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;

     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 |
| 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

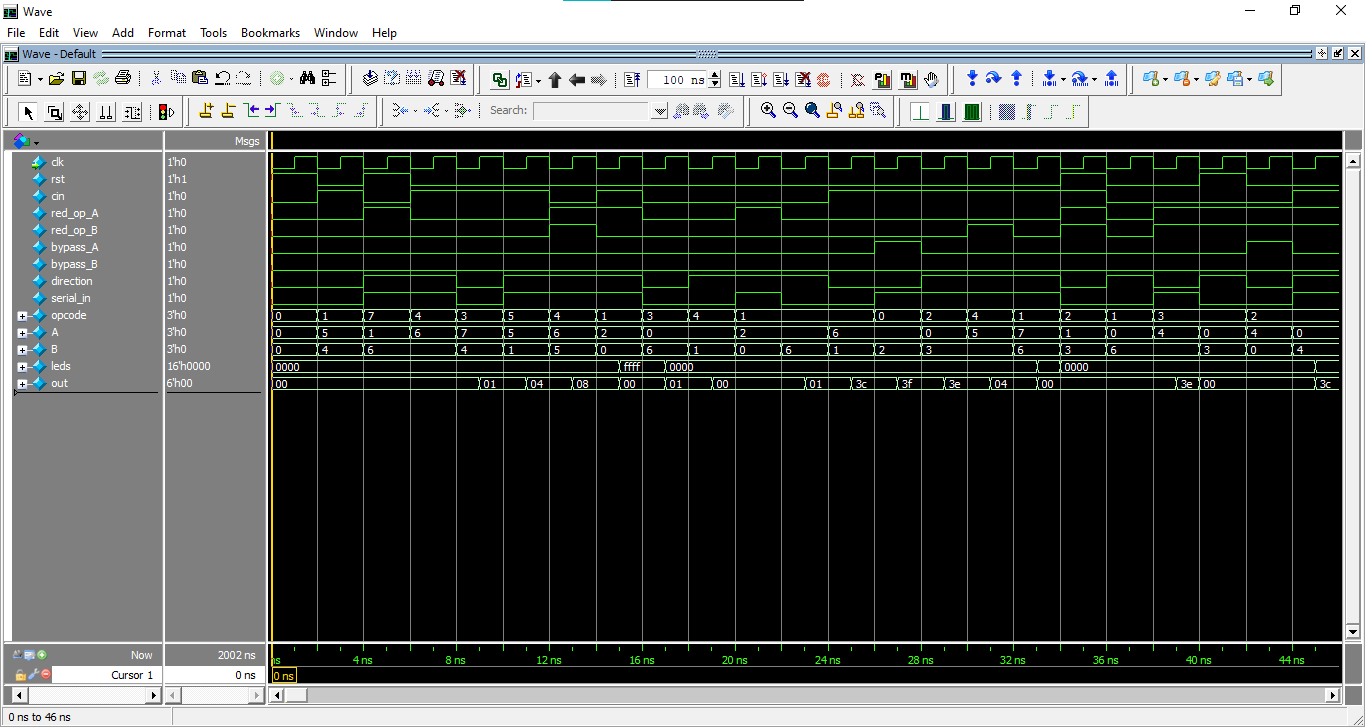
add wave /alsu\_tb/alsuif/\*

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%

==============================================================================

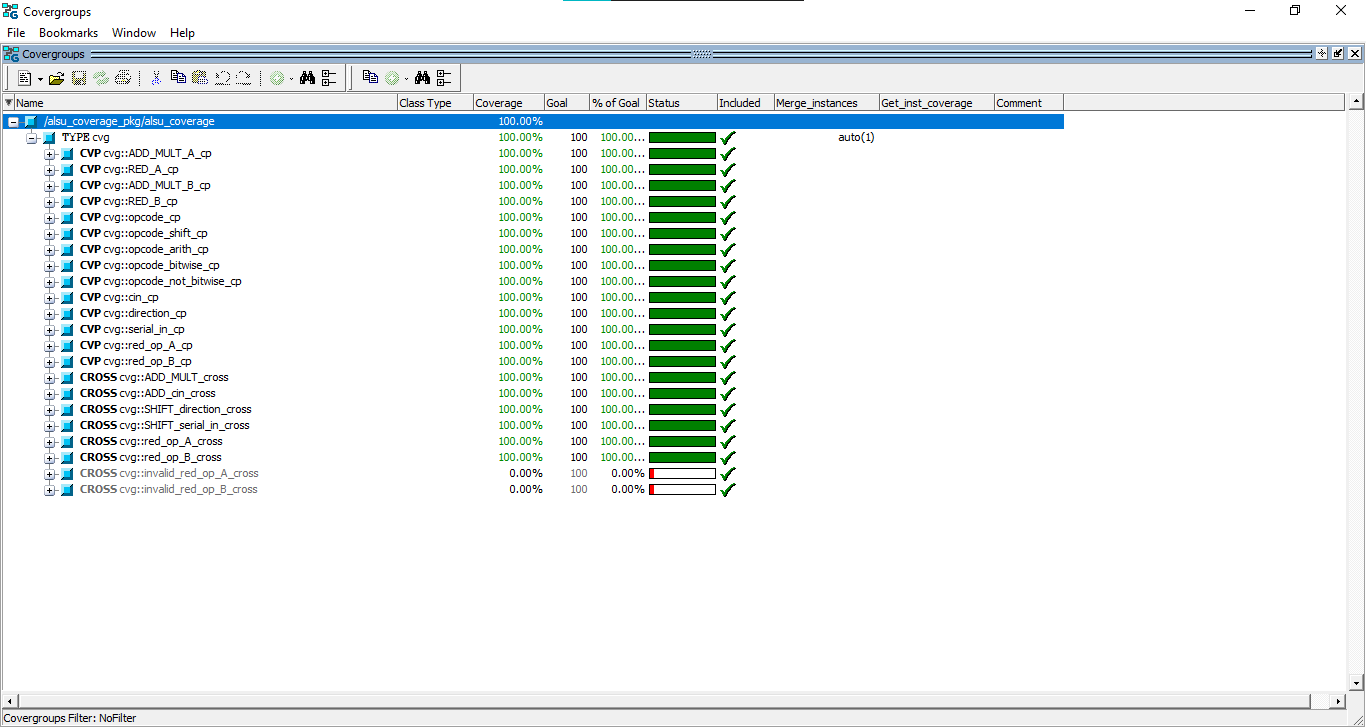
Toggle Coverage:

Enabled Coverage Bins Hits Misses Coverage

---------------- ---- ---- ------ --------

Toggles 118 118 0 100.00%

Functional coverage:



Assertions coverage:

