Digital Verification using SV and UVM
Assignment-6

Name: Fares Khalaf Sultan

❖ Top module:

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

module top_tb();

    bit clk;

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

    ALSU_if alsu_if(clk);

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

    bind ALSU ALSU_SVA alsuSVA(alsu_if.monitor);

    initial begin
        uvm_config_db#(virtual ALSU_if)::set(null,"uvm_test_top","ALSU_IF",alsu_if);
        run_test("alsu_test");
    end
endmodule
```

❖ Interface:

```
interface ALSU_if(input bit clk);

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

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

endinterface
```

❖ Test:

```
package alsu_test_pkg;
    import uvm_pkg::*;
    import alsu_env_pkg::*;
    import config_obj_pkg::*;
    import alsu_sequence_pkg::*;
    `include "uvm_macros.svh"

    class alsu_test extends uvm_test;
        `uvm_component_utils(alsu_test)

        alsu_env env0;
        alsu_config alsu_config_obj_test;
        alsu_reset_sequence reset_seq;
        alsu_sequence_1 seq_1;
        alsu_sequence_2 seq_2;
        alsu_sequence_3 seq_3;
        alsu_sequence_4 seq_4;
        alsu_sequence_5 seq_5;


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

```
28          function void build_phase(uvm_phase phase);
29              super.build_phase(phase);
30              alsu_config_obj_test = alsu_config::type_id::create("alsu_config_obj_test");
31
32              if(!uvm_config_db #(virtual ALSU_if)::get(this,"","ALSU_IF",alsu_config_obj_test.alsu_config_if))
33                  `uvm_fatal("build_phase","Test - unable to get the virtual interface")
34              uvm_config_db #(alsu_config)::set(this,"*","CFG", alsu_config_obj_test);
35
36              env0 = alsu_env::type_id::create("env0",this);
37              reset_seq = alsu_reset_sequence::type_id::create("reset_seq");
38              seq_1 = alsu_sequence_1::type_id::create("seq1");
39              seq_2 = alsu_sequence_2::type_id::create("seq2");
40              seq_3 = alsu_sequence_3::type_id::create("seq3");
41              seq_4 = alsu_sequence_4::type_id::create("seq4");
42              seq_5 = alsu_sequence_5::type_id::create("seq5");
43
44          endfunction
```

```
6           task run_phase(uvm_phase phase);
7               super.run_phase(phase);
8
9               phase.raise_objection(this);
0               //=========Reset_sequence===========
1                   `uvm_info("run_phase","Reset Asserted",UVM_MEDIUM)
2                   reset_seq.start(env0.agt.sqr);
3               //=========Sequence_1===========
4                   `uvm_info("run_phase","Sequence-1 started",UVM_MEDIUM)
5                   seq_1.start(env0.agt.sqr);
6               //=========Sequence_2===========
7                   `uvm_info("run_phase","Sequence-2 started",UVM_MEDIUM)
8                   seq_2.start(env0.agt.sqr);
9               //=========Sequence_3===========
0                   `uvm_info("run_phase","Sequence-3 started",UVM_MEDIUM)
1                   seq_3.start(env0.agt.sqr);
2               //=========Sequence_4===========
3                   `uvm_info("run_phase","Sequence-4 started",UVM_MEDIUM)
4                   seq_4.start(env0.agt.sqr);
5               //=========Sequence_5===========
6                   `uvm_info("run_phase","Sequence-5 started",UVM_MEDIUM)
7                   seq_5.start(env0.agt.sqr);
8               phase.drop_objection(this);
9           endtask
0       endclass
```

❖ Sequences:

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


//==============================Reset Sequence==============================
    class alsu_reset_sequence extends uvm_sequence #(alsu_seq_item);
        `uvm_object_utils(alsu_reset_sequence)
        alsu_seq_item seq_item;

        function new(string name = "alsu_reset_sequence");
            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 = 1;
            seq_item.B = 1;
            seq_item.opcode = ADD;

            finish_item(seq_item);
        endtask
    endclass
```

```
//=================opcode sequence constraint is disabled, all others are Enabled================

    class alsu_sequence_1 extends uvm_sequence #(alsu_seq_item);
        `uvm_object_utils(alsu_sequence_1)
        alsu_seq_item seq_item;

        function new(string name = "alsu_sequence_1");
            super.new(name);
        endfunction

        task body;
            seq_item = alsu_seq_item::type_id::create("seq_item");

            seq_item.OpcodeSequence.constraint_mode(0);
            repeat(2000) begin
                start_item(seq_item);
                    assert (seq_item.randomize());
                finish_item(seq_item);
            end


        endtask
    endclass
```

```
//=================opcode sequence constraint is Enabled, all others are Disabled===============
    class alsu_sequence_2 extends uvm_sequence #(alsu_seq_item);
        `uvm_object_utils(alsu_sequence_2)
        alsu_seq_item seq_item;

        function new(string name = "alsu_sequence_2");
            super.new(name);
        endfunction

        task body;
            seq_item = alsu_seq_item::type_id::create("seq_item");

            seq_item.OpcodeSequence.constraint_mode(1);
            seq_item.ALSUsignals.constraint_mode(0);
            seq_item.rst = 0;
            seq_item.bypass_A = 0;
            seq_item.bypass_B = 0;
            seq_item.red_op_A = 0;
            seq_item.red_op_B = 0;
            repeat(100) begin
                start_item(seq_item);
                    assert (seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
```

```
//=================sequence to hit BinsBitwiseCrossWalkingB bins================

    class alsu_sequence_4 extends uvm_sequence #(alsu_seq_item);
        `uvm_object_utils(alsu_sequence_4)
        alsu_seq_item seq_item;

        function new(string name = "alsu_sequence_4");
            super.new(name);
        endfunction

        task body;
            seq_item = alsu_seq_item::type_id::create("seq_item");

            seq_item.OpcodeSequence.constraint_mode(0);
            seq_item.ALSUsignals.constraint_mode(1);
            repeat(50) begin
                start_item(seq_item);
                    assert (seq_item.randomize() with{
                        A_exp inside{WalkingOnesSequence};
                        A == 0;
                        B == B_exp;
                        rst == 0;
                        opcode == opcode_exp;
                        red_op_B == 1;
                        red_op_A == 0;
                        opcode_exp dist {
                            OR := 50,
                            XOR := 50
                        };
                    });
                finish_item(seq_item);
            end
        endtask
    endclass

            end
//====================sequence to hit Opcode transition bins====================

    class alsu_sequence_5 extends uvm_sequence #(alsu_seq_item);
        `uvm_object_utils(alsu_sequence_5)
        alsu_seq_item seq_item;
        opcode_e transitionPattern[6] = '{OR, XOR, ADD, MULT, SHIFT, ROTATE};

        function new(string name = "alsu_sequence_5");
            super.new(name);
        endfunction

        task body;
            seq_item = alsu_seq_item::type_id::create("seq_item");

            seq_item.OpcodeSequence.constraint_mode(0);
            seq_item.ALSUsignals.constraint_mode(0);

            foreach (transitionPattern[i]) begin
                start_item(seq_item);
                    seq_item.rst = 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.opcode = transitionPattern[i];
                finish_item(seq_item);
            end
        endtask
    endclass

endpackage
```

❖ Sequence item:

```
package alsu_seq_item_pkg;

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

     typedef enum bit [2:0]  {
        OR = 3'b000,
        XOR = 3'b001,
        ADD = 3'b010,
        MULT = 3'b011,
        SHIFT = 3'b100,
        ROTATE = 3'b101,
        INVALID_6 = 3'b110,
        INVALID_7 = 3'b111
     } opcode_e;

    class alsu_seq_item extends uvm_sequence_item;
        `uvm_object_utils(alsu_seq_item)

        rand bit cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
        bit clk;
        rand opcode_e opcode;
        randc opcode_e opcode_exp;
        rand opcode_e opcode_sequence [6];
        rand bit signed [2:0] A, B;
        randc bit[2:0] A_exp,B_exp;
        bit [2:0] WalkingOnesSequence[3] = '{3'b001,3'b010,3'b100};
        parameter MAXPOS = 3 ;
        parameter MAXNEG = -4;

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

```
36          constraint ALSUsignals{
37
38                  soft rst dist{
39                      0 := 95, 1 := 5
40                  };
41
42                  if(opcode == (ADD||MULT)){
43                      soft A dist{
44                      0 := 30,3 := 30,-4:= 30
45                      };
46
47                      soft B dist{
48                      0 := 30,3 := 30,-4:= 30
49                      };
50                  }
51                  else if (opcode == (OR || XOR)){
52                      A > 0;
53                      B > 0;
54                  }
55
56              soft opcode dist {
57                  [OR:ROTATE] :/ 95,
58                  [INVALID_6:INVALID_7] :/ 5
59              };
60
61              bypass_A dist{
62                  0 := 90, 1 := 10
63              };
64              bypass_B dist{
65                  0 := 90,1 := 10
66              };
```

```
            if( opcode == (OR||XOR)){
                soft red_op_A dist{
                    0 := 50,1 := 50
                };
                soft red_op_B dist{
                    0 := 50,1 := 50
                };
            }
            else {
                soft red_op_A dist{
                    0 := 98,1 := 2
                };
                soft red_op_B dist{
                    0 := 98,1 := 2
                };
            }
        }
        constraint OpcodeSequence {
            unique{opcode_sequence};
            foreach (opcode_sequence[i]) opcode_sequence[i] inside {[OR:ROTATE]};
        }
    endclass
endpackage
```

❖ Environment:

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

    class alsu_env extends uvm_env;
        `uvm_component_utils(alsu_env)

        alsu_agent agt;
        alsu_scoreboard sb;
        alsu_coverage_collector cov;

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_a_port.connect(sb.sb_a_export);
            agt.agt_a_port.connect(cov.cov_a_export);
        endfunction
    endclass
```

❖ Scoreboard:

```
class alsu_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(alsu_scoreboard)

    uvm_analysis_export #(alsu_seq_item) sb_a_export;
    uvm_tlm_analysis_fifo #(alsu_seq_item) sb_fifo;
    alsu_seq_item sb_seq_item;

    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_a_export = new("sb_a_export", this);
        sb_fifo = new("sb_fifo", this);
    endfunction

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

        sb_a_export.connect(sb_fifo.analysis_export);
    endfunction

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

        forever begin
            sb_fifo.get(sb_seq_item);
        end
    endtask
endclass
```

❖ Coverage collector:

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

    class alsu_coverage_collector extends uvm_component;
        `uvm_component_utils(alsu_coverage_collector)

        parameter MAXPOS = 3 ;
        parameter MAXNEG = -4;

        uvm_analysis_export #(alsu_seq_item) cov_a_export;
        uvm_tlm_analysis_fifo #(alsu_seq_item) cov_fifo;
        alsu_seq_item cov_seq_item;

        covergroup CovCode ;

        //=============== A coverage points ====================
           A_cp: coverpoint cov_seq_item.A iff(!cov_seq_item.rst){
            bins A_data_0 = {0};
            bins A_data_max = {cov_seq_item.MAXPOS};
            bins A_data_min = {cov_seq_item.MAXNEG};
            bins A_data_default = default;
           }

           A_WalkingOnes_cp: coverpoint unsigned'(cov_seq_item.A) iff(!cov_seq_item.rst && cov_seq_item.red_op_A){
            bins A_data_walkingones001 = {3'b001};
            bins A_data_walkingones010 = {3'b010};
            bins A_data_walkingones100 = {3'b100};
           }
```

```
//=============== B coverage points ====================

    B_cp: coverpoint cov_seq_item.B iff(!(cov_seq_item.rst || cov_seq_item.bypass_A)){ //*
     bins B_data_0 = {0};
     bins B_data_max = {cov_seq_item.MAXPOS};
     bins B_data_min = {cov_seq_item.MAXNEG};
     bins B_data_default = default;
    }

    B_WalkingOnes_cp: coverpoint unsigned'(cov_seq_item.B)
     iff(!(cov_seq_item.rst || cov_seq_item.bypass_A) && cov_seq_item.red_op_B && !cov_seq_item.red_op_A){
     bins B_data_walkingones001 = {3'b001};
     bins B_data_walkingones010 = {3'b010};
     bins B_data_walkingones100 = {3'b100};
    }
```

```
//=============== Opcode coverage points ====================

    ALU_cp: coverpoint cov_seq_item.opcode
    iff(!(cov_seq_item.rst || cov_seq_item.bypass_A || cov_seq_item.bypass_B)){

        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);
    }
```

```
//=========== reduction operation coverage point ===============

    red_op_A_cp: coverpoint cov_seq_item.red_op_A{
        bins red_op_A_bins = {1};
    }

    red_op_B_cp: coverpoint cov_seq_item.red_op_B{
        bins red_op_B_bins = {1};
    }
```

```
//======================Cross coverpoints=====================

    //1. add/mult cross A_cp and B_cp

    addMultCrossAB: cross ALU_cp, A_cp,B_cp{
        //option.cross_auto_bin_max = 0;
        ignore_bins notArth = !binsof(ALU_cp.Bins_arith);
    }

    //2. add cross cin

    addCrossCin: cross ALU_cp, cov_seq_item.cin iff(cov_seq_item.opcode == ADD){
       //option.cross_auto_bin_max = 0;
       ignore_bins notADD = !binsof(ALU_cp.Bins_arith) intersect{ADD};
       ignore_bins notADD2 = !binsof(ALU_cp.Bins_arith);
    }

    //3. shift/rotate cross direction

    shiftRotateCrossDir: cross ALU_cp, cov_seq_item.direction {

        ignore_bins notShiftRotate = ! binsof(ALU_cp.Bins_shift);
    }

    //4. shift cross shift_in

    shiftCrossShiftin: cross ALU_cp, cov_seq_item.serial_in iff(cov_seq_item.opcode == SHIFT){

        ignore_bins notShiftBins = !binsof(ALU_cp.Bins_shift) intersect{SHIFT};
        ignore_bins notShift = !binsof(ALU_cp.Bins_shift);
    }
```

```
            //5. OR/XOR and redA cross (A=> walking ones, B=> 0)

            BinsBitwiseCrossWalkingA: cross ALU_cp, B_cp, A_WalkingOnes_cp iff(cov_seq_item.red_op_A){

                ignore_bins notBitwise = !binsof(ALU_cp.Bins_bitwise);
                ignore_bins BnotZero = !binsof(B_cp.B_data_0);

            }

            //6. OR/XOR and (!redA && redopB) cross (B=> walking ones, A=> 0)

            BinsBitwiseCrossWalkingB: cross ALU_cp, A_cp, B_WalkingOnes_cp
            iff((!cov_seq_item.red_op_A && cov_seq_item.red_op_B)){

                ignore_bins notBitwise = !binsof(ALU_cp.Bins_bitwise);
                ignore_bins BnotZero = !binsof(A_cp.A_data_0);

            }

            //7. reduction operation while opcode != OR/XOR

            Bins_invalid_reduction: cross ALU_cp ,  red_op_A_cp,red_op_B_cp {

                ignore_bins valid_A_reduction = binsof(ALU_cp.Bins_bitwise) && binsof(red_op_A_cp.red_op_A_bins);
                ignore_bins valid_B_reduction = binsof(ALU_cp.Bins_bitwise) && binsof(red_op_B_cp.red_op_B_bins);
                ignore_bins transition_cp = binsof(ALU_cp.Bins_trans) ;
            }

        endgroup
```

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


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

        cov_a_export = new("cov_a_export", this);
        cov_fifo = new("cov_fifo", this);
    endfunction

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

        cov_a_export.connect(cov_fifo.analysis_export);
    endfunction

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

        forever begin
            cov_fifo.get(cov_seq_item);
            CovCode.sample();
        end
    endtask

endclass
```

❖ Agent:

```
class alsu_agent extends uvm_agent;
    `uvm_component_utils(alsu_agent)
    alsu_driver driver;
    alsu_sequencer sqr;
    alsu_monitor mon;
    alsu_config alsu_config_obj;
    uvm_analysis_port #(alsu_seq_item) agt_a_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_config_obj))
            `uvm_fatal("build_phase","Agent - unable to get the virtual interface")

        sqr = alsu_sequencer::type_id::create("sqr",this);
        driver = alsu_driver::type_id::create("driver",this);
        mon = alsu_monitor::type_id::create("mon",this);
        agt_a_port = new("agt_a_port",this);

    endfunction

    function void connect_phase(uvm_phase phase);
        driver.alsu_driver_vif = alsu_config_obj.alsu_config_if;
        mon.alsu_monitor_vif = alsu_config_obj.alsu_config_if;
        driver.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_a_port.connect(agt_a_port);
    endfunction
endclass
```

❖ Sequencer:

```
package alsu_sequencer_pkg;

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

    class alsu_sequencer extends uvm_sequencer #(alsu_seq_item);
        `uvm_component_utils(alsu_sequencer)
        function new(string name = "alsu_sequencer", uvm_component parent = null);
            super.new(name,parent);
        endfunction
    endclass


endpackage
```

❖ Driver:

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

    virtual ALSU_if alsu_driver_vif;
    alsu_seq_item stim_seq_item;

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

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

        forever begin
            stim_seq_item = alsu_seq_item::type_id::create("stim_seq_item");
            seq_item_port.get_next_item(stim_seq_item);
                alsu_driver_vif.rst =  stim_seq_item.rst;
                alsu_driver_vif.cin =  stim_seq_item.cin;
                alsu_driver_vif.red_op_A = stim_seq_item.red_op_A;
                alsu_driver_vif.red_op_B = stim_seq_item.red_op_B;
                alsu_driver_vif.bypass_A = stim_seq_item.bypass_A;
                alsu_driver_vif.bypass_B = stim_seq_item.bypass_B;
                alsu_driver_vif.direction = stim_seq_item.direction;
                alsu_driver_vif.serial_in = stim_seq_item.serial_in;
                alsu_driver_vif.opcode = stim_seq_item.opcode;
                alsu_driver_vif.A = stim_seq_item.A;
                alsu_driver_vif.B = stim_seq_item.B;
                @(negedge alsu_driver_vif.clk);
            seq_item_port.item_done();
        end
    endtask
endclass
```

❖ Monitor:

```
class alsu_monitor extends uvm_monitor;
    `uvm_component_utils(alsu_monitor)
    virtual ALSU_if alsu_monitor_vif;
    alsu_seq_item rsp_seq_item;
    uvm_analysis_port #(alsu_seq_item) mon_a_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);
        mon_a_port = new("mon_a_port",this);
    endfunction


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

        forever begin
            rsp_seq_item = alsu_seq_item::type_id::create("rsp_seq_item");
            @(negedge alsu_monitor_vif.clk);

            rsp_seq_item.rst = alsu_monitor_vif.rst ;
            rsp_seq_item.cin = alsu_monitor_vif.cin ;
            rsp_seq_item.red_op_A = alsu_monitor_vif.red_op_A;
            rsp_seq_item.red_op_B = alsu_monitor_vif.red_op_B;
            rsp_seq_item.bypass_A = alsu_monitor_vif.bypass_A;
            rsp_seq_item.bypass_B = alsu_monitor_vif.bypass_B;
            rsp_seq_item.direction = alsu_monitor_vif.direction;
            rsp_seq_item.serial_in = alsu_monitor_vif.serial_in;
            rsp_seq_item.opcode = opcode_e'(alsu_monitor_vif.opcode);
            rsp_seq_item.A = alsu_monitor_vif.A;
            rsp_seq_item.B = alsu_monitor_vif.B;

            mon_a_port.write(rsp_seq_item); //broadcasts the DUT response
        end
    endtask
endclass
```

❖ Assertions module:

```
module ALSU_SVA(ALSU_if IF);

    wire invalid,invalid_opcode,invalid_red_op;

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

    always_comb begin
        if(IF.rst) begin
            Reset: assert final ((IF.out === 'b0) && (IF.leds === 'b0))
            else $error("Assertion Reset failed!");
        end
    end

//================================sequences=========================================

    sequence Shift_Left;
        (IF.opcode == 'b100) && IF.direction;
    endsequence

    sequence Shift_Right;
        (IF.opcode == 'b100) && !IF.direction;
    endsequence

    sequence Rotate_Left;
        (IF.opcode == 'b101) && IF.direction;
    endsequence

    sequence Rotate_Right;
        (IF.opcode == 'b101) && !IF.direction;
    endsequence
```

```
//================================Properties=========================================
    property Invalid_stimulus_leds_exp;
        @(posedge IF.clk) disable iff(IF.rst) (invalid) |=> ##1 IF.leds == ~$past(IF.leds,1);
    endproperty

    property Invalid_stimulus_out_exp; //****
        @(posedge IF.clk) disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
            (invalid) |=> ##1 IF.out === 3'b000;
    endproperty

    property valid_stimulus;
        @(posedge IF.clk) !(invalid) |=> ##1 IF.leds == 0;
    endproperty

    property bypass_A_feature;
        @(posedge IF.clk)
            disable iff(IF.rst)
                IF.bypass_A |=> ##1 IF.out == $past(IF.A,2);
    endproperty

    property bypass_B_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A)
                IF.bypass_B |=> ##1 IF.out == $past(IF.B,2);
    endproperty

    property redOR_A_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b000 && IF.red_op_A |=> ##1 IF.out == |$past(IF.A,2);
    endproperty

    property redXOR_A_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b001 && IF.red_op_A |=> ##1 IF.out == ^$past(IF.A,2);
    endproperty
```

```
    property redOR_B_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b000 && !IF.red_op_A && IF.red_op_B |=> ##1 IF.out == |$past(IF.B,2);
    endproperty

    property redXOR_B_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b001 && !IF.red_op_A && IF.red_op_B |=> ##1 IF.out == ^$past(IF.B,2);
    endproperty

    property normalOR_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b000 && !IF.red_op_A && !IF.red_op_B |=> ##1 IF.out == $past(IF.A,2) | $past(IF.B,2);
    endproperty

    property normalXOR_feature;
        @(posedge IF.clk)
            disable iff(IF.rst || IF.bypass_A || IF.bypass_B)
                IF.opcode =='b001 && !IF.red_op_A && !IF.red_op_B |=> ##1 IF.out == $past(IF.A,2) ^ $past(IF.B,2);
    endproperty
```

```
property ADD_feature;
    @(posedge IF.clk)
        disable iff (IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            (IF.opcode == 'b010) |=> ##1 (IF.out == ($past(IF.A,2) + $past(IF.B,2) + $past(IF.cin,2)));
endproperty

property Multiply_feature;
    @(posedge IF.clk)
        disable iff(IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            IF.opcode == 'b011 |=> ##1 IF.out == $past(IF.A,2) * $past(IF.B,2);
endproperty

property Shift_Left_feature;
    @(posedge IF.clk)
        disable iff(IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            Shift_Left |=> ##1 IF.out == {$past(IF.out[4:0],2), $past(IF.serial_in,2)};
endproperty

property Shift_Right_feature;
    @(posedge IF.clk)
        disable iff(IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            Shift_Right |=> ##1 IF.out == {$past(IF.serial_in,2),$past(IF.out[5:1],2)};
endproperty

property Rotate_Left_feature;
    @(posedge IF.clk)
        disable iff(IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            Rotate_Left |=> ##1 IF.out == {$past(IF.out[4:0],2), $past(IF.out[5],2)};
endproperty

property Rotate_Right_feature;
    @(posedge IF.clk)
        disable iff(IF.rst || IF.bypass_A || IF.bypass_B || invalid)
            Rotate_Right |=> ##1 IF.out == {$past(IF.out[0],2), $past(IF.out[5:1],2)};
endproperty
```

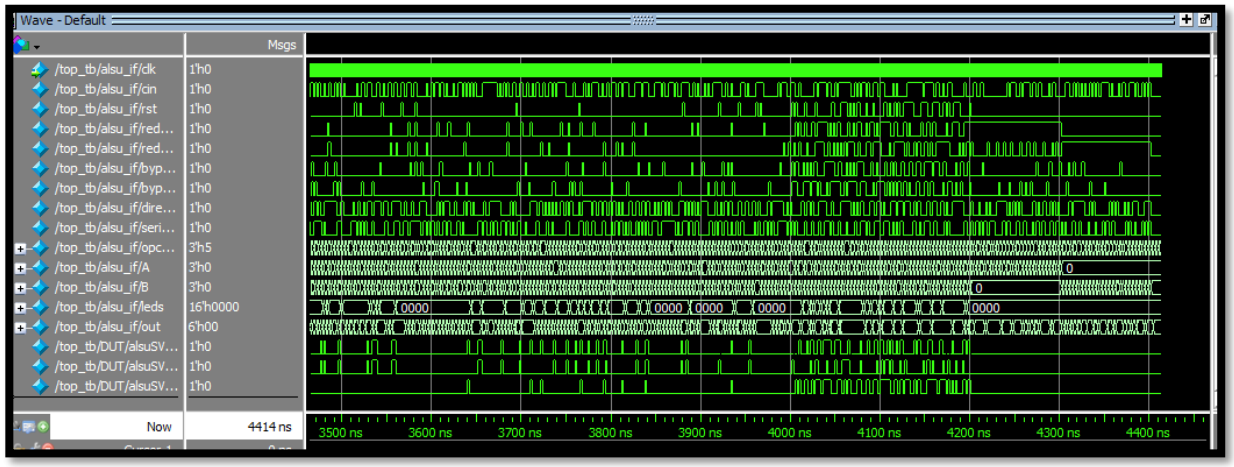
```
//==================================Assert Properties==================================

    invalid_stimulus_leds_exp: assert property (Invalid_stimulus_leds_exp)
        else $fatal("Assertion Invalid_stimulus_leds_exp failed!");
    cover property (Invalid_stimulus_leds_exp);

    invalid_stimulus_out_exp: assert property(Invalid_stimulus_out_exp)
        else $fatal("Assertion Invalid_stimulus_out_exp failed!");
    cover property(Invalid_stimulus_out_exp);

    Valid_stimulus: assert property (valid_stimulus)
        else $fatal("Assertion valid_stimulus failed!");
    cover property(valid_stimulus);

    Bypass_A_feature: assert property (bypass_A_feature)
        else $fatal("Assertion bypass_A_feature failed!");
    cover property(bypass_A_feature);

    Bypass_B_feature: assert property (bypass_B_feature)
        else $fatal("Assertion bypass_B_feature failed!");
    cover property(bypass_B_feature);
```

```
    RedOR_A_feature: assert property (redOR_A_feature)
        else $fatal("Assertion redOR_A_feature failed!");
    cover property(redOR_A_feature);

    RedXOR_A_feature: assert property (redXOR_A_feature)
        else $fatal("Assertion redXOR_A_feature failed!");
    cover property(redXOR_A_feature);

    RedOR_B_feature: assert property (redOR_B_feature)
        else $fatal("Assertion redOR_B_feature failed!");
    cover property(redOR_B_feature);

    RedXOR_B_feature: assert property (redXOR_B_feature)
        else $fatal("Assertion redXOR_B_feature failed!");
    cover property(redXOR_B_feature);

    NormalOR_feature: assert property (normalOR_feature)
        else $fatal("Assertion normalOR_feature failed!");
    cover property(normalOR_feature);

    NormalXOR_feature: assert property (normalXOR_feature)
        else $fatal("Assertion normalXOR_feature failed!");
    cover property(normalXOR_feature);
```

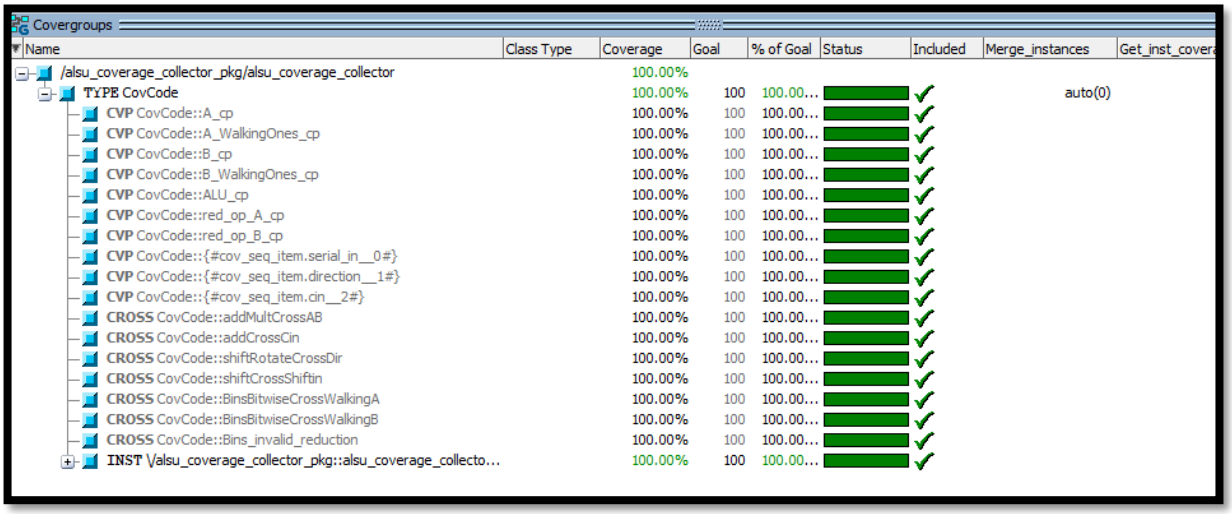
```
    add_feature: assert property (ADD_feature)
        else $fatal("Assertion ADD_feature failed!");
    cover property(ADD_feature);

    multiply_feature: assert property (Multiply_feature)
        else $fatal("Assertion Multiply_feature failed!");
    cover property(Multiply_feature);

    shift_Left_feature: assert property (Shift_Left_feature)
    $display("Shift passed ,Time:%0t",$time);
        else $fatal("Assertion Shift_Left_feature failed!");

     shift_Right_feature: assert property (Shift_Right_feature)

    shift_Right_feature: assert property (Shift_Right_feature)
        else $fatal("Assertion Shift_Right_feature failed!");
    cover property(Shift_Right_feature);

    rotate_Left_feature: assert property (Rotate_Left_feature)
        else $fatal("Assertion Rotate_Left_feature failed!");
    cover property(Rotate_Left_feature);

    rotate_Right_feature: assert property (Rotate_Right_feature)
        else $fatal("Assertion Rotate_Right_feature failed!");
    cover property(Rotate_Right_feature);

endmodule
```

❖ Results:

```
   Time: 4032 ns  Iteration: 2  Region: /uvm_pkg::uvm_task_phase::execute
UVM_INFO alsu_test.sv(60) @ 4202: uvm_test_top [run_phase] Sequence-3 started
UVM_INFO alsu_test.sv(63) @ 4302: uvm_test_top [run_phase] Sequence-4 started
UVM_INFO alsu_test.sv(66) @ 4402: uvm_test_top [run_phase] Sequence-5 started
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 4414: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase

--- 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
[RNTST]     1
[TEST_DONE]     1
[run_phase]     6
** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
   Time: 4414 ns  Iteration: 61  Instance: /top_tb
Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```



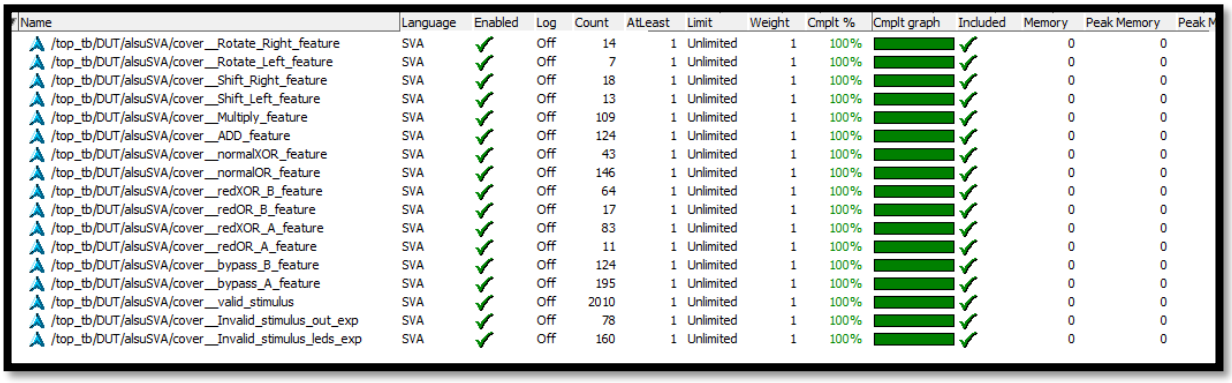
❖ Functional coverage:



Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_covera
/alsu_coverage_collector_pkg/alsu_coverage_collector		100.00%						
TYPE CovCode		100.00%	100	100.00...		✓	auto(0)	
CVP CovCode::A_cp		100.00%	100	100.00...		✓		
CVP CovCode::A_WalkingOnes_cp		100.00%	100	100.00...		✓		
CVP CovCode::B_cp		100.00%	100	100.00...		✓		
CVP CovCode::B_WalkingOnes_cp		100.00%	100	100.00...		✓		
CVP CovCode::ALU_cp		100.00%	100	100.00...		✓		
CVP CovCode::red_op_A_cp		100.00%	100	100.00...		✓		
CVP CovCode::red_op_B_cp		100.00%	100	100.00...		✓		
CVP CovCode::{#cov_seq_item.serial_in__0#}		100.00%	100	100.00...		✓		
CVP CovCode::{#cov_seq_item.direction__1#}		100.00%	100	100.00...		✓		
CVP CovCode::{#cov_seq_item.cin__2#}		100.00%	100	100.00...		✓		
CROSS CovCode::addMultCrossAB		100.00%	100	100.00...		✓		
CROSS CovCode::addCrossCin		100.00%	100	100.00...		✓		
CROSS CovCode::shiftRotateCrossDir		100.00%	100	100.00...		✓		
CROSS CovCode::shiftCrossShiftin		100.00%	100	100.00...		✓		
CROSS CovCode::BinsBitwiseCrossWalkingA		100.00%	100	100.00...		✓		
CROSS CovCode::BinsBitwiseCrossWalkingB		100.00%	100	100.00...		✓		
CROSS CovCode::Bins_invalid_reduction		100.00%	100	100.00...		✓		
INST \alsu_coverage_collector_pkg::alsu_coverage_collecto...		100.00%	100	100.00...		✓		

❖ Assertion coverage:



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak M
/top_tb/DUT/alsuSVA/cover__Rotate_Right_feature	SVA	✓	Off	14	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Rotate_Left_feature	SVA	✓	Off	7	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Shift_Right_feature	SVA	✓	Off	18	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Shift_Left_feature	SVA	✓	Off	13	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Multiply_feature	SVA	✓	Off	109	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__ADD_feature	SVA	✓	Off	124	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__normalXOR_feature	SVA	✓	Off	43	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__normalOR_feature	SVA	✓	Off	146	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__redXOR_B_feature	SVA	✓	Off	64	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__redOR_B_feature	SVA	✓	Off	17	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__redXOR_A_feature	SVA	✓	Off	83	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__redOR_A_feature	SVA	✓	Off	11	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__bypass_B_feature	SVA	✓	Off	124	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__bypass_A_feature	SVA	✓	Off	195	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__valid_stimulus	SVA	✓	Off	2010	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Invalid_stimulus_out_exp	SVA	✓	Off	78	1	Unlimited	1	100%		✓	0	0	
/top_tb/DUT/alsuSVA/cover__Invalid_stimulus_leds_exp	SVA	✓	Off	160	1	Unlimited	1	100%		✓	0	0	