Digital Verification using SV and UVM
Assignment-4

Name: Fares Khalaf Sultan

Q1)) ALSU

❖ Package class:

```
typedef enum bit [2:0]  {
        OR = 3'b00,
        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 ALSUconstraints;
    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;
    parameter MAXPOS = 3 ;
    parameter MAXNEG = -4;


    covergroup CovCode ;
```

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

        A_WalkingOnes_cp: coverpoint unsigned'(A) iff(!rst && 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 B iff(!(rst || bypass_A)){ //*
     bins B_data_0 = {0};
     bins B_data_max = {MAXPOS};
     bins B_data_min = {MAXNEG};
     bins B_data_default = default;
    }

    B_WalkingOnes_cp: coverpoint unsigned'(B) iff(!(rst || bypass_A) && red_op_B && !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 opcode iff(!(rst || bypass_A || 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 red_op_A{
    bins red_op_A_bins = {1};
}

red_op_B_cp: coverpoint 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, cin iff(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, direction {

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

    //4. shift cross shift_in

    shiftCrossShiftin: cross ALU_cp, serial_in iff(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(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((!red_op_A && 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
```

```
    constraint ALSUsignals{

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

            if(opcode == (ADD||MULT)){
                soft A dist{
                0 := 30,3 := 30,-4:= 30
                };

                soft B dist{
                0 := 30,3 := 30,-4:= 30
                };
            }
            else if (opcode == (OR || XOR)){
                A > 0;
                B > 0;
            }

        soft opcode dist {
            [OR:ROTATE] :/ 95,
            [INVALID_6:INVALID_7] :/ 5
        };

        bypass_A dist{
            0 := 90, 1 := 10
        };
        bypass_B dist{
            0 := 90,1 := 10
        };
```

```
        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]};
    }



        function new();
            CovCode = new();

        endfunction

        function void display_sequence();
            $display("opcode sequence: %0p",opcode_sequence);
        endfunction

    endclass

endpackage
```

❖ TestBench:

```
import Assignment3::*;

module ALSU_tb();

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

    bit cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
    opcode_e opcode_reg;
    bit signed [2:0] A_reg, B_reg;
    opcode_e transitionPattern[6];

    int Error_Count,Correct_Count;

    ALSUconstraints constraintVals = new;

    ALSU DUT (.*);

    //---clk generation
    initial begin
        clk = 0;
        forever begin
            constraintVals.clk = clk;
            #1 clk = !clk;
        end
    end
```

```
    always @(negedge rst or negedge bypass_A or negedge bypass_B ) begin

        if(!(rst || bypass_A || bypass_B)) constraintVals.CovCode.start();
        else constraintVals.CovCode.stop();
    end

    always @(posedge rst or posedge bypass_A or posedge bypass_B ) begin
        constraintVals.CovCode.stop();
    end

    always @(posedge clk) begin
       if(!(rst || bypass_A || bypass_B)) constraintVals.CovCode.sample();
    end

    initial begin
        Error_Count = 0;
        Correct_Count = 0;

    //--ALSU1
        assert_reset;
```

```
//========= LOOP1: opcode sequence constraint disabled, all others enabled =========

        constraintVals.OpcodeSequence.constraint_mode(0);
        constraintVals.ALSUsignals.constraint_mode(1);

        repeat(2000) begin
            assert (constraintVals.randomize());
            rst = constraintVals.rst;
            cin = constraintVals.cin;
            red_op_A = constraintVals.red_op_A;
            red_op_B = constraintVals.red_op_B;
            bypass_A = constraintVals.bypass_A;
            bypass_B = constraintVals.bypass_B;
            direction = constraintVals.direction;
            serial_in = constraintVals.serial_in;
            opcode = constraintVals.opcode;
            A = constraintVals.A;
            B = constraintVals.B;

            if (rst) check_reset;
            else begin
                check_result();
            end
        end
            red_op_A = 1;
            red_op_B = 1;
            check_result();
```

```
            // directed case to complete code coverage
            bypass_A = 1;
            bypass_B = 1;
            check_result();

//========= LOOP2: opcode sequence constraint Enabled, all others Disabled =========


        constraintVals.OpcodeSequence.constraint_mode(1);
        constraintVals.ALSUsignals.constraint_mode(0);
        rst = 0;
        red_op_A = 0;
        red_op_B = 0;
        bypass_A = 0;
        bypass_B = 0;

        repeat (100) begin
                assert (constraintVals.randomize());
                cin = constraintVals.cin;
                direction = constraintVals.direction;
                serial_in = constraintVals.serial_in;
                A = constraintVals.A;
                B = constraintVals.B;

                foreach (constraintVals.opcode_sequence[i]) begin
                    opcode = constraintVals.opcode_sequence[i];

                    check_result();
                end
        end
```

```
//====== Directed Test case to complete functional coverage(opcode transition)
        transitionPattern = '{OR, XOR, ADD, MULT, SHIFT, ROTATE};

            foreach (transitionPattern[i]) begin
                opcode = transitionPattern[i];
                constraintVals.opcode = opcode;
                check_result();
            end

        $display("Correct Count: %0d",Correct_Count);
        $display("Error Count: %0d", Error_Count);
        $stop;

    end
```

```
//========= LOOP3: new constraints to hit BinsBitwiseCrossWalkingA bins =========
        constraintVals.OpcodeSequence.constraint_mode(0);
        constraintVals.ALSUsignals.constraint_mode(1);
        repeat(50) begin
            assert (constraintVals.randomize() with{
                A_exp inside{WalkingOnesSequence};
                A == A_exp;
                B == 0;
                rst == 0;
                opcode == opcode_exp;
                red_op_A == 1;
                opcode_exp dist {
                    OR := 50,
                    XOR := 50
                };
            });
            rst = constraintVals.rst;
            cin = constraintVals.cin;
            red_op_A = constraintVals.red_op_A;
            red_op_B = constraintVals.red_op_B;
            bypass_A = constraintVals.bypass_A;
            bypass_B = constraintVals.bypass_B;
            direction = constraintVals.direction;
            serial_in = constraintVals.serial_in;
            opcode = constraintVals.opcode;
            A = constraintVals.A;
            B = constraintVals.B;

            if (rst) check_reset;
            else begin
                check_result();
            end
        end
```

```
//========= LOOP4: new constraints to hit BinsBitwiseCrossWalkingB bins =========
        repeat(50) begin
            assert (constraintVals.randomize() with{
                B_exp inside{WalkingOnesSequence};
                A == 0;
                B == B_exp;
                rst == 0;
                opcode == opcode_exp;
                red_op_A == 0;
                red_op_B == 1;
                opcode_exp dist {
                    OR := 50,
                    XOR := 50
                };
            });
            rst = constraintVals.rst;
            cin = constraintVals.cin;
            red_op_A = constraintVals.red_op_A;
            red_op_B = constraintVals.red_op_B;
            bypass_A = constraintVals.bypass_A;
            bypass_B = constraintVals.bypass_B;
            direction = constraintVals.direction;
            serial_in = constraintVals.serial_in;
            opcode = constraintVals.opcode;
            A = constraintVals.A;
            B = constraintVals.B;

            if (rst) check_reset;
            else begin
                check_result();
            end
        end
```

```
        $display("Correct Count: %0d",Correct_Count);
        $display("Error Count: %0d", Error_Count);
        $stop;

    end
```

```
    task assert_reset();

        rst = 1;
        check_reset();
    endtask

    task check_reset();
        @(negedge clk);
        if (out == 0 && leds == 0) begin
            @(negedge clk);
            if(out == 0 && leds == 0) begin
                Correct_Count++;
                $display("Reset passed");
            end
            else begin
                Error_Count++;
                $display("Reset failed");
                $stop;
            end
        end
        else begin
                Error_Count++;
                $display("Reset failed");
                $stop;
        end
        reset_internals();
    endtask
```

```
task GoldenModel();

    bit valid;
if (rst) begin
    reset_internals();
    out_exp = 'b0;
    leds_exp ='b0;
end

else begin
    check_validity(red_op_A_reg,red_op_B_reg,opcode_reg,valid);

    // leds expected behavior
    if (!valid) begin
        leds_exp = ~leds;
    end
    else begin
        leds_exp = 'b0;
    end

    // out expected behavior
    if(bypass_A_reg) begin
        out_exp = A_reg;
    end
    else if (bypass_B_reg) begin
        out_exp = B_reg;
    end

    else if (!valid) begin
        out_exp = 'b0;
    end

```

```
    else begin
        case (opcode_reg)
            OR: begin
                    if(red_op_A_reg) begin
                        out_exp = |A_reg;
                    end else if (!red_op_A_reg && red_op_B_reg) begin
                        out_exp = |B_reg;
                    end else begin
                        out_exp = A_reg | B_reg;
                end
            end

            XOR: begin
                    if(red_op_A_reg) begin
                        out_exp = ^A_reg;
                    end else if (!red_op_A_reg && red_op_B_reg) begin
                        out_exp = ^B_reg;
                    end else begin
                        out_exp = A_reg ^ B_reg;
                    end
            end

            ADD: begin
                    out_exp = A_reg+B_reg+cin_reg;
            end

            MULT: begin
                    out_exp = A_reg*B_reg;
            end

            SHIFT: begin
                    if (direction_reg) out_exp = {out_exp[4:0], serial_in_reg};
                    else if (!direction_reg) out_exp = {serial_in_reg, out_exp[5:1]};
            end

            ROTATE: begin
                    if (direction_reg) out_exp = {out_exp[4:0], out_exp[5]};
                    else if (!direction_reg) out_exp = {out_exp[0], out_exp[5:1]};
            end
            default: begin
                out_exp = 'b0;
            end
        endcase
    end
end
    update_internals();
endtask
```

```
        task reset_internals();
            cin_reg = 'b0;
            red_op_A_reg = 'b0;
            red_op_B_reg = 'b0;
            bypass_A_reg = 'b0;
            bypass_B_reg = 'b0;
            direction_reg = 'b0;
            serial_in_reg = 'b0;
            opcode_reg = OR;
            A_reg = 'b0;
            B_reg = 'b0;
        endtask

        task update_internals();
            cin_reg = cin;
            red_op_A_reg = red_op_A;
            red_op_B_reg = red_op_B;
            bypass_A_reg = bypass_A;
            bypass_B_reg = bypass_B;
            direction_reg = direction;
            serial_in_reg = serial_in;
            opcode_reg = opcode;
            A_reg = A;
            B_reg = B;
        endtask
```

```
        task check_validity(
            input red_op_A,red_op_B, opcode_e opcode,
            output isValid
        );
            case (opcode)
                INVALID_6,INVALID_7 : isValid = 0;

                ADD,MULT,SHIFT,ROTATE: begin
                    if( red_op_A || red_op_B) isValid = 0;
                    else isValid = 1;
                end

                default: isValid =1;
            endcase

        endtask

        task check_result (
            input rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in,
            signed [2:0] A, B, opcode_e opcode
        );
            GoldenModel();

            @(negedge clk);
            if((out == out_exp)&&(leds == leds_exp)) begin
                Correct_Count++;
            end

            else begin
                if (out != out_exp)
                    $display("Time: %0t, Failed, out = %0h, Exected: %0h",$time,out, out_exp);

                else if (leds != leds_exp)
                    $display("Time: %0t, Failed, leds = %0h, Expected: %0h",$time,leds,leds_exp);

                Error_Count++;
                $stop;
            end
        endtask

    endmodule
```

```
    task check_result ();
        GoldenModel();

        @(negedge clk);
        if((out == out_exp)&&(leds == leds_exp)) begin
            Correct_Count++;
        end

        else begin
            if (out != out_exp)
                $display("Time: %0t, Failed, out = %0h, Exected: %0h",$time,out, out_exp);

            else if (leds != leds_exp)
                $display("Time: %0t, Failed, leds = %0h, Expected: %0h",$time,leds,leds_exp);

            Error_Count++;
            $stop;
        end
    endtask

endmodule
```

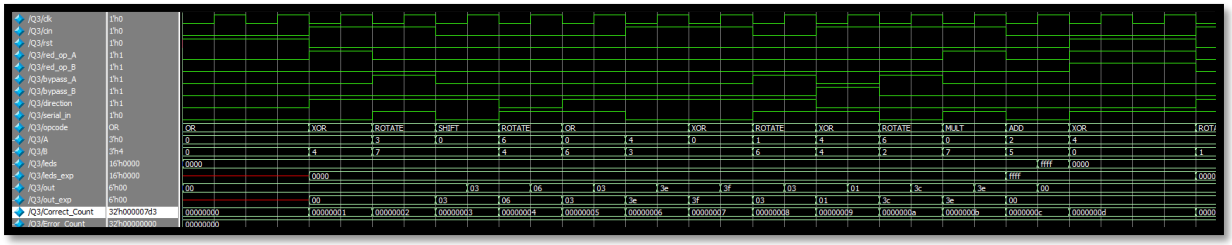
❖ Do file:

```
vlib work
vlog package.sv
vlog ALSU.v ALSU_tb.sv +cover -covercells
vsim -voptargs=+acc ALSU_tb -cover
add wave *
coverage save ALSU_tb.ucdb -onexit
run -all
```

❖ Results:

```
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Correct Count: 2003
# Error Count: 0
# ** Note: $stop    : Assignment2.sv(236)
#    Time: 4182 ns  Iteration: 1  Instance: /Q3
```

Waveform:



❖ Functional Coverage Report:

INST \/Assignment3::ALSUconstraints::CovCode	100.00%	100	100.00...
CVP A_cp	100.00%	100	100.00...
CVP A_WalkingOnes_cp	100.00%	100	100.00...
CVP B_cp	100.00%	100	100.00...
CVP B_WalkingOnes_cp	100.00%	100	100.00...
CVP ALU_cp	100.00%	100	100.00...
CVP red_op_A_cp	100.00%	100	100.00...
CVP red_op_B_cp	100.00%	100	100.00...
CVP serial_in	100.00%	100	100.00...
CVP direction	100.00%	100	100.00...
CVP cin	100.00%	100	100.00...
CROSS addMultCrossAB	100.00%	100	100.00...
CROSS addCrossCin	100.00%	100	100.00...
CROSS shiftRotateCrossDir	100.00%	100	100.00...
CROSS shiftCrossShiftin	100.00%	100	100.00...
CROSS BinsBitwiseCrossWalkingA	100.00%	100	100.00...
CROSS BinsBitwiseCrossWalkingB	100.00%	100	100.00...
CROSS Bins_invalid_reduction	100.00%	100	100.00...

CROSS addMultCrossAB	100.00%	100	100.00...
bin <Bins_arith[MULT],A_data_min,B_data_min>	3	1	100.00...
bin <Bins_arith[ADD],A_data_min,B_data_min>	4	1	100.00...
bin <Bins_arith[MULT],A_data_max,B_data_min>	4	1	100.00...
bin <Bins_arith[ADD],A_data_max,B_data_min>	2	1	100.00...
bin <Bins_arith[MULT],A_data_0,B_data_min>	3	1	100.00...
bin <Bins_arith[ADD],A_data_0,B_data_min>	2	1	100.00...
bin <Bins_arith[MULT],A_data_min,B_data_max>	6	1	100.00...
bin <Bins_arith[ADD],A_data_min,B_data_max>	5	1	100.00...
bin <Bins_arith[MULT],A_data_min,B_data_0>	5	1	100.00...
bin <Bins_arith[ADD],A_data_min,B_data_0>	3	1	100.00...
bin <Bins_arith[MULT],A_data_max,B_data_max>	3	1	100.00...
bin <Bins_arith[ADD],A_data_max,B_data_max>	2	1	100.00...
bin <Bins_arith[MULT],A_data_max,B_data_0>	4	1	100.00...
bin <Bins_arith[ADD],A_data_max,B_data_0>	3	1	100.00...
bin <Bins_arith[MULT],A_data_0,B_data_max>	5	1	100.00...
bin <Bins_arith[ADD],A_data_0,B_data_max>	10	1	100.00...
bin <Bins_arith[MULT],A_data_0,B_data_0>	3	1	100.00...
bin <Bins_arith[ADD],A_data_0,B_data_0>	6	1	100.00...

CROSS addCrossCin	100.00%	100	100.00...
bin <Bins_arith[ADD],auto[1]>	131	1	100.00...
bin <Bins_arith[ADD],auto[0]>	133	1	100.00...
ignore_bin notADD2	0	-	-
ignore_bin notADD	0	-	-

CROSS shiftRotateCrossDir	100.00%	100	100.00...	✓
bin <Bins_shift[ROTATE],auto[1]>	125	1	100.00...	✓
bin <Bins_shift[SHIFT],auto[1]>	128	1	100.00...	✓
bin <Bins_shift[ROTATE],auto[0]>	144	1	100.00...	✓
bin <Bins_shift[SHIFT],auto[0]>	107	1	100.00...	✓
ignore_bin notShiftRotate	1128	-	-	✓

CROSS shiftCrossShiftin	100.00%	100	100.00...	✓
bin <Bins_shift[SHIFT],auto[1]>	125	1	100.00...	✓
bin <Bins_shift[SHIFT],auto[0]>	110	1	100.00...	✓
ignore_bin notShift	0	-	-	✓
ignore_bin notShiftBins	0	-	-	✓

CROSS BinsBitwiseCrossWalkingA	100.00%	100	100.00...	✓
bin <Bins_bitwise[XOR],B_data_0,A_data_walkingones100>	19	1	100.00...	✓
bin <Bins_bitwise[OR],B_data_0,A_data_walkingones100>	9	1	100.00...	✓
bin <Bins_bitwise[XOR],B_data_0,A_data_walkingones010>	8	1	100.00...	✓
bin <Bins_bitwise[OR],B_data_0,A_data_walkingones010>	8	1	100.00...	✓
bin <Bins_bitwise[XOR],B_data_0,A_data_walkingones001>	10	1	100.00...	✓
bin <Bins_bitwise[OR],B_data_0,A_data_walkingones001>	6	1	100.00...	✓
ignore_bin BnotZero	28	-	-	✓
ignore_bin notBitwise	10	-	-	✓

CROSS BinsBitwiseCrossWalkingB	100.00%	100	100.00...	✓
bin <Bins_bitwise[XOR],A_data_0,B_data_walkingones100>	14	1	100.00...	✓
bin <Bins_bitwise[OR],A_data_0,B_data_walkingones100>	4	1	100.00...	✓
bin <Bins_bitwise[XOR],A_data_0,B_data_walkingones010>	6	1	100.00...	✓
bin <Bins_bitwise[OR],A_data_0,B_data_walkingones010>	9	1	100.00...	✓
bin <Bins_bitwise[XOR],A_data_0,B_data_walkingones001>	7	1	100.00...	✓
bin <Bins_bitwise[OR],A_data_0,B_data_walkingones001>	7	1	100.00...	✓
ignore_bin BnotZero	16	-	-	✓
ignore_bin notBitwise	5	-	-	✓

CROSS Bins_invalid_reduction	100.00%	100	100.00...	✓
bin <Bins_arith[MULT],red_op_A_bins,red_op_B_bins>	12	1	100.00...	✓
bin <Bins_shift[ROTATE],red_op_A_bins,red_op_B_bins>	6	1	100.00...	✓
bin <Bins_arith[ADD],red_op_A_bins,red_op_B_bins>	6	1	100.00...	✓
bin <Bins_shift[SHIFT],red_op_A_bins,red_op_B_bins>	7	1	100.00...	✓
ignore_bin transition_cp	0	-	-	✓
ignore_bin valid_B_reduction	81	-	-	✓
ignore_bin valid_A_reduction	81	-	-	✓

❖ Coverage Report:

```
Coverage Report by instance with details

================================================================================
=== Instance: /\Q3#DUT
=== Design Unit: work.ALSU
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        31        31         0   100.00%

================================Branch Details================================
```

- Excluded **all False case** in the case statement, since invalid opcodes are handled by the flag (**invalid_opcode**)

```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       6         6         0   100.00%
```

```
Expression Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Expressions                      8         8         0   100.00%
```

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

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

```
Total Coverage By Instance (filtered view): 100.00%
```

Q2) Write Assertions:

```
//=============Q2-1================
property P1;
    @(posedge clk) a |-> ##2 b;
endproperty
A1: assert property (P1)
    else $error("Assertion A1 failed!");


//=============Q2-2================
property P2;
    @(posedge clk) (a&&b) |-> ##[1:3] c;
endproperty
A2: assert property (P2)
    else $error("Assertion A2 failed!");

//=============Q2-3================
sequence s11b
    ##2 (b==0);
endsequence

//=============Q2-4A================
sequence Y_exp
    Y == (8'b0000_0001 ||8'b0000_0010 ||8'b0000_0100 ||8'b0000_1000 ||
          8'b0001_0000 ||8'b0010_0000 ||8'b0100_0000 ||8'b1000_0000 );
endsequence
property Y;
    @(posedge clk) Y_exp;
endproperty
Check_Y: assert property (Y)
    else $error("Assertion Check_Y failed!");

//=============Q2-4B================

property valid_check;
    @(posedge clk) (~|D) => !valid;
endproperty

Check_vaild: assert property (valid_check)
    else $error("Assertion Check_vaild failed!");
```

Q3) Counter:

❖ Verification Plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
Counter1	When reset is asserted, Output should be low, and **zero** should be high	Directed at the start of the simulation, then randomized with a constraint to be inactive 90% of the time during the simulation	-	Immediate assertion to verify Async reset functionality, concurrent assertion to check ZERO functionality
Counter2	When **load_n** is low, **count_out** should take the value of **load_data** input	Randomization with constraint on **load_n** to be high 70% of simulation time	Cover all values of load_data	Concurrent assertion to check the load_data
Counter3	Counter should only increment or decrement if **rst_n** is inactive and, **ce** signal is high, else keep the current **count_out** value.	Randomization with constraint on **ce** to be high 70% of simulation time	Cover all values of count_out& transition bin from max to zero	Concurrent assertion to check the count_out freeze
Counter4	If rst_n is disabled and ce is enabled, if: **up_down = o → decrement** **Up_down = 1 → increment**	Randomization with constraint on **up_down** to be high 50% of simulation time	Bin for decrement & another for increment	2 Concurrent assertions to check the decrement and increment functionalities
Counter5	Check that when the bus **count_out** value equals the max possible value, **max_count** should be high	Randomized with no constraints	Bin for max_count	Concurrent assertion to check the max_count functionality

❖ Constraints class:

```
package A2;
class CounterConstraints;
//Counter

class CounterConstraints;

parameter WIDTH = `WIDTH;
rand bit rst_n;
rand bit load_n;
rand bit up_down;
rand bit ce;
rand bit [WIDTH-1:0] data_load;

function new();
    $display("WIDTH = %0d",WIDTH);
endfunction
```

```
covergroup CovCode ;

    laod_data_cp: coverpoint data_load iff(rst_n && (!load_n)) ;
    count_outIncrement_cp: coverpoint count_out iff(rst_n && load_n && ce && up_down);

    count_outOverflow_cp: coverpoint count_out iff(rst_n && load_n && ce && up_down){
        bins OV = (MAX => 0);
    }

    count_outDecrement_cp: coverpoint count_out iff(rst_n && load_n && ce && !up_down);

    count_outUnderflow_cp: coverpoint count_out iff(rst_n && load_n && ce && !up_down){
        bins UV = (0 => MAX);
    }
endgroup
```

```
constraint CounterSignals{

    rst_n dist {
        'b0 := 10,
        'b1 := 90
    };
    load_n dist {
        'b1 := 90,
        'b0 := 10
    };
    ce dist {
        'b0 := 10,
        'b1 := 90
    };
    up_down dist {
        'b0 := 50,
        'b1 := 50
    };
    data_load !=0;
}

```

```
    function new();
        CovCode = new();
    endfunction
endclass
```

❖ SVA module:

```
module counter_sva(clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
    parameter WIDTH = 4;
    input clk;
    input rst_n;
    input load_n;
    input up_down;
    input ce;
    input [WIDTH-1:0] data_load;
    input [WIDTH-1:0] count_out;
    input max_count;
    input zero;

    reg [WIDTH-1:0] count_out_prev;

    always @(posedge clk or negedge rst_n) begin
        count_out_prev <= count_out;
    end

// //=============Reset immediate assertion===============

    always_comb begin
        if(!rst_n) begin
            Reset: assert final (count_out === 'b0)
            else $error("Assertion Reset failed!");
        end
    end
```

```
//=========Checking zero flag functionality=============

    property Zero;
      @(negedge clk) (count_out ==={ WIDTH{1'b0} }) |-> zero;
    endproperty

    zero_flag_a: assert property (Zero)
        else $error("Assertion zero_flag failed!");

    zero_flag_c: cover property (Zero);
//=========Checking max flag functionality=============
  property Max;
    @(negedge clk) (count_out ==={ WIDTH{1'b1} }) |-> max_count;
  endproperty
    max_flag_a: assert property (Max)
      else $error("Assertion max_count_flag failed!");

    max_flag_c: cover property (Max);
```

```
//================Load functionality====================
    property LoadData;
        @(negedge clk) (!load_n && rst_n) |-> (count_out == data_load);
    endproperty

    Load_a: assert property (LoadData)
        else $error("Assertion Load_a failed!");

    Load_c: cover property (LoadData);

//========no load or ce or rst => counter freeze=======

    property Counter_Freeze;
        @(negedge clk) (load_n && rst_n && !ce) |-> (count_out == count_out_prev);
    endproperty

    Counter_Freeze_a: assert property (Counter_Freeze)
        else $error("Assertion Counter_Freeze_a failed!");

    Counter_Freeze_c: cover property (Counter_Freeze);
```

```
//=============Increment functionality===============
    sequence increment_en;
        load_n && rst_n && ce && up_down;
    endsequence
    property Increment;
        @(negedge clk) increment_en |-> (count_out == count_out_prev +1'b1);  // 1 only is 32 bit, causing error
    endproperty

    Increment_a: assert property (Increment)
        else $error("Assertion Increment_a failed!");

    Increment_c: cover property (Increment);

//=============Decrement functionality===============
    sequence decrement_en;
        (load_n && rst_n) && (ce==1) && (up_down==0);
    endsequence
    property Decrement;
        @(negedge clk) decrement_en |-> (count_out == count_out_prev - 1'b1);
    endproperty

    decrement_a: assert property (Decrement)
        else $error("Assertion decrement_a failed!");

    Decrement_c: cover property (Decrement);

endmodule
```

❖ Counter interface:

```
interface counter_if(input bit clk);
    input rst_n;
    input load_n;
    input up_down;
    input ce;
    input [3:0] data_load;
    output [3:0] count_out;
    output reg max_count;
    output zero;


    modport TEST(
        output logic rst_n,load_n,up_down,ce,data_load,
        input logic count_out,max_count,zero
        );

    modport DUT(
        input rst_n,load_n,up_down,ce,data_load,
        output count_out,max_count,zero
        );

endinterface
```

❖ Top module:

```
module top();
    bit clk;

    always #1 clk = ~clk;

    counter_if countIF(clk);

    counter DUT(
        .clk(countIF.clk) ,.rst_n(countIF.rst_n), .load_n(countIF.load_n),
        .up_down(countIF.up_down), .ce(countIF.ce), .data_load(countIF.data_load),
        .count_out(countIF.count_out), .max_count(countIF.max_count), .zero(countIF.zero)
    );

    counter_tb tb(countIF);

    bind counter counter_sva countSVA (
        countIF.clk,countIF.rst_n,countIF.load_n,
        countIF.up_down,countIF.ce,countIF.data_load,countIF.count_out,
        countIF.max_count,countIF.zero
    );

endmodule
```

❖ Test bench:

```
module counter_tb(counter_if.TEST counterIF);

    CounterConstraints constraintVals = new;

    int Error_Count,Correct_Count;

    initial begin
        Error_Count = 0;
        Correct_Count = 0;

    //--Counter1
        counterIF.load_n = 0;
        counterIF.data_load = 'hf;
        assert_reset;

    //--Counter2,3,4,5,6

        repeat(1000) begin

            assert (constraintVals.randomize());
            counterIF.rst_n = constraintVals.rst_n;
            counterIF.load_n = constraintVals.load_n;
            counterIF.ce = constraintVals.ce;
            counterIF.up_down = constraintVals.up_down;
            counterIF.data_load = constraintVals.data_load;
            @(negedge counterIF.clk);
            constraintVals.count_out = counterIF.count_out;
            constraintVals.CovCode.sample();
        end
```

```
        // directed test to complete functional coverage (load zero to the counter)
            counterIF.rst_n = 1;
            constraintVals.rst_n = counterIF.rst_n;
            counterIF.load_n = 0;
            constraintVals.load_n = counterIF.load_n;
            counterIF.data_load = 0;
            constraintVals.data_load = counterIF.data_load;
            @(negedge counterIF.clk);
            constraintVals.CovCode.sample();

        $display("Correct Count: %0d",Correct_Count);
        $display("Error Count: %0d", Error_Count);
        $stop;
    end
```

```
    task assert_reset ;
    counterIF.rst_n = 0;
    @(negedge counterIF.clk);
    if ((counterIF.count_out != 'b0)&&(counterIF.zero != 1'b1))begin
        $display("Reset failed");
        Error_Count++;
    end
    else begin
        $display("Reset Passed");
        Correct_Count++;
    end
    counterIF.rst_n = 1;
    //@(negedge counterIF.clk);
    endtask

endmodule
```

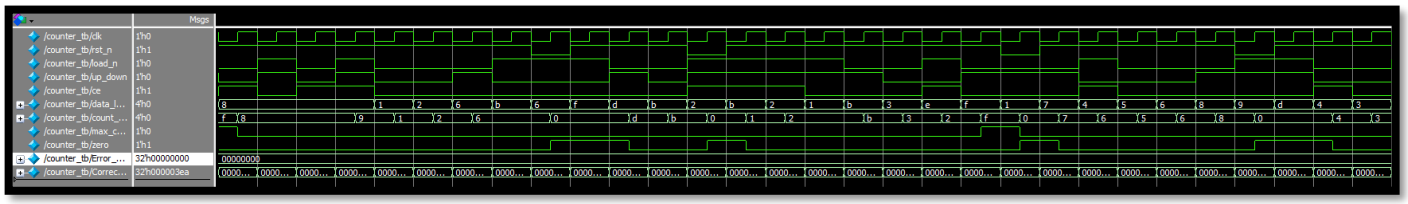
❖ Do file:

```
run.do
vlib work
vlog package.sv counter_tb.sv
vlog counter.v counter_tb_SVA.sv counter_top.sv +cover -covercells
vsim -voptargs=+acc top -cover
add wave -position insertpoint sim:/top/tb/counterIF/*
coverage save counter_tb.ucdb -onexit
run -all
```

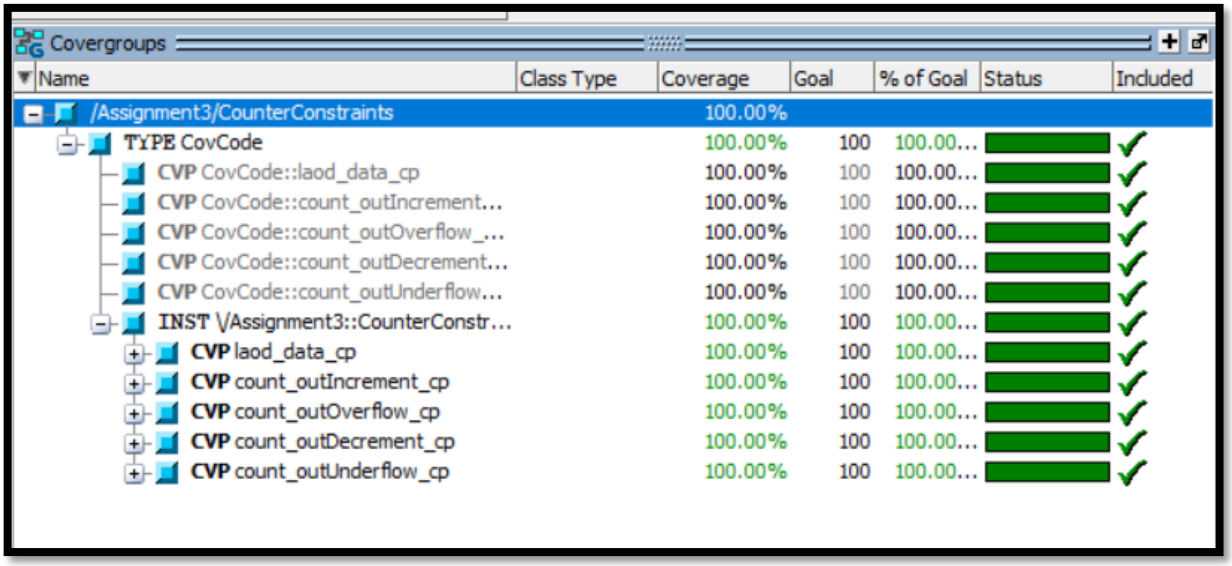
❖ Result:

```
# Reset Passed
# Correct Count: 1002
# Error Count: 0
# ** Note: $stop    : ./Assignment/counter_tb.sv(61)
```

Waveform:



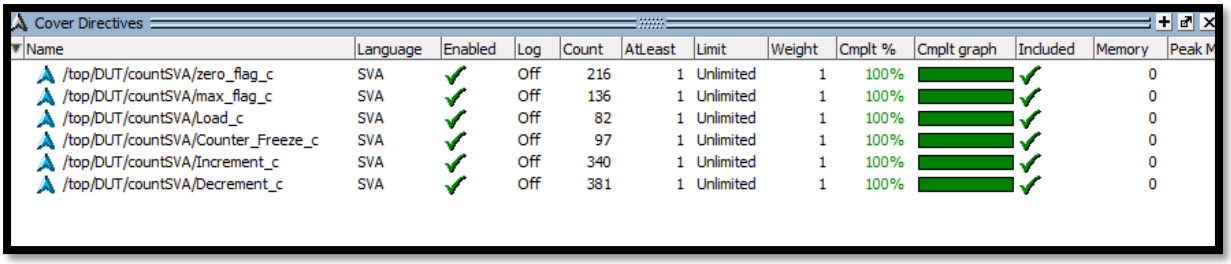
❖ Functional Coverage:



Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/Assignment3/CounterConstraints		100.00%				
TYPE CovCode		100.00%	100	100.00...		✓
CVP CovCode::laod_data_cp		100.00%	100	100.00...		✓
CVP CovCode::count_outIncrement...		100.00%	100	100.00...		✓
CVP CovCode::count_outOverflow_...		100.00%	100	100.00...		✓
CVP CovCode::count_outDecrement...		100.00%	100	100.00...		✓
CVP CovCode::count_outUnderflow...		100.00%	100	100.00...		✓
INST \/Assignment3::CounterConstr...		100.00%	100	100.00...		✓
CVP laod_data_cp		100.00%	100	100.00...		✓
CVP count_outIncrement_cp		100.00%	100	100.00...		✓
CVP count_outOverflow_cp		100.00%	100	100.00...		✓
CVP count_outDecrement_cp		100.00%	100	100.00...		✓
CVP count_outUnderflow_cp		100.00%	100	100.00...		✓

❖ Assertion coverage:



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak M
/top/DUT/countSVA/zero_flag_c	SVA	✓	Off	216	1	Unlimited	1	100%		✓	0	
/top/DUT/countSVA/max_flag_c	SVA	✓	Off	136	1	Unlimited	1	100%		✓	0	
/top/DUT/countSVA/Load_c	SVA	✓	Off	82	1	Unlimited	1	100%		✓	0	
/top/DUT/countSVA/Counter_Freeze_c	SVA	✓	Off	97	1	Unlimited	1	100%		✓	0	
/top/DUT/countSVA/Increment_c	SVA	✓	Off	340	1	Unlimited	1	100%		✓	0	
/top/DUT/countSVA/Decrement_c	SVA	✓	Off	381	1	Unlimited	1	100%		✓	0	

❖ Code Coverage :

```
Coverage Report by instance with details

================================================================================
=== Instance: /\Q2#DUT
=== Design Unit: work.counter
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       7         7         0   100.00%

================================Statement Details================================
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         30        30         0   100.00%

==============================Toggle Details==============================

Toggle Coverage for instance /\Q2#DUT  --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                            ce           1           1      100.00
                                           clk           1           1      100.00
                                   count_out[3-0]        1           1      100.00
                                   data_load[0-3]        1           1      100.00
                                          load_n         1           1      100.00
                                       max_count         1           1      100.00
                                           rst_n         1           1      100.00
                                         up_down         1           1      100.00
                                            zero         1           1      100.00

Total Node Count     =         15
Toggled Node Count   =         15
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (30 of 30 bins)


Total Coverage By Instance (filtered view): 100.00%
```

Q4) Configuration Register:

❖ Verification Plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
Config1	Check reset functionality by asserting reset and compare each register value to its corresponding in reset_assoc[] array.	Directed at the start of the simulation	-	A checker in the testbench to ensure correct reset values for every register
config2	For each register, write the inverted value for its default reset value, so that any unwritable bit is detected	Directed during the simulation	-	A checker in the testbench to ensure correct values for every register

❖ TestBench:

```
typedef enum logic [2:0]{
    adc0_reg = 0,adc1_reg =1,
    temp_sensor0_reg=2,temp_sensor1_reg=3,
    analog_test=4,digital_test=5,
    amp_gain=6,digital_config=7
} reg_addr_e;

typedef logic[15:0] register_word_t;

module config_reg_tb();

    bit clk,rst,write;
    reg_addr_e reg_addr;
    register_word_t data_in,data_out;
    register_word_t reset_assoc[string];
    register_word_t reset_toggled_assoc[string];
    register_word_t current_reg_value_assoc[string];
    int Error_count, Correct_count;

    always #1  clk = ~clk;

    config_reg DUT(
        .clk(clk),.reset(rst),.write(write),
        .data_in(data_in),.data_out(data_out),.address(reg_addr)
    );
```

```
    initial begin

        //==========Reset Values==============

        reset_assoc["adc0_reg"] = 16'hFFFF;
        reset_assoc["adc1_reg"] = 16'h0000;
        reset_assoc["temp_sensor0_reg"] = 16'h0000;
        reset_assoc["temp_sensor1_reg"] = 16'h0000;
        reset_assoc["analog_test"] = 16'hABCD;
        reset_assoc["digital_test"] = 16'h0000;
        reset_assoc["amp_gain"] = 16'h0000;
        reset_assoc["digital_config"] = 16'h0001;

        //==========Inverted Values==============
        reset_toggled_assoc["adc0_reg"] = ~reset_assoc["adc0_reg"];
        reset_toggled_assoc["adc1_reg"] = ~reset_assoc["adc1_reg"];
        reset_toggled_assoc["temp_sensor0_reg"] = ~reset_assoc["temp_sensor0_reg"];
        reset_toggled_assoc["temp_sensor1_reg"] = ~reset_assoc["temp_sensor1_reg"];
        reset_toggled_assoc["analog_test"] = ~reset_assoc["analog_test"];
        reset_toggled_assoc["digital_test"] =~reset_assoc["digital_test"];
        reset_toggled_assoc["amp_gain"] = ~reset_assoc["amp_gain"];
        reset_toggled_assoc["digital_config"] = ~reset_assoc["digital_config"];

        //==============Config1=================
        assert_reset();

        //==============Config2=================
        toggle_current_values();

        $display("Error count: %0d",Error_count);
        $display("Correct count: %0d",Correct_count);
        $stop;
    end
```

```
//==================Tasks====================

    task assert_reset();
        rst = 1;
        read_current_values();
        check_result("R");
        rst = 0;
        @(negedge clk);

    endtask

    task toggle_current_values();
        write = 1;
        reg_addr = adc0_reg;
       for (int i=0; i<=reg_addr.last(); i++) begin
           $display(
                "Writing on reg: %0d (%0s) -> %0h ",
                reg_addr,reg_addr.name,reset_toggled_assoc[reg_addr.name]
            );
           data_in = reset_toggled_assoc[reg_addr.name];
           @(negedge clk);
           reg_addr = reg_addr.next;
       end
        read_current_values();
        check_result("T");

    endtask

    task read_current_values();
       reg_addr = adc0_reg;
       @(negedge clk);
       for (int i=0; i<=reg_addr.last(); i++) begin
           current_reg_value_assoc[reg_addr.name] = data_out;
           reg_addr = reg_addr.next;
           @(negedge clk);
       end
    endtask

    task check_result(string mode);
            reg_addr = adc0_reg;

            if(mode == "R") begin
                for (int i=0; i<=reg_addr.last(); i++) begin
                    if (current_reg_value_assoc[reg_addr.name] !== reset_assoc[reg_addr.name]) begin
                        $display(
                            "Failed for Register %0s, Actual Value = %0h,Expected: %0h",
                            reg_addr.name,current_reg_value_assoc[reg_addr.name],reset_assoc[reg_addr.name]
                        );
                        Error_count++;
                    end else begin
                        Correct_count++;
                    end
                    reg_addr = reg_addr.next;
                end
            end
            else if (mode == "T") begin
                for (int i=0; i<=reg_addr.last(); i++) begin
                    if (current_reg_value_assoc[reg_addr.name] !== reset_toggled_assoc[reg_addr.name]) begin
                        $display(
                            "Failed for Register %0s, Actual Value = %0h,Expected: %0h",
                            reg_addr.name,current_reg_value_assoc[reg_addr.name],reset_toggled_assoc[reg_addr.name]
                        );
                        Error_count++;
                    end else begin
                        Correct_count++;
                    end
                    reg_addr = reg_addr.next;
                end
            end

    endtask

endmodule
```

❖ Fixed Design:

```
module my_mem(
input clk,
input write,
input read,
input [7:0] data_in,
input [15:0] address,
output reg [8:0] data_out    // was [7:0] parity bit always ignored
);

// Declare a 9-bit associative array using the logic data type & the key of int datatype
    logic [8:0] mem_array [int];

    always @(posedge clk) begin
        if (write)
            mem_array[address] = {~^data_in, data_in};
        else if (read)
            data_out = mem_array[address];
        end

endmodule
```

❖ Do file:

runQ4.do

```
vlib work
vlog config_reg_buggy_questa.svp config_reg_tb.sv  +cover -covercells
vsim -voptargs=+acc config_reg_tb -cover
add wave *
coverage save config_reg_tb.ucdb -onexit
run -all
```

❖ Results:

```
# Failed for Register analog_test, Actual Value = abcc,Expected: abcd
# Writing on reg: 0 (adc0_reg) -> 0
# Writing on reg: 1 (adc1_reg) -> ffff
# Writing on reg: 2 (temp_sensor0_reg) -> ffff
# Writing on reg: 3 (temp_sensor1_reg) -> ffff
# Writing on reg: 4 (analog_test) -> 5432
# Writing on reg: 5 (digital_test) -> ffff
# Writing on reg: 6 (amp_gain) -> ffff
# Writing on reg: 7 (digital_config) -> fffe
# Failed for Register adc0_reg, Actual Value = fffe,Expected: 0
# Failed for Register adc1_reg, Actual Value = feff,Expected: ffff
# Failed for Register temp_sensor0_reg, Actual Value = fffc,Expected: ffff
# Failed for Register temp_sensor1_reg, Actual Value = fffe,Expected: ffff
# Failed for Register analog_test, Actual Value = fffe,Expected: 5432
# Failed for Register amp_gain, Actual Value = fffe,Expected: ffff
# Failed for Register digital_config, Actual Value = 7ffe,Expected: fffe
# Error count: 8
# Correct count: 8
# ** Note: $stop    : config_reg_tb.sv(58)
#    Time: 54 ns  Iteration: 1  Instance: /config_reg_tb
```

❖ Bug:

Resigter	Stimulus	Expected output	Actual output
Adc0_reg	Toggled reset default value	‘h0000	‘hFFFE
Adc1_reg	Toggled reset default value	‘hFFFF	‘hFEFF
Temp_sensor0_reg	Toggled reset default value	‘hFFFF	‘hFFFC
Temp_sensor0_reg	Toggled reset default value	‘hFFFF	‘hFFFE
analog_test	reset default value	‘hABCD	‘hABCC
digital_test	Toggled reset default value	‘hFFFF	‘hFFFE
amp_gain	Toggled reset default value	‘hFFFF	‘hFFFE
digital_config	Toggled reset default value	‘hFFFE	‘h7FFe