Design file:

```
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 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;
```

```verilog
      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 <= (FULL_ADDER == "ON")? A_reg + B_reg + cin : A_reg +
B_reg;
        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
```

Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU1	When rst is asserted high, out should be 0 and leds should be 0.	Directed at the start of simulation then randomized with constraint (most of the time low).	-	Checker ensures out = 0, leds = 0.
ALSU2	when opcode is ADD, then out should perform addition on ports A and B taking cin.	Randomized under constraints on the A and B	Cover MAXPOS, MAXNEG, ZERO as well as default for all other values	Checker ensures out = A + B + cin
ALSU3	when opcode is MUL, then out should perform the multiplication on ports A and B	Randomized under constraints on the A and B	Cover MAXPOS, MAXNEG, ZERO as well as default for all other values	Checker ensures out = A * B
ALSU4	when opcode is OR, then out should perform the OR operation on ports A and B if reduction_A, reduction B are low	Randomized without constraints on A or B.	-	Checker ensures out = A | B
ALSU5	when opcode is OR & any of the inputs reduction_A or reduction_B is high.	Randomized under constraints on the A and B	Cover the walking ones values for A and B	Checker ensures out = (INPUT_PRIORITY == ‘A’) ? |A : |B
ALSU6	when opcode is XOR, then out should perform the XOR operation on ports A and B if reduction_A, reduction B are low.	Randomized without constraints on A or B	-	Checker ensures out = A ^ B

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU7	when opcode is XOR & any of the inputs reduction_A or reduction_B is high.	Randomized under constraints on the A and B	Cover the walking ones values for A and B	Checker ensures out = (INPUT_PRIORITY == ‘A’) ? ^A : ^B
ALSU8	when opcode is SHIFT then the output will shift right or left based on the direction input	Randomized without constraints on A or B	Cover the SHIFT opcode	Checker ensures out = (Direction == ‘left’) ? out << : out >>
ALSU9	when opcode is ROTATE then the output will rotate right or left based on the direction input	Randomized without constraints on A or B.	Cover the ROTATE opcode	Checker ensures out = (Direction == ‘left’) ? out << : out >>
ALSU10	When invalid cases exist without bypass, out should be low and leds should blink	Randomized under constraints (valid cases Happen frequently)	Cover the illegal opcode	Checker ensures out = 0, leds blinking
ALSU11	If the bypass inputs are high, then the output will bypass port A or B	Randomized under constraints	-	Checker ensures out = A or B

Interface:

```
interface alsu_if(clk);
    input bit clk;
    logic rst;
    logic cin;
    logic red_op_A;
    logic red_op_B;
    logic bypass_A;
    logic bypass_B;
    logic direction;
    logic serial_in;
    logic [2:0] opcode;
    logic signed [2:0] A;
    logic signed [2:0] B;
    logic [15:0] leds;
    logic signed [5:0] out;
endinterface: alsu_if
```

Testbench file:

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

module alsu_tb();
    // Clock generation
    bit clk;

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

    // Instantiate the interface
    alsu_if alsuif(clk);

    // Instantiate the DUT
    ALSU #(
        .INPUT_PRIORITY(INPUT_PRIORITY),
        .FULL_ADDER(FULL_ADDER)
    ) DUT (
        .clk(clk),
        .rst(alsuif.rst),
        .cin(alsuif.cin),
        .red_op_A(alsuif.red_op_A),
        .red_op_B(alsuif.red_op_B),
        .bypass_A(alsuif.bypass_A),
        .bypass_B(alsuif.bypass_B),
        .direction(alsuif.direction),
        .serial_in(alsuif.serial_in),
        .opcode(alsuif.opcode),
        .A(alsuif.A),
        .B(alsuif.B),
        .leds(alsuif.leds),
        .out(alsuif.out)
    );

    bind ALSU alsu_sva #(
        .INPUT_PRIORITY(INPUT_PRIORITY),
        .FULL_ADDER(FULL_ADDER)
    ) ASSERT (
        .clk(clk),
        .rst(alsuif.rst),
        .cin(alsuif.cin),
        .red_op_A(alsuif.red_op_A),
```

```
        .red_op_B(alsuif.red_op_B),
        .bypass_A(alsuif.bypass_A),
        .bypass_B(alsuif.bypass_B),
        .direction(alsuif.direction),
        .serial_in(alsuif.serial_in),
        .opcode(alsuif.opcode),
        .A(alsuif.A),
        .B(alsuif.B),
        .leds(alsuif.leds),
        .out(alsuif.out)
    );

    // Run the test
    initial begin
        uvm_config_db #(virtual alsu_if)::set(null, "uvm_test_top",
"ALSU_VIF", alsuif);
        run_test("alsu_test");
    end

endmodule
```

Test package:

```
package alsu_test_pkg;
import uvm_pkg::*;
import alsu_env_pkg::*;
import alsu_config_pkg::*;
import alsu_reset_seq_pkg::*;
import alsu_main_seq_pkg::*;
`include "uvm_macros.svh"

class alsu_test extends uvm_test;
    `uvm_component_utils(alsu_test)

    alsu_env env;
    alsu_config alsu_cfg;
    alsu_reset_seq reset_seq;
    alsu_main_seq main_seq;

    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");
        reset_seq =  alsu_reset_seq::type_id::create("reset_seq");
        main_seq =  alsu_main_seq::type_id::create("main_seq");

        if (!uvm_config_db #(virtual alsu_if)::get(this, "", "ALSU_VIF",
alsu_cfg.alsu_vif)) begin
            `uvm_fatal("BUILD_PHASE", "Test - Unable to get virtual interface
of the ALSU")
        end

        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 sequence
        reset_seq.start(env.agent.sequencer);
        `uvm_info("RUN_PHASE", "Reset Asserted", UVM_LOW)
        `uvm_info("RUN_PHASE", "Reset Deasserted", UVM_LOW)

        // main sequence
```

```
        `uvm_info("RUN_PHASE", "Stimulus Generation Started", UVM_LOW)
        main_seq.start(env.agent.sequencer);
        `uvm_info("RUN_PHASE", "Stimulus Generation Ended", UVM_LOW)

        phase.drop_objection(this);
    endtask
endclass: alsu_test

endpackage: alsu_test_pkg
```

Env. package:

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

class alsu_env extends uvm_env;
    `uvm_component_utils(alsu_env)

    alsu_agent agent;
    alsu_scoreboard scoreboard;
    alsu_coverage 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);

        agent = alsu_agent::type_id::create("agent", this);
        scoreboard = alsu_scoreboard::type_id::create("scoreboard", this);
        coverage = alsu_coverage::type_id::create("coverage", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        agent.analysis_port.connect(scoreboard.analysis_export);
        agent.analysis_port.connect(coverage.analysis_export);
    endfunction
endclass: alsu_env

endpackage: alsu_env_pkg
```

Agent package:

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

class alsu_agent extends uvm_agent;
    `uvm_component_utils(alsu_agent)

    alsu_sequencer sequencer;
    alsu_driver driver;
    alsu_monitor monitor;
    alsu_config alsu_cfg;
    uvm_analysis_port #(alsu_seq_item) analysis_port;

    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))
begin
            `uvm_fatal("BUILD_PHASE", "Agent - Unable to get the configuration
abject")
        end

        sequencer = alsu_sequencer::type_id::create("sequencer", this);
        driver = alsu_driver::type_id::create("driver", this);
        monitor = alsu_monitor::type_id::create("monitor", this);

        analysis_port = new("analysis_port", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        driver.alsu_vif = alsu_cfg.alsu_vif;
        monitor.alsu_vif = alsu_cfg.alsu_vif;

        driver.seq_item_port.connect(sequencer.seq_item_export);
        monitor.analysis_port.connect(analysis_port);
    endfunction
endclass: alsu_agent
endpackage: alsu_agent_pkg
```

Driver package:

```
package alsu_driver_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
import alsu_shared_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 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
            seq_item = alsu_seq_item::type_id::create("seq_item");
            seq_item_port.get_next_item(seq_item);
            alsu_vif.rst = seq_item.rst;
            alsu_vif.A = seq_item.A;
            alsu_vif.B = seq_item.B;
            alsu_vif.opcode = opcode_e'(seq_item.opcode);
            alsu_vif.cin = seq_item.cin;
            alsu_vif.red_op_A = seq_item.red_op_A;
            alsu_vif.red_op_B = seq_item.red_op_B;
            alsu_vif.bypass_A = seq_item.bypass_A;
            alsu_vif.bypass_B = seq_item.bypass_B;
            alsu_vif.direction = direction_e'(seq_item.direction);
            alsu_vif.serial_in = seq_item.serial_in;
            @(negedge alsu_vif.clk);
            seq_item_port.item_done();
            `uvm_info("RUN_PHASE", seq_item.convert2string_stimulus(),
UVM_HIGH)
        end
    endtask
endclass: alsu_driver

endpackage: alsu_driver_pkg
```

Monitor package:

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

class alsu_monitor extends uvm_monitor;
    `uvm_component_utils(alsu_monitor)

    virtual alsu_if alsu_vif;
    alsu_seq_item seq_item;
    uvm_analysis_port #(alsu_seq_item) analysis_port;

    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);
        analysis_port = new("analysis_port", this);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = alsu_seq_item::type_id::create("seq_item");
            @(negedge alsu_vif.clk);
            seq_item.rst = alsu_vif.rst;
            seq_item.A = alsu_vif.A;
            seq_item.B = alsu_vif.B;
            seq_item.opcode = opcode_e'(alsu_vif.opcode);
            seq_item.cin = alsu_vif.cin;
            seq_item.red_op_A = alsu_vif.red_op_A;
            seq_item.red_op_B = alsu_vif.red_op_B;
            seq_item.bypass_A = alsu_vif.bypass_A;
            seq_item.bypass_B = alsu_vif.bypass_B;
            seq_item.direction = direction_e'(alsu_vif.direction);
            seq_item.serial_in = alsu_vif.serial_in;
            seq_item.out = alsu_vif.out;
            seq_item.leds = alsu_vif.leds;

            analysis_port.write(seq_item);
            `uvm_info("RUN_PHASE", seq_item.convert2string(), UVM_HIGH)
        end
    endtask
endclass: alsu_monitor
endpackage: alsu_monitor_pkg
```

Sequencer package:

```
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: alsu_sequencer

endpackage: alsu_sequencer_pkg
```

Coverage collector package:

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

class alsu_coverage extends uvm_component;
    `uvm_component_utils(alsu_coverage)

    uvm_analysis_export #(alsu_seq_item) analysis_export;
    uvm_tlm_analysis_fifo #(alsu_seq_item) analysis_fifo;
    alsu_seq_item seq_item;

    covergroup cvg;
        ADD_MULT_A_cp: coverpoint seq_item.A{
            bins A_data_0 = {0};
            bins A_data_max = {MAXPOS};
            bins A_data_min = {MAXNEG};
            bins A_data_default = default;
        }

        RED_A_cp: coverpoint seq_item.A iff (seq_item.red_op_A){
            bins A_data_walkingones[] = {3'sb001,3'sb010,3'sb100};
        }

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

        RED_B_cp: coverpoint seq_item.B iff (seq_item.red_op_B &&
(!seq_item.red_op_A)){
            bins B_data_walkingones[] = {3'sb001,3'sb010,3'sb100};
        }

        opcode_cp: coverpoint seq_item.opcode{
            bins bins_shift[] = {SHIFT, ROTATE};
            bins bins_arith[] = {ADD, MULT};
            bins bins_bitwise[] = {OR, XOR};
            illegal_bins bins_invalid = {INVALID_6, INVALID_7};
        }

        opcode_shift_cp: coverpoint seq_item.opcode{
            option.weight = 0;
            bins bins_shift[] = {SHIFT, ROTATE};
```

```
        }

        opcode_arith_cp: coverpoint seq_item.opcode{
            option.weight = 0;
            bins bins_arith[] = {ADD, MULT};
        }

        opcode_bitwise_cp: coverpoint seq_item.opcode{
            option.weight = 0;
            bins bins_arith[] = {OR, XOR};
        }

        opcode_not_bitwise_cp: coverpoint seq_item.opcode{
            option.weight = 0;
            bins bins_not_bitwise[] = {[ADD:$]};
        }

        cin_cp: coverpoint seq_item.cin{
            option.weight = 0;
        }

        direction_cp: coverpoint seq_item.direction{
            option.weight = 0;
        }

        serial_in_cp: coverpoint seq_item.serial_in{
            option.weight = 0;
        }

        red_op_A_cp: coverpoint seq_item.red_op_A{
            option.weight = 0;
        }

        red_op_B_cp: coverpoint seq_item.red_op_B{
            option.weight = 0;
        }

        ADD_MULT_cross: cross ADD_MULT_A_cp, ADD_MULT_B_cp, opcode_arith_cp;

        ADD_cin_cross: cross cin_cp, opcode_arith_cp{
            ignore_bins bins_MULT = binsof(opcode_arith_cp) intersect {MULT};
        }

        SHIFT_direction_cross: cross direction_cp, opcode_shift_cp{
            ignore_bins bins_ROTATE = binsof(opcode_shift_cp) intersect
{ROTATE};
        }
```

```
        SHIFT_serial_in_cross: cross serial_in_cp, opcode_shift_cp{
            ignore_bins bins_ROTATE = binsof(opcode_shift_cp) intersect
{ROTATE};
        }

        red_op_A_cross: cross RED_A_cp, ADD_MULT_B_cp, opcode_bitwise_cp iff
(seq_item.red_op_A){
            ignore_bins B_data_max = binsof(ADD_MULT_B_cp.B_data_max);
            ignore_bins B_data_min = binsof(ADD_MULT_B_cp.B_data_min);
        }

        red_op_B_cross: cross RED_B_cp, ADD_MULT_A_cp, opcode_bitwise_cp iff
(seq_item.red_op_B && (!seq_item.red_op_A)){
            ignore_bins A_data_max = binsof(ADD_MULT_A_cp.A_data_max);
            ignore_bins A_data_min = binsof(ADD_MULT_A_cp.A_data_min);
        }

        invalid_red_op_A_cross: cross red_op_A_cp, opcode_not_bitwise_cp{
            ignore_bins red_op_A_0 = binsof(red_op_A_cp) intersect {0};
            illegal_bins red_op_A_1 = binsof(red_op_A_cp) intersect {1};
        }

        invalid_red_op_B_cross: cross red_op_B_cp, opcode_not_bitwise_cp{
            ignore_bins red_op_B_0 = binsof(red_op_B_cp) intersect {0};
            illegal_bins red_op_B_1 = binsof(red_op_B_cp) intersect {1};
        }
    endgroup

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        analysis_export = new("analysis_export", this);
        analysis_fifo = new("analysis_fifo", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        analysis_export.connect(analysis_fifo.analysis_export);
    endfunction

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

```
        forever begin
            analysis_fifo.get(seq_item);
            sample_cvg(seq_item);
        end
    endtask

    task sample_cvg(alsu_seq_item seq_item);
        if((!seq_item.rst) || (!seq_item.bypass_A) || (!seq_item.bypass_B))
begin
            cvg.sample();
        end
    endtask
endclass: alsu_coverage

endpackage: alsu_coverage_pkg
```

Scoreboard package:

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

class alsu_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(alsu_scoreboard)

    uvm_analysis_export #(alsu_seq_item) analysis_export;
    uvm_tlm_analysis_fifo #(alsu_seq_item) analysis_fifo;
    alsu_seq_item seq_item;

    logic cin_reg;
    logic red_op_A_reg;
    logic red_op_B_reg;
    logic bypass_A_reg;
    logic bypass_B_reg;
    direction_e direction_reg;
    logic serial_in_reg;
    opcode_e opcode_reg;
    logic signed [2:0] A_reg;
    logic signed [2:0] B_reg;
    logic [15:0] leds_ref;
    logic signed [5:0] out_ref;

    int error_count = 0;
    int correct_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);

        analysis_export = new("analysis_export", this);
        analysis_fifo = new("analysis_fifo", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        analysis_export.connect(analysis_fifo.analysis_export);
    endfunction
```

```
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = alsu_seq_item::type_id::create("seq_item");
            analysis_fifo.get(seq_item);
            ref_model(seq_item);
            check_result(seq_item);
        end
    endtask

    task check_result(alsu_seq_item seq_item);
        if ((out_ref !== seq_item.out) || (leds_ref !== seq_item.leds)) begin
            `uvm_error("RUN_PHASE", $sformatf("Comparsion failed, Transaction
received from DUT: %s While the reference out = 0b%0b, leds = 0x%0h ",
seq_item.convert2string(), out_ref, leds_ref));
            error_count++;
        end else begin
            `uvm_info("RUN_PHASE", $sformatf("Comparsion successed,
Transaction received from DUT: %s ", seq_item.convert2string()), UVM_HIGH);
            correct_count++;
        end
    endtask

    task ref_model(alsu_seq_item seq_item);
        logic invalid_red_op, invalid_opcode, invalid;

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

        if (seq_item.rst) begin
            leds_ref = 16'h0;
        end else begin
            if (invalid) begin
                leds_ref = ~leds_ref;
            end else begin
                leds_ref = 16'h0;
            end
        end

        if (seq_item.rst) begin
            out_ref = 0;
        end else begin
            if (bypass_A_reg && bypass_B_reg) begin
                out_ref = (INPUT_PRIORITY == "A") ? A_reg : B_reg;
            end else if (bypass_A_reg) begin
                out_ref = A_reg;
```

```
            end else if (bypass_B_reg) begin
                out_ref = B_reg;
            end else if (invalid) begin
                out_ref = 0;
            end else begin
                case (opcode_reg)
                    OR: begin
                        if (red_op_A_reg && red_op_B_reg) begin
                            out_ref = (INPUT_PRIORITY == "A") ? (|A_reg) :
(|B_reg);
                        end else if (red_op_A_reg) begin
                            out_ref = |A_reg;
                        end else if (red_op_B_reg) begin
                            out_ref = |B_reg;
                        end else begin
                            out_ref = A_reg | B_reg;
                        end
                    end
                    XOR: begin
                        if (red_op_A_reg && red_op_B_reg) begin
                            out_ref = (INPUT_PRIORITY == "A") ? (^A_reg) :
(^B_reg);
                        end else if (red_op_A_reg) begin
                            out_ref = ^A_reg;
                        end else if (red_op_B_reg) begin
                            out_ref = ^B_reg;
                        end else begin
                            out_ref = A_reg ^ B_reg;
                        end
                    end
                    ADD: begin
                        if (FULL_ADDER == "ON") begin
                            out_ref = A_reg + B_reg + cin_reg;
                        end else begin
                            out_ref = A_reg + B_reg;
                        end
                    end
                    MULT: out_ref = A_reg * B_reg;
                    SHIFT: begin
                        if (direction_reg) begin
                            out_ref = {out_ref[4:0], serial_in_reg};
                        end else begin
                            out_ref = {serial_in_reg, out_ref[5:1]};
                        end
                    end
                    ROTATE: begin
                        if (direction_reg) begin
                            out_ref = {out_ref[4:0], out_ref[5]};
```

```
                        end else begin
                            out_ref = {out_ref[0], out_ref[5:1]};
                        end
                    end
                    default: out_ref = out_ref;
                endcase
            end
        end

        if (seq_item.rst) begin
            cin_reg = 0;
            red_op_A_reg = 0;
            red_op_B_reg = 0;
            bypass_A_reg = 0;
            bypass_B_reg = 0;
            direction_reg = direction_e'(0);
            serial_in_reg = 0;
            opcode_reg = opcode_e'(0);
            A_reg = 0;
            B_reg = 0;
        end else begin
            cin_reg = seq_item.cin;
            red_op_A_reg = seq_item.red_op_A;
            red_op_B_reg = seq_item.red_op_B;
            bypass_A_reg = seq_item.bypass_A;
            bypass_B_reg = seq_item.bypass_B;
            direction_reg = seq_item.direction;
            serial_in_reg = seq_item.serial_in;
            opcode_reg = seq_item.opcode;
            A_reg = seq_item.A;
            B_reg = seq_item.B;
        end
    endtask

    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("REPORT_PHASE", $sformatf("Total successful transactions:
%0d ", correct_count), UVM_MEDIUM);
        `uvm_info("REPORT_PHASE", $sformatf("Total failed transactions: %0d ",
error_count), UVM_MEDIUM);
    endfunction

endclass: alsu_scoreboard

endpackage: alsu_scoreboard_pkg
```

Configuration package:

```
package alsu_config_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: alsu_config

endpackage: alsu_config_pkg
```

Sequence item package:

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

class alsu_seq_item extends uvm_sequence_item;
    `uvm_object_utils(alsu_seq_item)

    rand bit rst;
    rand bit cin;
    rand bit red_op_A;
    rand bit red_op_B;
    rand bit bypass_A;
    rand bit bypass_B;
    rand direction_e direction;
    rand bit serial_in;
    rand opcode_e opcode;
    rand bit signed [2:0] A;
    rand bit signed [2:0] B;
    logic [15:0] leds;
    logic signed [5:0] out;

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

    function string convert2string();
        return $sformatf("%s rst = %0b, A = 0b%0b, B = 0b%0b, opcode = %0s,
cin = %0b, red_op_A = %0b, red_op_B = %0b, bypass_A = %0b, bypass_B = %0b,
direction = %s, serial_in = %0b, out = 0b%0b, leds = 0x%0h ",
            super.convert2string(), rst, A, B, opcode, cin, red_op_A,
red_op_B, bypass_A, bypass_B, direction, serial_in, out, leds);
    endfunction

    function string convert2string_stimulus();
        return $sformatf("rst = %0b, A = 0b%0b, B = 0b%0b, opcode = %0s, cin =
%0b, red_op_A = %0b, red_op_B = %0b, bypass_A = %0b, bypass_B = %0b, direction
= %s, serial_in = %0b",
                    rst, A, B, opcode, cin, red_op_A, red_op_B, bypass_A,
bypass_B, direction, serial_in);
    endfunction

    constraint c_rst {
        rst dist {0 := 95, 1 := 5};
    }

    constraint c_ADD_MULT {
```

```
        if (opcode == ADD || opcode == MULT) {
            A dist {MAXNEG := 30, ZERO := 30, MAXPOS := 30, [-3:-1] := 5,
[1:2] := 5};
            B dist {MAXNEG := 30, ZERO := 30, MAXPOS := 30, [-3:-1] := 5,
[1:2] := 5};
        }
    }

    constraint c_OR_XOR_A {
        if ((opcode inside {OR, XOR}) && red_op_A && !red_op_B) {
            A dist {3'b001 := 30, 3'b010 := 30, 3'b100 := 30, [3'b000:3'b111]
:= 10};
            B == 3'b000;
        }
    }

    constraint c_OR_XOR_B {
        if ((opcode inside {OR, XOR}) && red_op_B && !red_op_A) {
            B dist {3'b001 := 30, 3'b010 := 30, 3'b100 := 30, [3'b000:3'b111]
:= 10};
            A == 3'b000;
        }
    }

    constraint c_opcode {
        opcode dist {[OR:ROTATE] := 80, [INVALID_6:INVALID_7] := 20};
    }

    constraint c_bypass {
        bypass_A dist {0 := 95, 1 := 5};
        bypass_B dist {0 := 95, 1 := 5};
    }

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

endpackage: alsu_seq_item_pkg
```

Reset sequence package:

```
package alsu_reset_seq_pkg;
import uvm_pkg::*;
import alsu_seq_item_pkg::*;
import alsu_shared_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;
        seq_item.A = 0;
        seq_item.B = 0;
        seq_item.opcode = opcode_e'(0);
        seq_item.cin = 0;
        seq_item.red_op_A = 0;
        seq_item.red_op_B = 0;
        seq_item.bypass_A = 0;
        seq_item.bypass_B = 0;
        seq_item.direction = direction_e'(0);
        seq_item.serial_in = 0;
        finish_item(seq_item);
    endtask
endclass: alsu_reset_seq

endpackage: alsu_reset_seq_pkg
```

Main sequence package:

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

class alsu_main_seq extends uvm_sequence #(alsu_seq_item);
    `uvm_object_utils(alsu_main_seq)

    alsu_seq_item seq_item;

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

    task body();
        repeat (100) begin
            seq_item = alsu_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
endclass: alsu_main_seq
endpackage: alsu_main_seq_pkg
```

Assertions file:

```
import alsu_shared_pkg::*;

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

    logic invalid_red_op, invalid_opcode, invalid;

    assign invalid_red_op = (red_op_A | red_op_B) & (opcode[1] | opcode[2]);
    assign invalid_opcode = opcode[1] & opcode[2];
    assign invalid        = invalid_red_op | invalid_opcode;

    always_comb begin
        if (rst) begin
            assert_reset_leds: assert final(leds == 16'b0);
        end
    end

    property invalid_leds_p;
        @(posedge clk) disable iff (rst) (invalid) |-> ##2 (leds ==
~$past(leds));
    endproperty

    property valid_leds_p;
        @(posedge clk) disable iff (rst) (!invalid) |-> ##2 (leds == 16'b0);
    endproperty

    always_comb begin
        if (rst) begin
            assert_reset_out: assert final(out == 6'b0);
        end
```

```
    end
    property priority_bypass_A_p;
        @(posedge clk) disable iff (rst)
            (bypass_A && bypass_B && INPUT_PRIORITY == "A") |-> ##2 (out ==
$past(A, 2));
    endproperty

    property priority_bypass_B_p;
        @(posedge clk) disable iff (rst)
            (bypass_A && bypass_B && INPUT_PRIORITY == "B") |-> ##2 (out ==
$past(B, 2));
    endproperty

    property bypass_A_p;
        @(posedge clk) disable iff (rst)
            (bypass_A && !bypass_B) |-> ##2 (out == $past(A, 2));
    endproperty

    property bypass_B_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && bypass_B) |-> ##2 (out == $past(B, 2));
    endproperty

    property invalid_out_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && invalid) |-> ##2 (out == 6'b0);
    endproperty

    property red_or_priority_A_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == OR && red_op_A &&
red_op_B && INPUT_PRIORITY == "A") |-> ##2
            (out == |$past(A, 2));
    endproperty

    property red_or_priority_B_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == OR && red_op_A &&
red_op_B && INPUT_PRIORITY == "B") |-> ##2
            (out == |$past(B, 2));
    endproperty

    property red_or_A_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == OR && red_op_A &&
!red_op_B) |-> ##2
            (out == |$past(A, 2));
    endproperty
```

```
    property red_or_B_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == OR && !red_op_A
&& red_op_B) |-> ##2
            (out == |$past(B, 2));
    endproperty

    property or_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == OR && !red_op_A
&& !red_op_B) |-> ##2
            (out == $past(A, 2) | $past(B, 2));
    endproperty

    property red_xor_priority_A_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == XOR && red_op_A
&& red_op_B && INPUT_PRIORITY == "A") |-> ##2
            (out == ^$past(A, 2));
    endproperty

    property red_xor_priority_B_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == XOR && red_op_A
&& red_op_B && INPUT_PRIORITY == "B") |-> ##2
            (out == ^$past(B, 2));
    endproperty

    property red_xor_A_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == XOR && red_op_A
&& !red_op_B) |-> ##2
            (out == ^$past(A, 2));
    endproperty

    property red_xor_B_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == XOR && !red_op_A
&& red_op_B) |-> ##2
            (out == ^$past(B, 2));
    endproperty

    property xor_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == XOR && !red_op_A
&& !red_op_B) |-> ##2
            (out == $past(A, 2) ^ $past(B, 2));
    endproperty
```

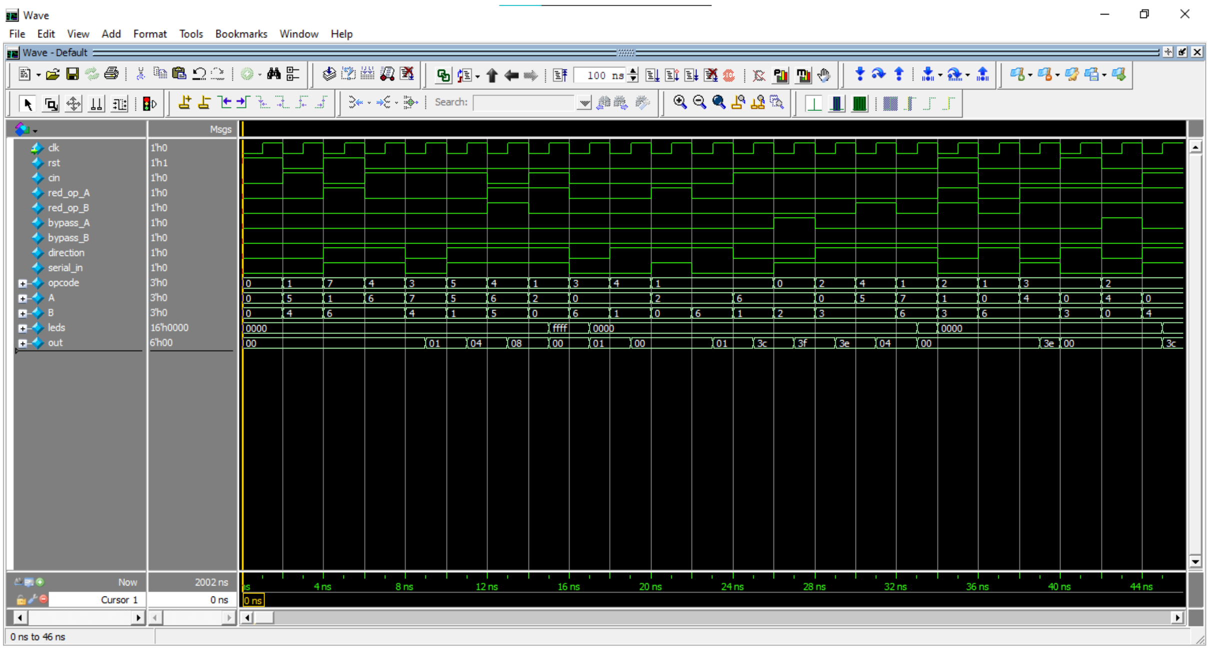
```
    property full_add_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == ADD && FULL_ADDER
== "ON") |-> ##2
            (out == $past(A, 2) + $past(B, 2) + $past(cin, 2));
    endproperty

    property half_add_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == ADD && FULL_ADDER
== "OFF") |-> ##2
            (out == $past(A, 2) + $past(B, 2));
    endproperty

    property mult_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == MULT) |-> ##2
            (out == $past(A, 2) * $past(B, 2));
    endproperty

    property right_shift_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == SHIFT &&
direction == RIGHT) |-> ##2
            (out == {$past(serial_in,2), $past(out[5:1])});
    endproperty

    property left_shift_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == SHIFT &&
direction == LEFT) |-> ##2
            (out == {$past(out[4:0]), $past(serial_in, 2)});
    endproperty

    property right_rotate_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == ROTATE &&
direction == RIGHT) |-> ##2
                (out =={$past(out[0]), $past(out[5:1])});
    endproperty

    property left_rotate_opcode_p;
        @(posedge clk) disable iff (rst)
            (!bypass_A && !bypass_B && !invalid && opcode == ROTATE &&
direction == LEFT) |-> ##2
            (out =={$past(out[4:0]), $past(out[5])});
    endproperty
```

```
    assert_invalid_leds_p: assert property(invalid_leds_p);
    assert_valid_leds_p: assert property(valid_leds_p);
    assert_priority_bypass_A_p: assert property(priority_bypass_A_p);
    assert_priority_bypass_B_p: assert property(priority_bypass_B_p);
    assert_invalid_out_p: assert property(invalid_out_p);
    assert_red_or_priority_A_p: assert property(red_or_priority_A_p);
    assert_red_or_priority_B_p: assert property(red_or_priority_B_p);
    assert_red_or_A_p: assert property(red_or_A_p);
    assert_red_or_B_p: assert property(red_or_B_p);
    assert_or_opcode_p: assert property(or_opcode_p);
    assert_red_xor_priority_A_p: assert property(red_xor_priority_A_p);
    assert_red_xor_priority_B_p: assert property(red_xor_priority_B_p);
    assert_red_xor_A_p: assert property(red_xor_A_p);
    assert_red_xor_B_p: assert property(red_xor_B_p);
    assert_xor_opcode_p: assert property(xor_opcode_p);
    assert_full_add_opcode_p: assert property(full_add_opcode_p);
    assert_half_add_opcode_p: assert property(half_add_opcode_p);
    assert_mult_opcode_p: assert property(mult_opcode_p);
    assert_right_shift_opcode_p: assert property(right_shift_opcode_p);
    assert_left_shift_opcode_p: assert property(left_shift_opcode_p);
    assert_right_rotate_opcode_p: assert property(right_rotate_opcode_p);
    assert_left_rotate_opcode_p: assert property(left_rotate_opcode_p);

    cover_invalid_leds_p: cover property(invalid_leds_p);
    cover_valid_leds_p: cover property(valid_leds_p);
    cover_priority_bypass_A_p: cover property(priority_bypass_A_p);
    cover_priority_bypass_B_p: cover property(priority_bypass_B_p);
    cover_invalid_out_p: cover property(invalid_out_p);
    cover_red_or_priority_A_p: cover property(red_or_priority_A_p);
    cover_red_or_priority_B_p: cover property(red_or_priority_B_p);
    cover_red_or_A_p: cover property(red_or_A_p);
    cover_red_or_B_p: cover property(red_or_B_p);
    cover_or_opcode_p: cover property(or_opcode_p);
    cover_red_xor_priority_A_p: cover property(red_xor_priority_A_p);
    cover_red_xor_priority_B_p: cover property(red_xor_priority_B_p);
    cover_red_xor_A_p: cover property(red_xor_A_p);
    cover_red_xor_B_p: cover property(red_xor_B_p);
    cover_xor_opcode_p: cover property(xor_opcode_p);
    cover_full_add_opcode_p: cover property(full_add_opcode_p);
    cover_half_add_opcode_p: cover property(half_add_opcode_p);
    cover_mult_opcode_p: cover property(mult_opcode_p);
    cover_right_shift_opcode_p: cover property(right_shift_opcode_p);
    cover_left_shift_opcode_p: cover property(left_shift_opcode_p);
    cover_right_rotate_opcode_p: cover property(right_rotate_opcode_p);
    cover_left_rotate_opcode_p: cover property(left_rotate_opcode_p);

endmodule
```

Do file:

```
vlib work

vlog -f src_files.list +cover -covercells

vsim -voptargs=+acc work.alsu_tb -cover -classdebug -uvmcontrol=all

add wave /alsu_tb/alsuif/*

coverage save ALSU_tb.ucdb -onexit

run -all

quit -sim
```

Simulation snippet:



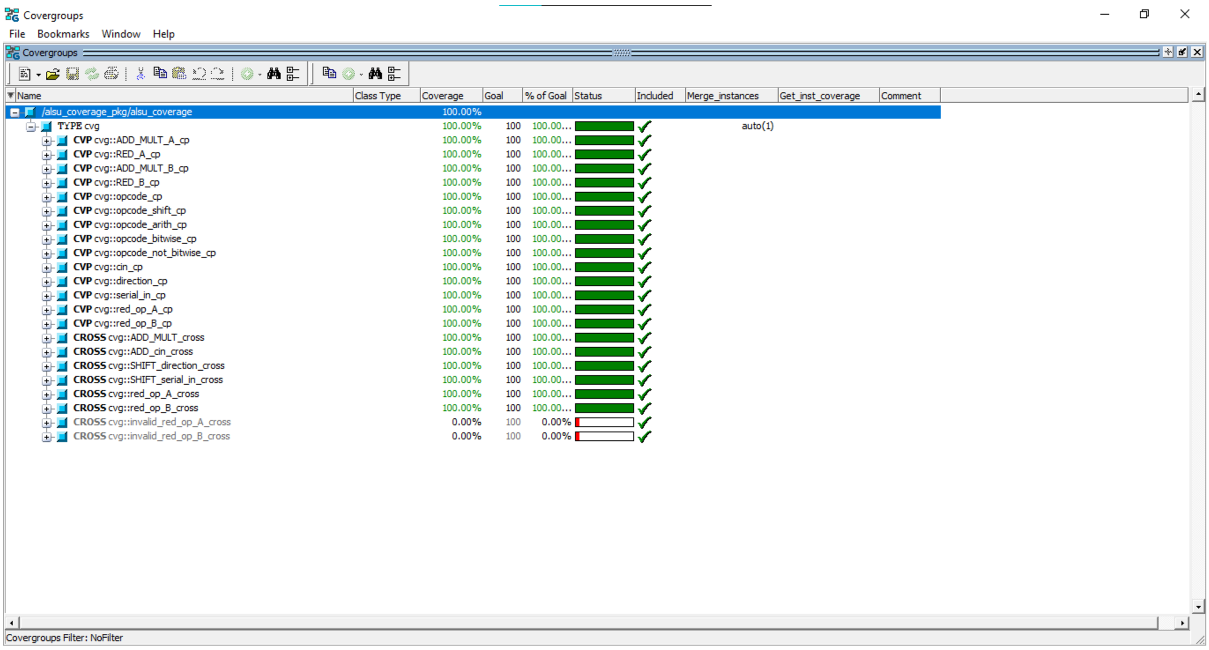
Transcript:

```
# UVM_INFO ALSU_test.sv(42) @ 2: uvm_test_top [RUN_PHASE] Reset Asserted

# UVM_INFO ALSU_test.sv(43) @ 2: uvm_test_top [RUN_PHASE] Reset Deasserted

# UVM_INFO ALSU_test.sv(46) @ 2: uvm_test_top [RUN_PHASE] Stimulus Generation
Started

# UVM_INFO ALSU_test.sv(48) @ 2002: uvm_test_top [RUN_PHASE] Stimulus Generation
Ended

# UVM_INFO ALSU_scoreboard.sv(173) @ 2002: uvm_test_top.env.scoreboard
[REPORT_PHASE] Total successful transactions: 1001

# UVM_INFO ALSU_scoreboard.sv(174) @ 2002: uvm_test_top.env.scoreboard
[REPORT_PHASE] Total failed transactions: 0

#

# --- UVM Report Summary ---

#

# ** Report counts by severity

# UVM_INFO :   10

# UVM_WARNING :    0

# UVM_ERROR :    0

# UVM_FATAL :    0

# ** Report counts by id

# [Questa UVM]     2

# [REPORT_PHASE]     2

# [RNTST]     1

# [RUN_PHASE]     4

# [TEST_DONE]     1

# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-
1.1d/src/base/uvm_root.svh(430)

#    Time: 2002 ns  Iteration: 61  Instance: /alsu_tb
```

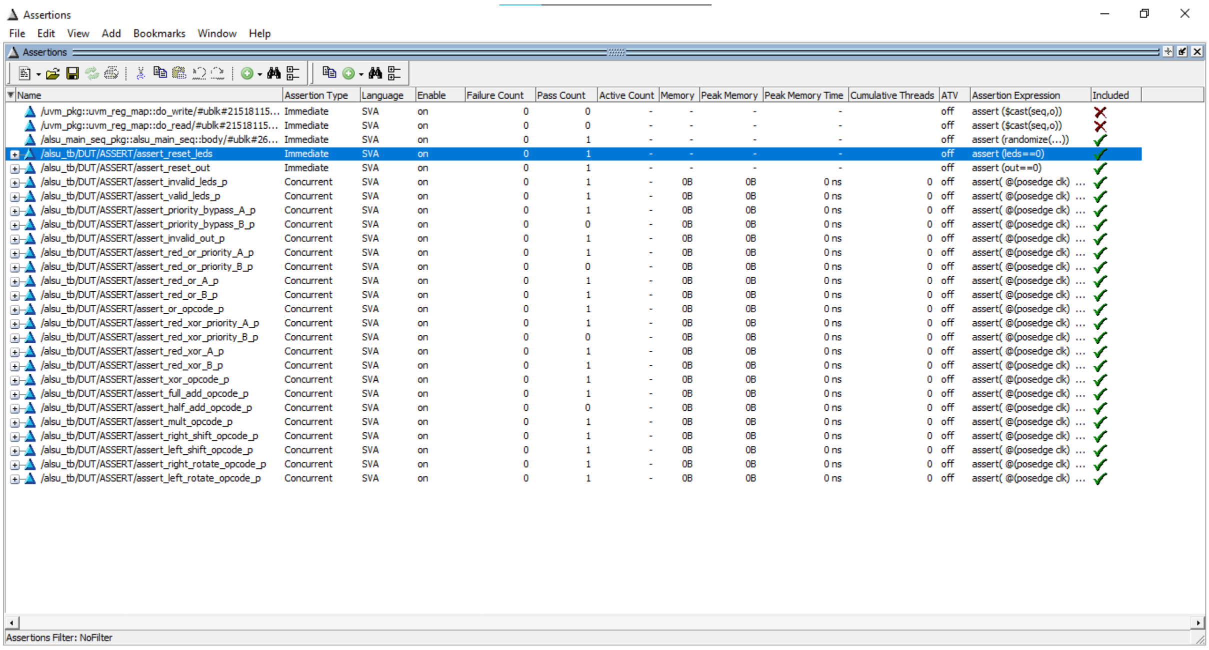
Code coverage:

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

=============================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        32         0   100.00%

=============================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      48        48         0   100.00%

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

Functional coverage:



Assertions coverage:



Assertions

File Edit View Add Bookmarks Window Help

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_reg_map::do_write/#ublk#21518115...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#21518115...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/alsu_main_seq_pkg::alsu_main_seq::body/#ublk#26...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/alsu_tb/DUT/ASSERT/assert_reset_leds	Immediate	SVA	on	0	1	-	-	-	-		off	assert (leds==0)	✓
/alsu_tb/DUT/ASSERT/assert_reset_out	Immediate	SVA	on	0	1	-	-	-	-		off	assert (out==0)	✓
/alsu_tb/DUT/ASSERT/assert_invalid_leds_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_valid_leds_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_priority_bypass_A_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_priority_bypass_B_p	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_invalid_out_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_or_priority_A_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_or_priority_B_p	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_or_A_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_or_B_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_or_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_xor_priority_A_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_xor_priority_B_p	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_xor_A_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_red_xor_B_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_xor_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_full_add_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_half_add_opcode_p	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_mult_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_right_shift_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_left_shift_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_right_rotate_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓
/alsu_tb/DUT/ASSERT/assert_left_rotate_opcode_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) ...	✓

Assertions Filter: NoFilter