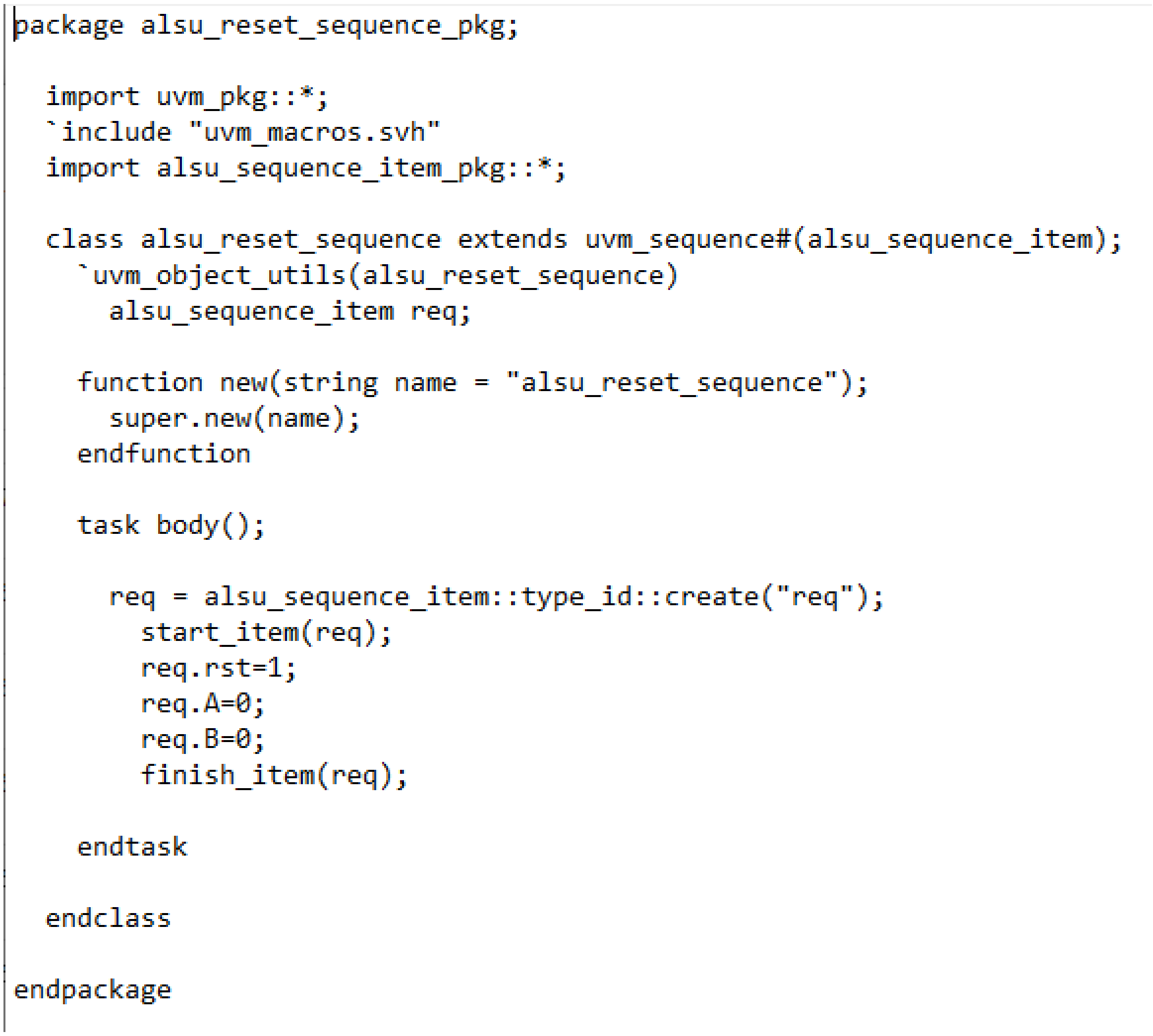
```
package alsu_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_config extends uvm_object;
  `uvm_object_utils(alsu_config)

virtual interface alsu_if alsu_vif;

  function new(string name = "alsu_config");
        super.new(name);
  endfunction
endclass
endpackage
```

9. Sequence



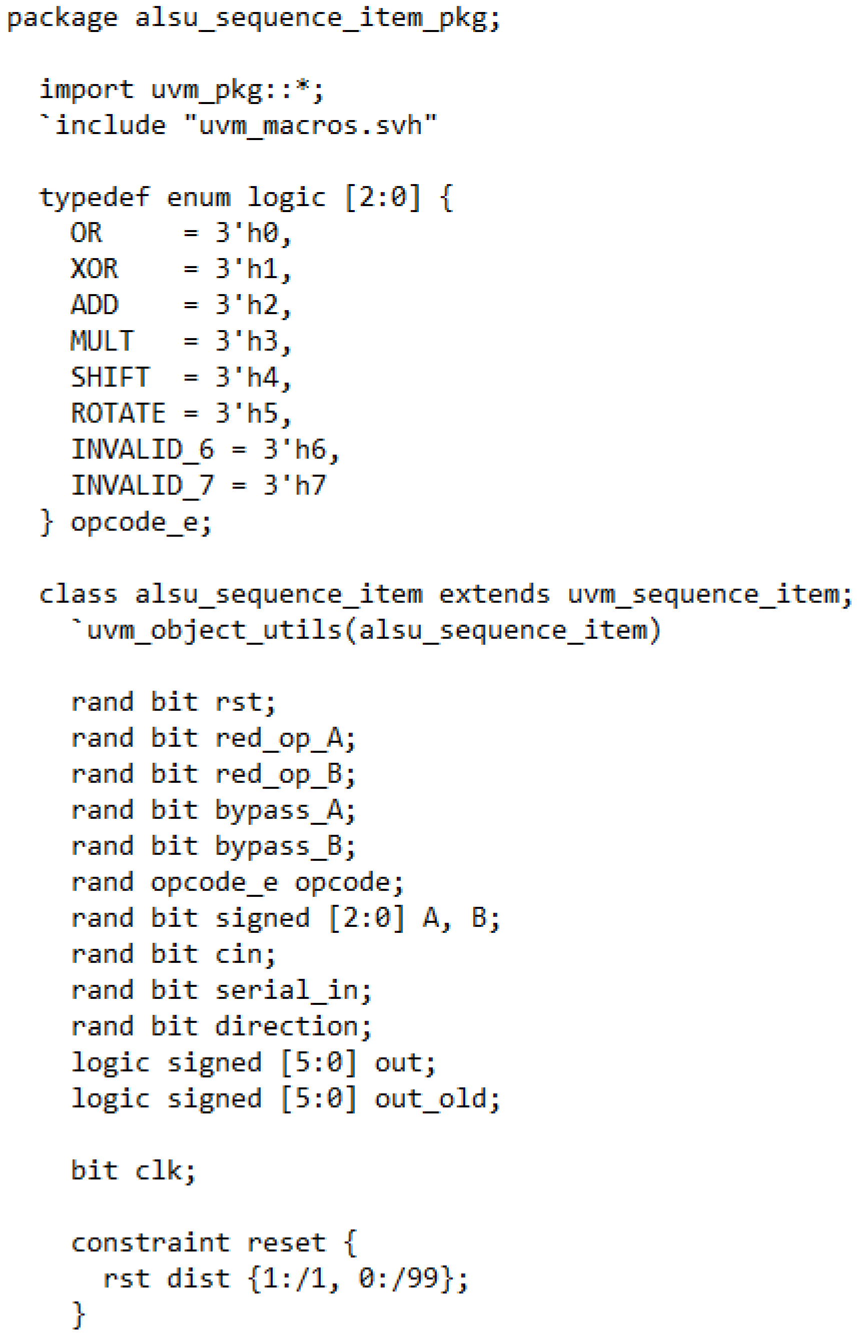
```
package alsu_sequence_pkg;

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

  class alsu_sequence extends uvm_sequence#(alsu_sequence_item);
    `uvm_object_utils(alsu_sequence)
      alsu_sequence_item req;

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

    task body();

      req = alsu_sequence_item::type_id::create("req");
      req.constraint8.constraint_mode(0);
      repeat (10000) begin
        start_item(req);
        assert(req.randomize());
        finish_item(req);
    end

    endtask

  endclass

endpackage
```

10. Reset Sequence

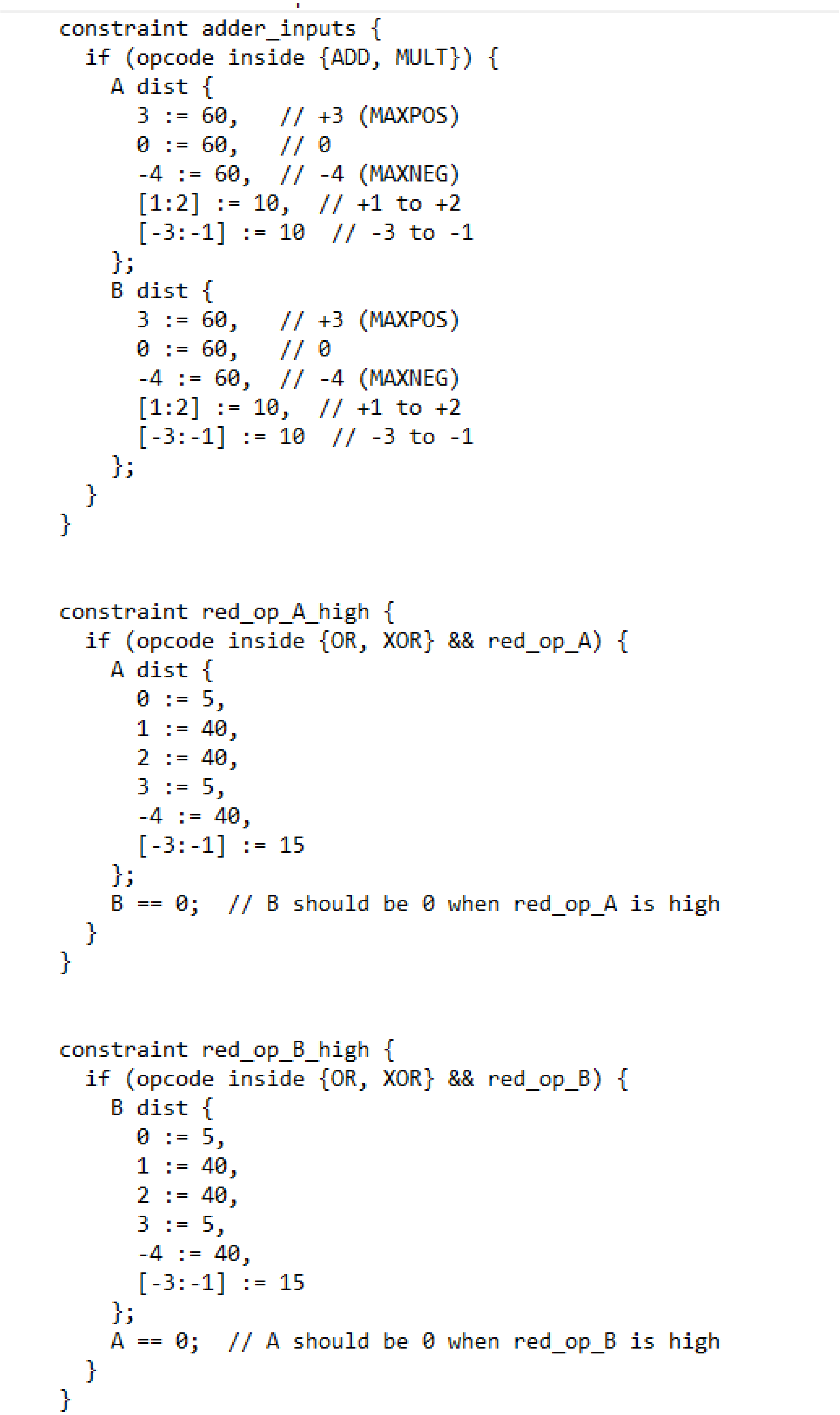


```
package alsu_reset_sequence_pkg;

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

  class alsu_reset_sequence extends uvm_sequence#(alsu_sequence_item);
    `uvm_object_utils(alsu_reset_sequence)
      alsu_sequence_item req;

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

    task body();

      req = alsu_sequence_item::type_id::create("req");
        start_item(req);
        req.rst=1;
        req.A=0;
        req.B=0;
        finish_item(req);

    endtask

  endclass

endpackage
```

11. Sequence Item



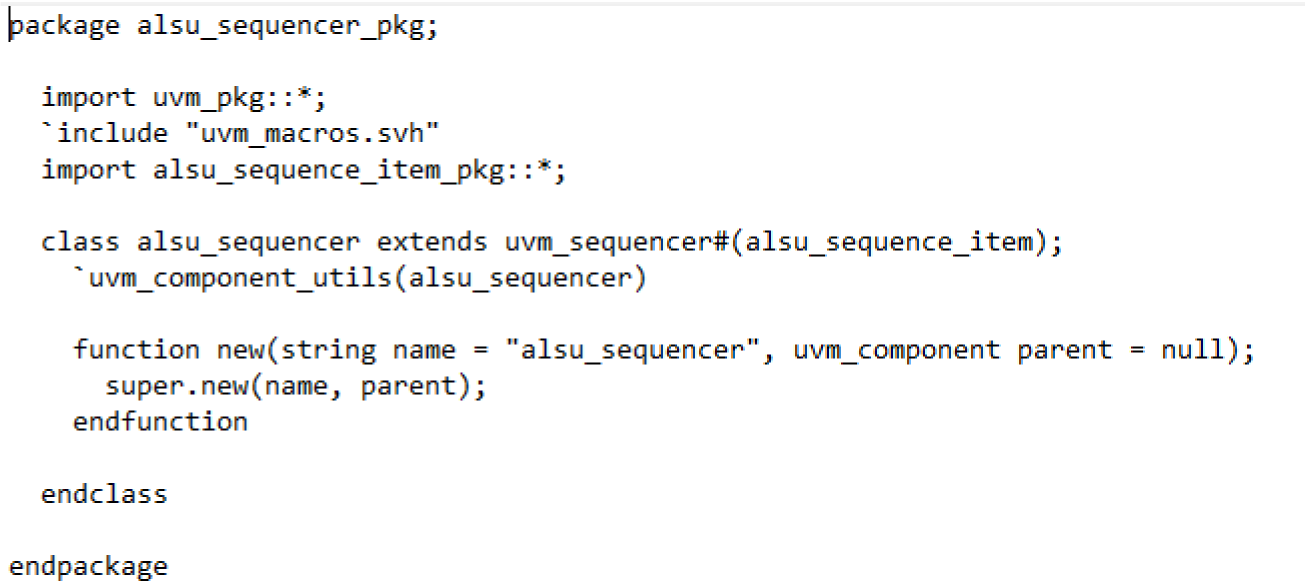
```
package alsu_sequence_item_pkg;

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

  typedef enum logic [2:0] {
    OR     = 3'h0,
    XOR    = 3'h1,
    ADD    = 3'h2,
    MULT   = 3'h3,
    SHIFT  = 3'h4,
    ROTATE = 3'h5,
    INVALID_6 = 3'h6,
    INVALID_7 = 3'h7
  } opcode_e;

  class alsu_sequence_item extends uvm_sequence_item;
    `uvm_object_utils(alsu_sequence_item)

    rand bit rst;
    rand bit red_op_A;
    rand bit red_op_B;
    rand bit bypass_A;
    rand bit bypass_B;
    rand opcode_e opcode;
    rand bit signed [2:0] A, B;
    rand bit cin;
    rand bit serial_in;
    rand bit direction;
    logic signed [5:0] out;
    logic signed [5:0] out_old;

    bit clk;

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



```
constraint adder_inputs {
  if (opcode inside {ADD, MULT}) {
    A dist {
      3 := 60,    // +3 (MAXPOS)
      0 := 60,    // 0
      -4 := 60,   // -4 (MAXNEG)
      [1:2] := 10,   // +1 to +2
      [-3:-1] := 10  // -3 to -1
    };
    B dist {
      3 := 60,    // +3 (MAXPOS)
      0 := 60,    // 0
      -4 := 60,   // -4 (MAXNEG)
      [1:2] := 10,   // +1 to +2
      [-3:-1] := 10  // -3 to -1
    };
  }
}


constraint red_op_A_high {
  if (opcode inside {OR, XOR} && red_op_A) {
    A dist {
      0 := 5,
      1 := 40,
      2 := 40,
      3 := 5,
      -4 := 40,
      [-3:-1] := 15
    };
    B == 0;  // B should be 0 when red_op_A is high
  }
}


constraint red_op_B_high {
  if (opcode inside {OR, XOR} && red_op_B) {
    B dist {
      0 := 5,
      1 := 40,
      2 := 40,
      3 := 5,
      -4 := 40,
      [-3:-1] := 15
    };
    A == 0;  // A should be 0 when red_op_B is high
  }
}
```

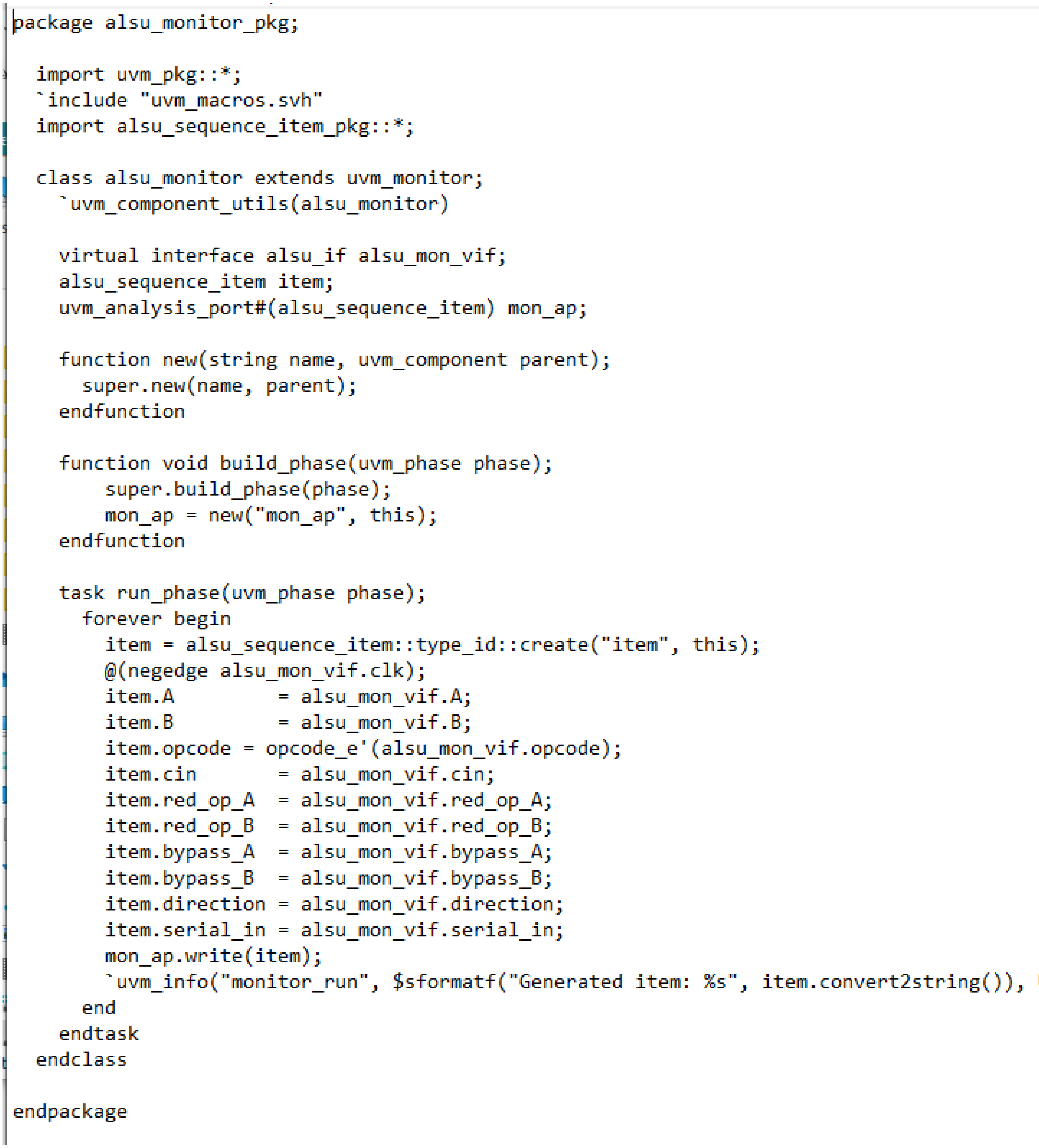
```
    constraint invalid_cases {
      opcode dist {
        INVALID_6 := 5,
        INVALID_7 := 5,
        OR := 50,
        XOR := 50,
        ADD := 50,
        MULT := 50,
        SHIFT := 50,
        ROTATE := 50
      };
      red_op_A dist {1:/5, 0:/95};
      red_op_B dist {1:/5, 0:/95};
    }


    constraint bypass_behavior {
      bypass_A dist {1:/5, 0:/95};
      bypass_B dist {1:/5, 0:/95};
    }

//8th constraint
    rand opcode_e op_arr[6];

constraint constraint8 {
  foreach(op_arr[i])
    op_arr[i] inside {OR, XOR, ADD, MULT, SHIFT, ROTATE};
  unique { op_arr };
}

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

    function string convert2string();
      return $sformatf("A=%0d B=%0d cin=%0b red_op_A=%0b red_op_B=%0b bypass_A=%0b bypass_
                        A, B, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, seri
    endfunction

  endclass

endpackage
```

12. Sequencer



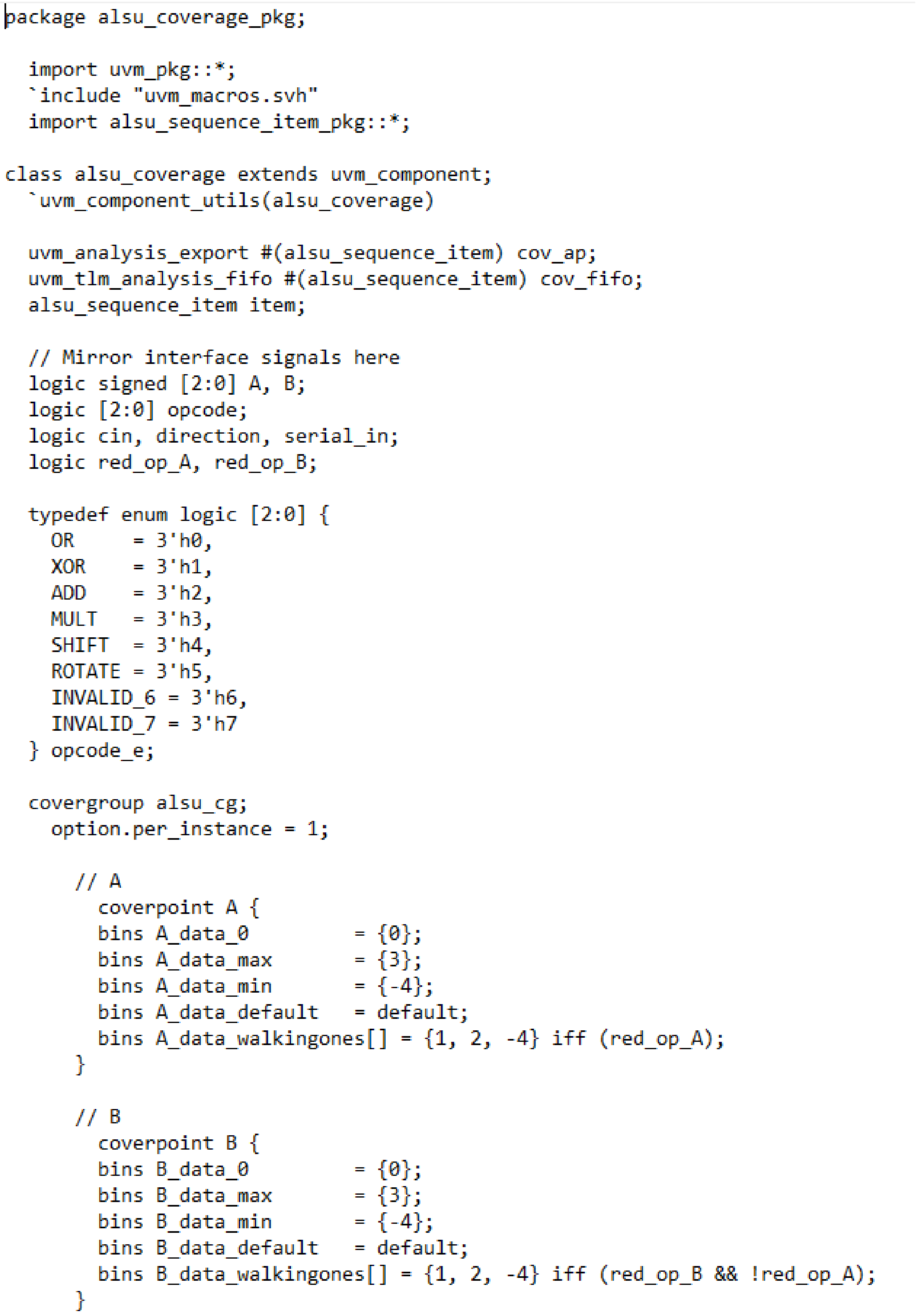
```
package alsu_sequencer_pkg;

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

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

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

  endclass

endpackage
```

13. Monitor



```
package alsu_monitor_pkg;

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

  class alsu_monitor extends uvm_monitor;
    `uvm_component_utils(alsu_monitor)

    virtual interface alsu_if alsu_mon_vif;
    alsu_sequence_item item;
    uvm_analysis_port#(alsu_sequence_item) mon_ap;

    function new(string name, uvm_component parent);
      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);
      forever begin
        item = alsu_sequence_item::type_id::create("item", this);
        @(negedge alsu_mon_vif.clk);
        item.A         = alsu_mon_vif.A;
        item.B         = alsu_mon_vif.B;
        item.opcode = opcode_e'(alsu_mon_vif.opcode);
        item.cin       = alsu_mon_vif.cin;
        item.red_op_A  = alsu_mon_vif.red_op_A;
        item.red_op_B  = alsu_mon_vif.red_op_B;
        item.bypass_A  = alsu_mon_vif.bypass_A;
        item.bypass_B  = alsu_mon_vif.bypass_B;
        item.direction = alsu_mon_vif.direction;
        item.serial_in = alsu_mon_vif.serial_in;
        mon_ap.write(item);
        `uvm_info("monitor_run", $sformatf("Generated item: %s", item.convert2string()),
      end
    endtask
  endclass

endpackage
```

14. Coverage Collecter



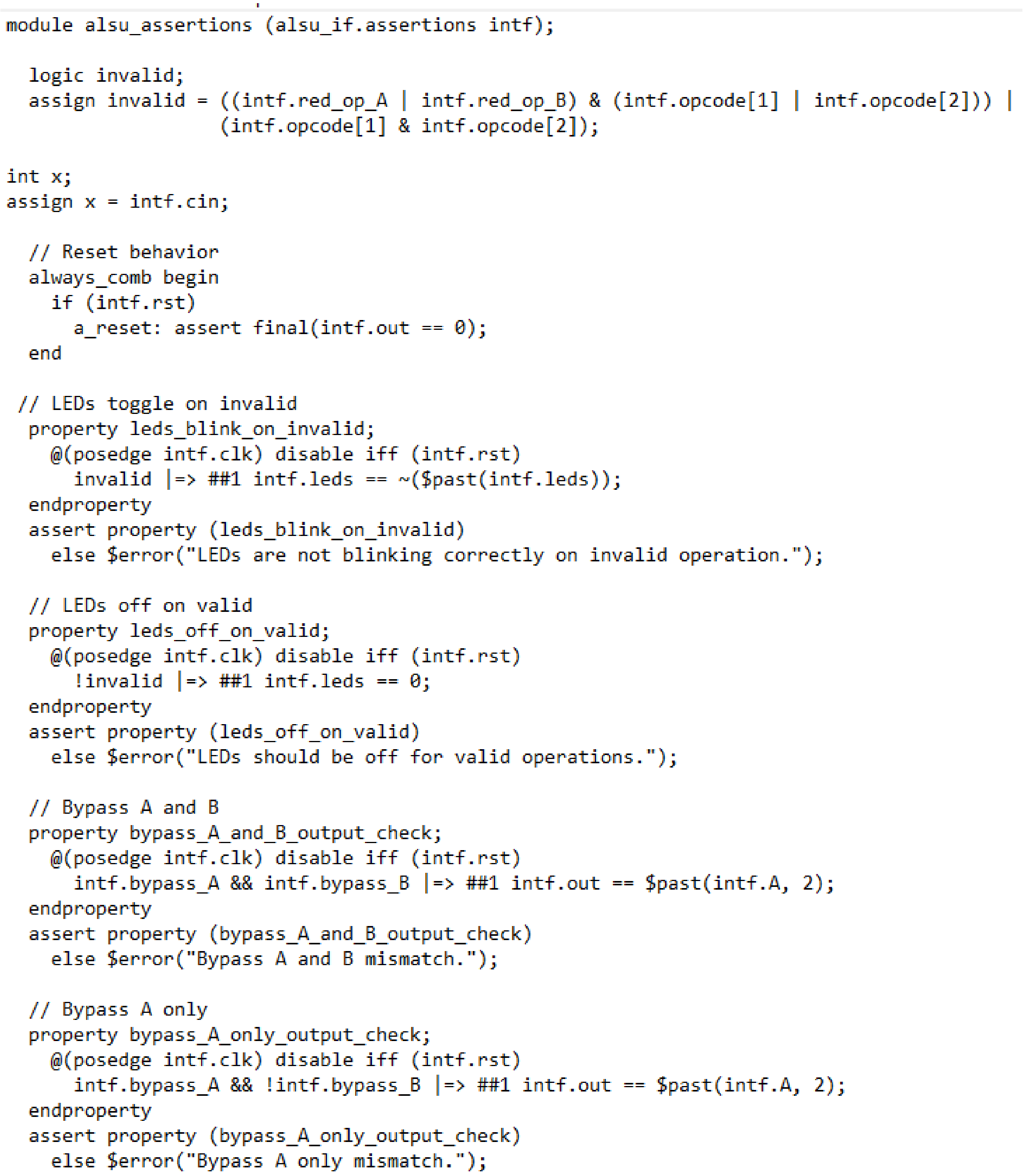
```
package alsu_coverage_pkg;

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

class alsu_coverage extends uvm_component;
  `uvm_component_utils(alsu_coverage)

  uvm_analysis_export #(alsu_sequence_item) cov_ap;
  uvm_tlm_analysis_fifo #(alsu_sequence_item) cov_fifo;
  alsu_sequence_item item;

  // Mirror interface signals here
  logic signed [2:0] A, B;
  logic [2:0] opcode;
  logic cin, direction, serial_in;
  logic red_op_A, red_op_B;

  typedef enum logic [2:0] {
    OR     = 3'h0,
    XOR    = 3'h1,
    ADD    = 3'h2,
    MULT   = 3'h3,
    SHIFT  = 3'h4,
    ROTATE = 3'h5,
    INVALID_6 = 3'h6,
    INVALID_7 = 3'h7
  } opcode_e;

  covergroup alsu_cg;
    option.per_instance = 1;

      // A
        coverpoint A {
        bins A_data_0         = {0};
        bins A_data_max       = {3};
        bins A_data_min       = {-4};
        bins A_data_default   = default;
        bins A_data_walkingones[] = {1, 2, -4} iff (red_op_A);
      }

      // B
        coverpoint B {
        bins B_data_0         = {0};
        bins B_data_max       = {3};
        bins B_data_min       = {-4};
        bins B_data_default   = default;
        bins B_data_walkingones[] = {1, 2, -4} iff (red_op_B && !red_op_A);
      }
      // Opcode
        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);
      }


    coverpoint cin        { bins c_in_vals[] = {0, 1}; }
    coverpoint direction  { bins dir_vals[] = {0, 1}; }
    coverpoint serial_in  { bins shift_vals[] = {0, 1}; }
    coverpoint red_op_A   { bins active = {1}; }
    coverpoint red_op_B   { bins active = {1}; }

//1
    cross A, B iff (opcode inside {ADD, MULT}) {
      bins cross_arith = binsof(A) intersect {3, -4, 0} &&
                         binsof(B) intersect {3, -4, 0};
      option.cross_auto_bin_max = 0;
    }

//2
  cross opcode, cin {bins cross_add_cin = binsof(opcode) intersect {ADD};
  option.cross_auto_bin_max = 0;
}

//3
  cross opcode, direction {bins cross_shift_dir = binsof(opcode) intersect {SHIFT, ROTATE
  option.cross_auto_bin_max = 0;
}

//4
  cross opcode, serial_in {bins cross_shift_serialin = binsof(opcode) intersect {SHIFT};
  option.cross_auto_bin_max = 0;
}

//5
  cross  A, B iff( opcode == OR || opcode == XOR ){
    bins cross_redA = binsof(A.A_data_walkingones) && binsof(B.B_data_0);
    option.cross_auto_bin_max = 0;
  }
```

```
//6
  cross  A, B iff( opcode == OR || opcode == XOR ){
    bins cross_redB = binsof(B.B_data_walkingones) && binsof(A.A_data_0);
    option.cross_auto_bin_max = 0;
  }

//7
cross opcode, red_op_A, red_op_B {
    bins invalid_redA =
        binsof(opcode) intersect {ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7} &&
        binsof(red_op_A) intersect {1};
    bins invalid_redB =
        binsof(opcode) intersect {ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7} &&
        binsof(red_op_B) intersect {1};
        option.cross_auto_bin_max = 0;
  }

  endgroup

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    cov_ap = new("cov_ap", this);
    cov_fifo = new("cov_fifo", this);

  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    cov_ap.connect(cov_fifo.analysis_export);
  endfunction
  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
      cov_fifo.get(item);

      A           = item.A;
      B           = item.B;
      opcode      = item.opcode;
      cin         = item.cin;
      direction   = item.direction;
      serial_in   = item.serial_in;
      red_op_A    = item.red_op_A;
      red_op_B    = item.red_op_B;

      if (!item.rst && !item.bypass_A && !item.bypass_B) begin
        alsu_cg.sample();
      end
    end
  endtask

endclass

endpackage
```

15. Scoreboard

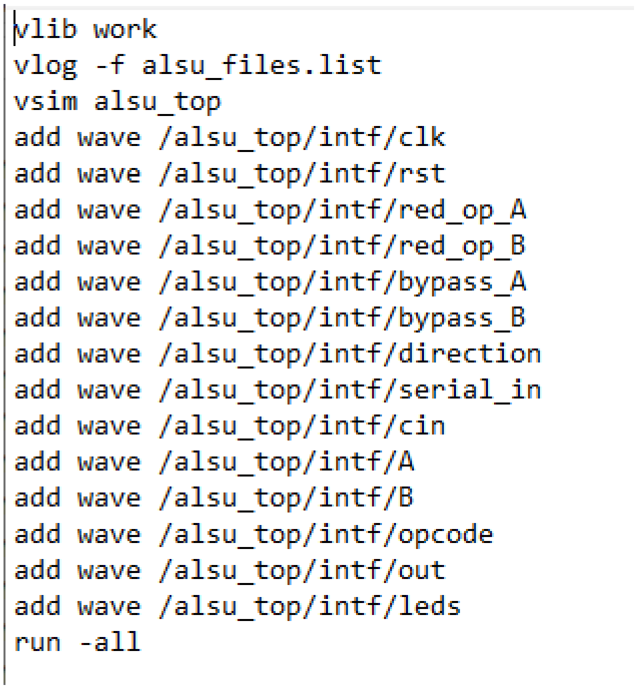
```
package alsu_scoreboard_pkg;

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

  class alsu_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(alsu_scoreboard)

    uvm_analysis_export #(alsu_sequence_item) sb_export;
    uvm_tlm_analysis_fifo #(alsu_sequence_item) sb_fifo;

    alsu_sequence_item seq_item_sb;
    logic signed [5:0] golden_model_output;

    int correct_count = 0;
    int error_count = 0;

    function new(string name = "alsu_scoreboard", 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);
        ref_model(seq_item_sb);

        if (seq_item_sb.out != golden_model_output) begin
          `uvm_error("run_phase", $sformatf("Mismatch: DUT=%0d, REF=%0d",
                                            seq_item_sb.out, golden_model_output))
          error_count++;
        end
        else begin
          correct_count++;
        end
      end
    endtask
    function void report_phase(uvm_phase phase);
      super.report_phase(phase);
      `uvm_info("report_phase", $sformatf("Correct transactions: %0d", correct_count), UVM
      `uvm_info("report_phase", $sformatf("Failed transactions: %0d", error_count), UVM_ME
    endfunction

task ref_model(alsu_sequence_item chk);
  chk.out_old = chk.out;
  if (chk.rst) begin
    golden_model_output = 0;
  end

  else begin
    if (chk.bypass_A && chk.bypass_B)
      golden_model_output = chk.A;
    else if (chk.bypass_A)
      golden_model_output = chk.A;
    else if (chk.bypass_B)
      golden_model_output = chk.B;
    else if ((chk.opcode == 3'b110 || chk.opcode == 3'b111) ||
      ((chk.red_op_A || chk.red_op_B) && !(chk.opcode == 3'b000 || chk.opcode == 3'b001)))
      golden_model_output = 0;

    else begin
      case (chk.opcode)
        3'h0: begin  // OR Operation
          if (chk.red_op_A && chk.red_op_B)
            golden_model_output = |chk.A;
          else if (chk.red_op_A)
            golden_model_output = |chk.A;
          else if (chk.red_op_B)
            golden_model_output = |chk.B;
          else
            golden_model_output = chk.A | chk.B;
        end
        3'h1: begin  // XOR Operation
          if (chk.red_op_A && chk.red_op_B)
            golden_model_output = ^chk.A;
          else if (chk.red_op_A)
            golden_model_output = ^chk.A;
          else if (chk.red_op_B)
            golden_model_output = ^chk.B;
          else
            golden_model_output = chk.A ^ chk.B;
        end
        3'h2: golden_model_output = chk.A + chk.B + chk.cin;  // ADD
        3'h3: golden_model_output = chk.A * chk.B;            // MULTIPLY
        3'h4: begin  // SHIFT
          if (chk.direction)
            golden_model_output = {chk.out_old[4:0], chk.serial_in};  // Shift left
          else
            golden_model_output = {chk.serial_in, chk.out_old[5:1]};  // Shift right
        end
        3'h5: begin  // ROTATE
          if (chk.direction)
            golden_model_output = {chk.out_old[4:0], chk.out_old[5]};  // Rotate left
          else
            golden_model_output = {chk.out_old[0], chk.out_old[5:1]};  // Rotate right
        end
        default: golden_model_output = 0;
      endcase
    end
  end

endtask

endclass

endpackage
```

16. Assertions

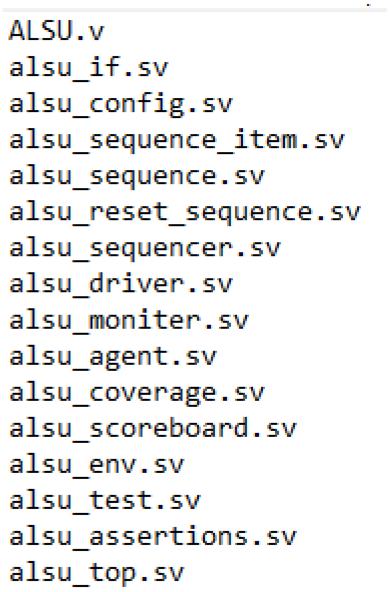


```
module alsu_assertions (alsu_if.assertions intf);

  logic invalid;
  assign invalid = ((intf.red_op_A | intf.red_op_B) & (intf.opcode[1] | intf.opcode[2])) |
                   (intf.opcode[1] & intf.opcode[2]);

int x;
assign x = intf.cin;

  // Reset behavior
  always_comb begin
    if (intf.rst)
      a_reset: assert final(intf.out == 0);
  end

 // LEDs toggle on invalid
  property leds_blink_on_invalid;
    @(posedge intf.clk) disable iff (intf.rst)
      invalid |=> ##1 intf.leds == ~($past(intf.leds));
  endproperty
  assert property (leds_blink_on_invalid)
    else $error("LEDs are not blinking correctly on invalid operation.");

  // LEDs off on valid
  property leds_off_on_valid;
    @(posedge intf.clk) disable iff (intf.rst)
      !invalid |=> ##1 intf.leds == 0;
  endproperty
  assert property (leds_off_on_valid)
    else $error("LEDs should be off for valid operations.");

  // Bypass A and B
  property bypass_A_and_B_output_check;
    @(posedge intf.clk) disable iff (intf.rst)
      intf.bypass_A && intf.bypass_B |=> ##1 intf.out == $past(intf.A, 2);
  endproperty
  assert property (bypass_A_and_B_output_check)
    else $error("Bypass A and B mismatch.");

  // Bypass A only
  property bypass_A_only_output_check;
    @(posedge intf.clk) disable iff (intf.rst)
      intf.bypass_A && !intf.bypass_B |=> ##1 intf.out == $past(intf.A, 2);
  endproperty
  assert property (bypass_A_only_output_check)
    else $error("Bypass A only mismatch.");
  // Bypass B only
  property bypass_B_only_output_check;
    @(posedge intf.clk) disable iff (intf.rst)
      !intf.bypass_A && intf.bypass_B |=> ##1 intf.out == $past(intf.B, 2);
  endproperty
  assert property (bypass_B_only_output_check)
    else $error("Bypass B only mismatch.");

  // Addition
  property addition_output_check;
    @(posedge intf.clk) disable iff (intf.rst || intf.red_op_A || intf.red_op_B || intf.bypass_B || intf.bypass_A)
      (intf.opcode == 2) |-> ##2
        intf.out == ($past(intf.A, 2) + $past(intf.B, 2) + $past(x, 2));
  endproperty
  assert property (addition_output_check)
    else $error("Addition mismatch.");

  // Multiplication
  property multiply_output_check;
    @(posedge intf.clk) disable iff (intf.rst || intf.red_op_A || intf.red_op_B || intf.bypass_B || intf.bypass_A)
      (intf.opcode == 3) |-> ##2
        intf.out == ($past(intf.A, 2) * $past(intf.B, 2));
  endproperty
  assert property (multiply_output_check)
    else $error("Multiply mismatch.");


endmodule
```

17. Do file & alsu_files

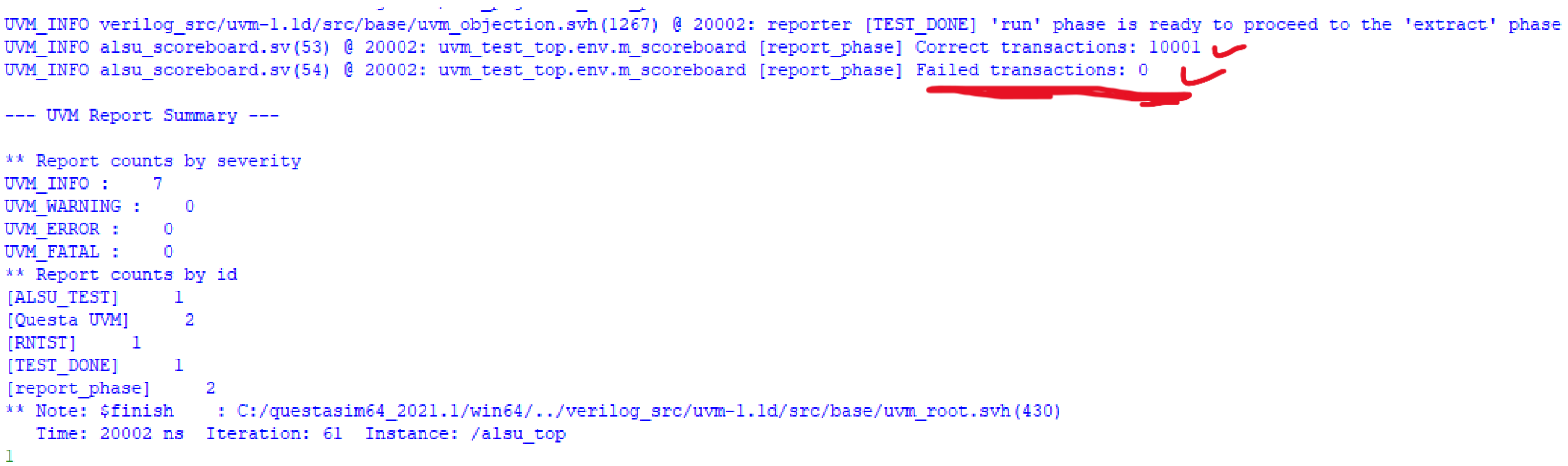


```
vlib work
vlog -f alsu_files.list
vsim alsu_top
add wave /alsu_top/intf/clk
add wave /alsu_top/intf/rst
add wave /alsu_top/intf/red_op_A
add wave /alsu_top/intf/red_op_B
add wave /alsu_top/intf/bypass_A
add wave /alsu_top/intf/bypass_B
add wave /alsu_top/intf/direction
add wave /alsu_top/intf/serial_in
add wave /alsu_top/intf/cin
add wave /alsu_top/intf/A
add wave /alsu_top/intf/B
add wave /alsu_top/intf/opcode
add wave /alsu_top/intf/out
add wave /alsu_top/intf/leds
run -all
```

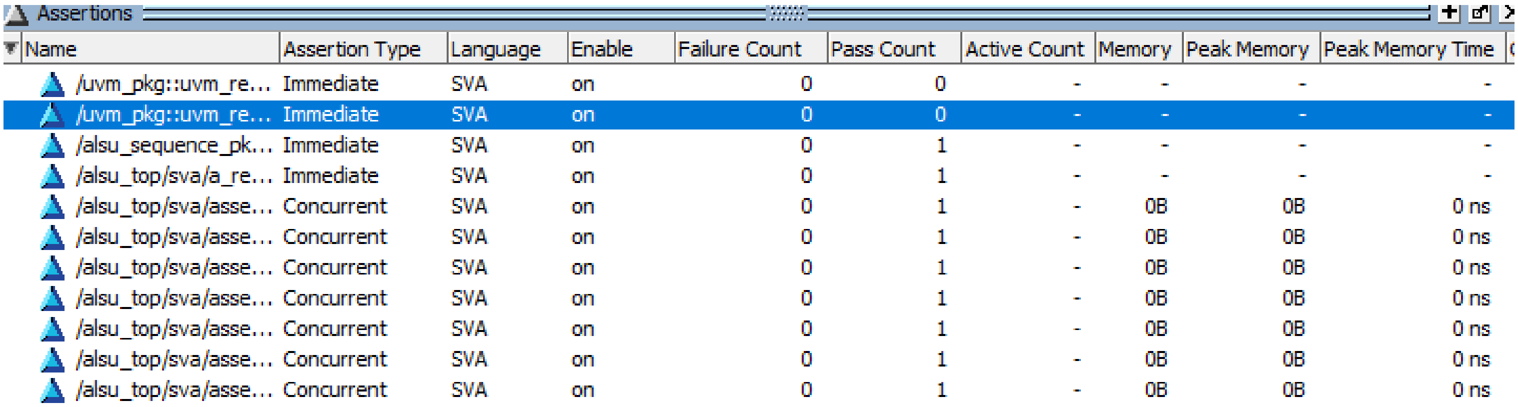


```
ALSU.v
alsu_if.sv
alsu_config.sv
alsu_sequence_item.sv
alsu_sequence.sv
alsu_reset_sequence.sv
alsu_sequencer.sv
alsu_driver.sv
alsu_moniter.sv
alsu_agent.sv
alsu_coverage.sv
alsu_scoreboard.sv
alsu_env.sv
alsu_test.sv
alsu_assertions.sv
alsu_top.sv
```

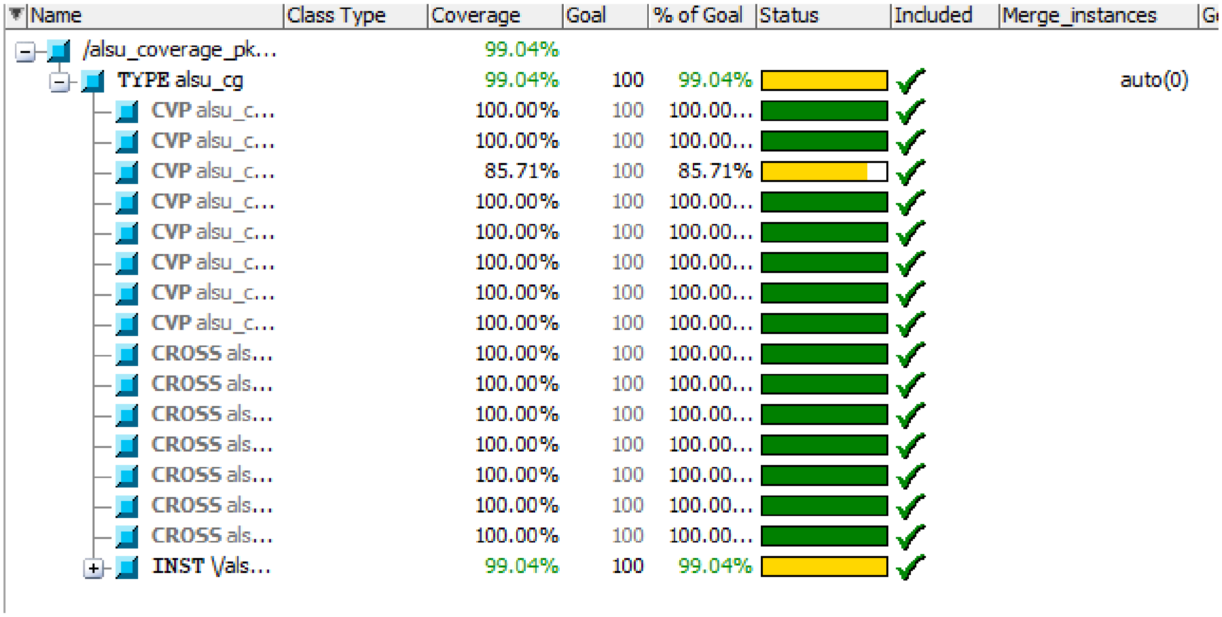
18. Transcript & Assertions & Coverage



```
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 20002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO alsu_scoreboard.sv(53) @ 20002: uvm_test_top.env.m_scoreboard [report_phase] Correct transactions: 10001
UVM_INFO alsu_scoreboard.sv(54) @ 20002: uvm_test_top.env.m_scoreboard [report_phase] Failed transactions: 0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :    7
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[ALSU_TEST]     1
[Questa UVM]     2
[RNTST]     1
[TEST_DONE]     1
[report_phase]     2
** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
   Time: 20002 ns  Iteration: 61  Instance: /alsu_top
1
```

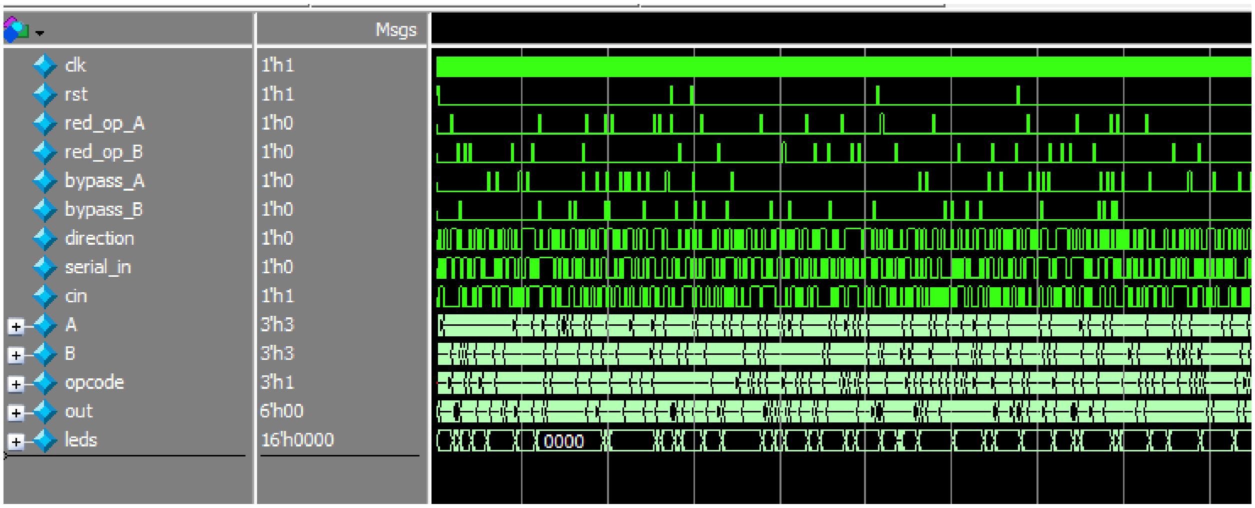


Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-
/alsu_sequence_pk...	Immediate	SVA	on	0	1	-	-	-	-
/alsu_top/sva/a_re...	Immediate	SVA	on	0	1	-	-	-	-
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns
/alsu_top/sva/asse...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns

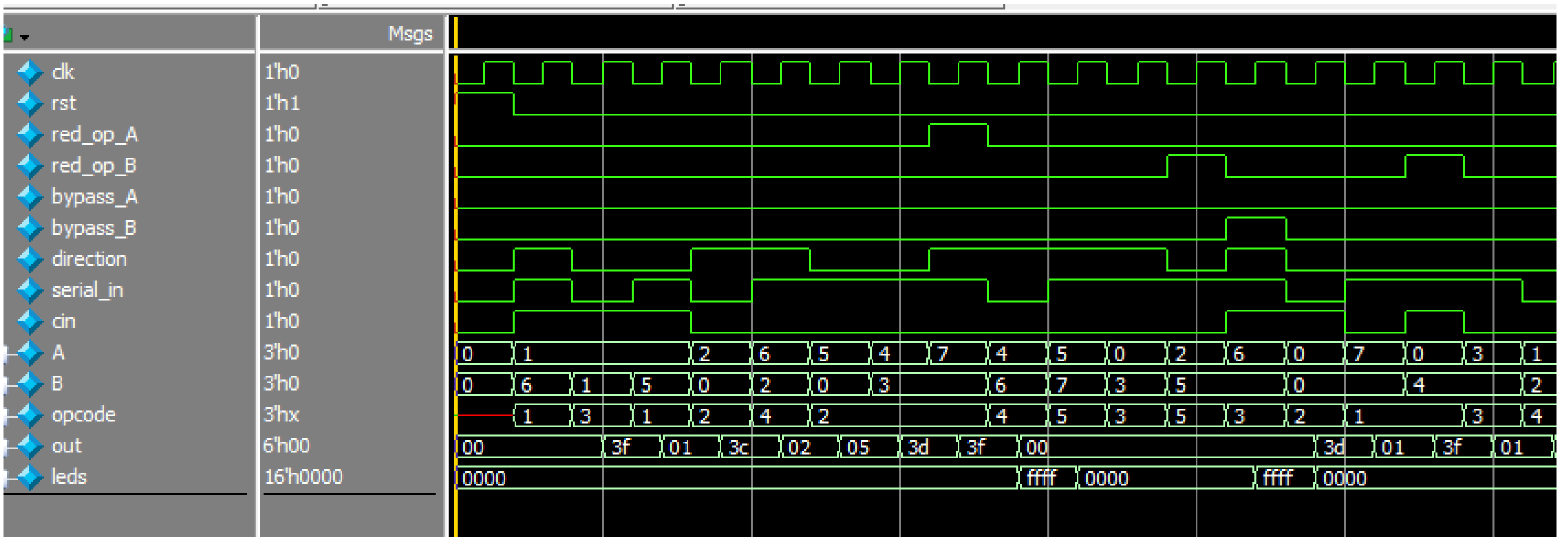


Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances
/alsu_coverage_pk...		99.04%					
TYPE alsu_cg		99.04%	100	99.04%	🟨	✓	auto(0)
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		85.71%	100	85.71%	🟨	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CVP alsu_c...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
CROSS als...		100.00%	100	100.00...	🟩	✓	
INST \/als...		99.04%	100	99.04%	🟨	✓	

19. Waveform



	Msgs
clk	1'h1
rst	1'h1
red_op_A	1'h0
red_op_B	1'h0
bypass_A	1'h0
bypass_B	1'h0
direction	1'h0
serial_in	1'h0
cin	1'h1
A	3'h3
B	3'h3
opcode	3'h1
out	6'h00
leds	16'h0000



	Msgs
clk	1'h0
rst	1'h1
red_op_A	1'h0
red_op_B	1'h0
bypass_A	1'h0
bypass_B	1'h0
direction	1'h0
serial_in	1'h0
cin	1'h0
A	3'h0
B	3'h0
opcode	3'hx
out	6'h00
leds	16'h0000