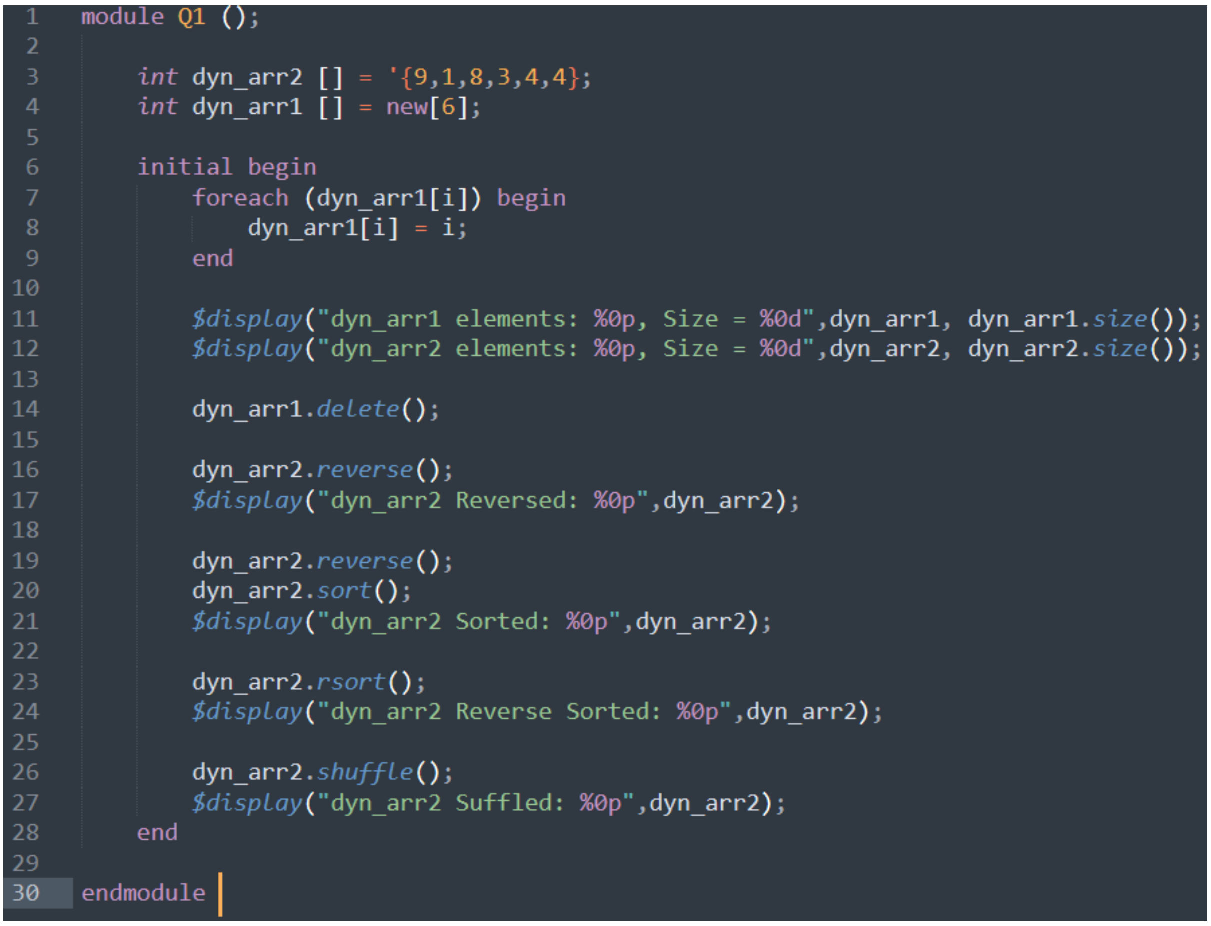
Digital Verification using SV and UVM
Assignment-2

Name: Fares Khalaf Sultan

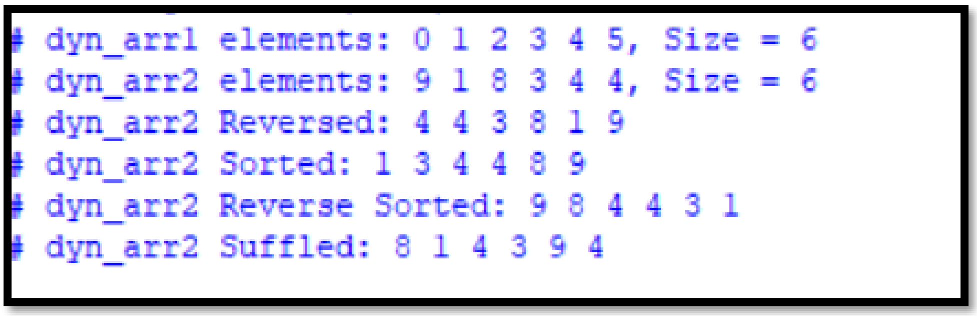
Q1) Dynamic Arrays

❖ Code:



```
module Q1 ();

    int dyn_arr2 [] = '{9,1,8,3,4,4};
    int dyn_arr1 [] = new[6];

    initial begin
        foreach (dyn_arr1[i]) begin
            dyn_arr1[i] = i;
        end

        $display("dyn_arr1 elements: %0p, Size = %0d",dyn_arr1, dyn_arr1.size());
        $display("dyn_arr2 elements: %0p, Size = %0d",dyn_arr2, dyn_arr2.size());

        dyn_arr1.delete();

        dyn_arr2.reverse();
        $display("dyn_arr2 Reversed: %0p",dyn_arr2);

        dyn_arr2.reverse();
        dyn_arr2.sort();
        $display("dyn_arr2 Sorted: %0p",dyn_arr2);

        dyn_arr2.rsort();
        $display("dyn_arr2 Reverse Sorted: %0p",dyn_arr2);

        dyn_arr2.shuffle();
        $display("dyn_arr2 Suffled: %0p",dyn_arr2);
    end

endmodule
```

❖ Simulation Transcript:



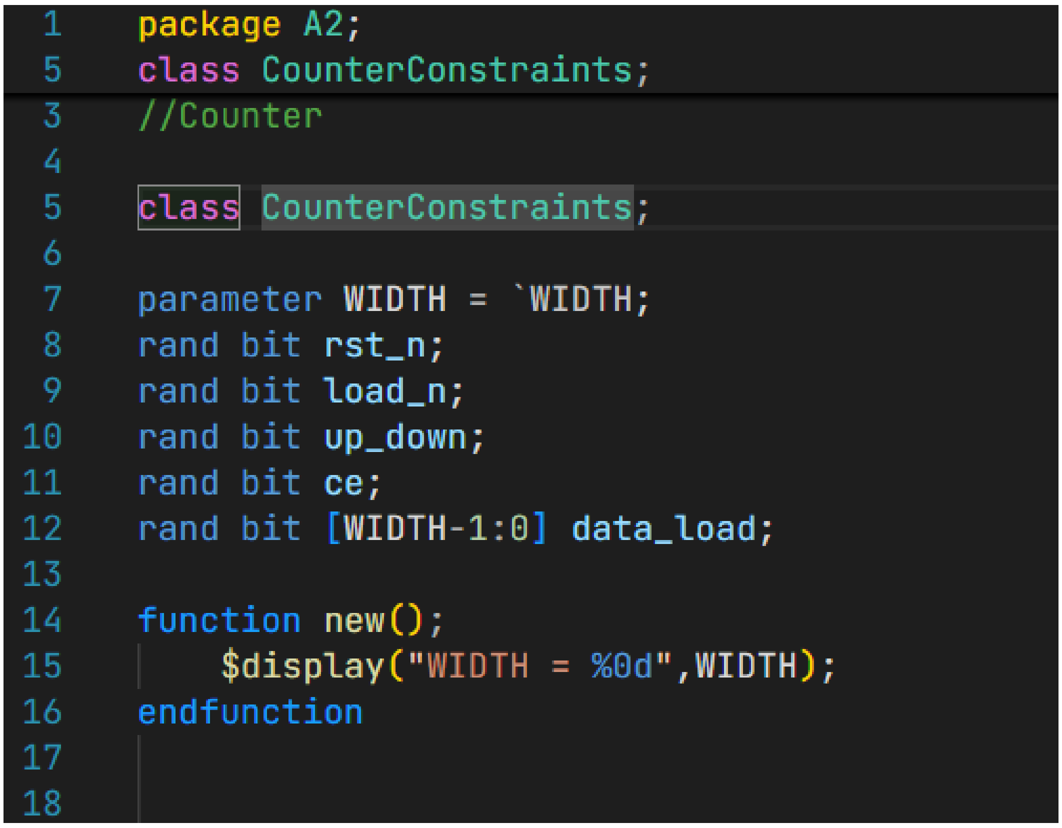
```
# dyn_arr1 elements: 0 1 2 3 4 5, Size = 6
# dyn_arr2 elements: 9 1 8 3 4 4, Size = 6
# dyn_arr2 Reversed: 4 4 3 8 1 9
# dyn_arr2 Sorted: 1 3 4 4 8 9
# dyn_arr2 Reverse Sorted: 9 8 4 4 3 1
# dyn_arr2 Suffled: 8 1 4 3 9 4
```

Q2) Counter:

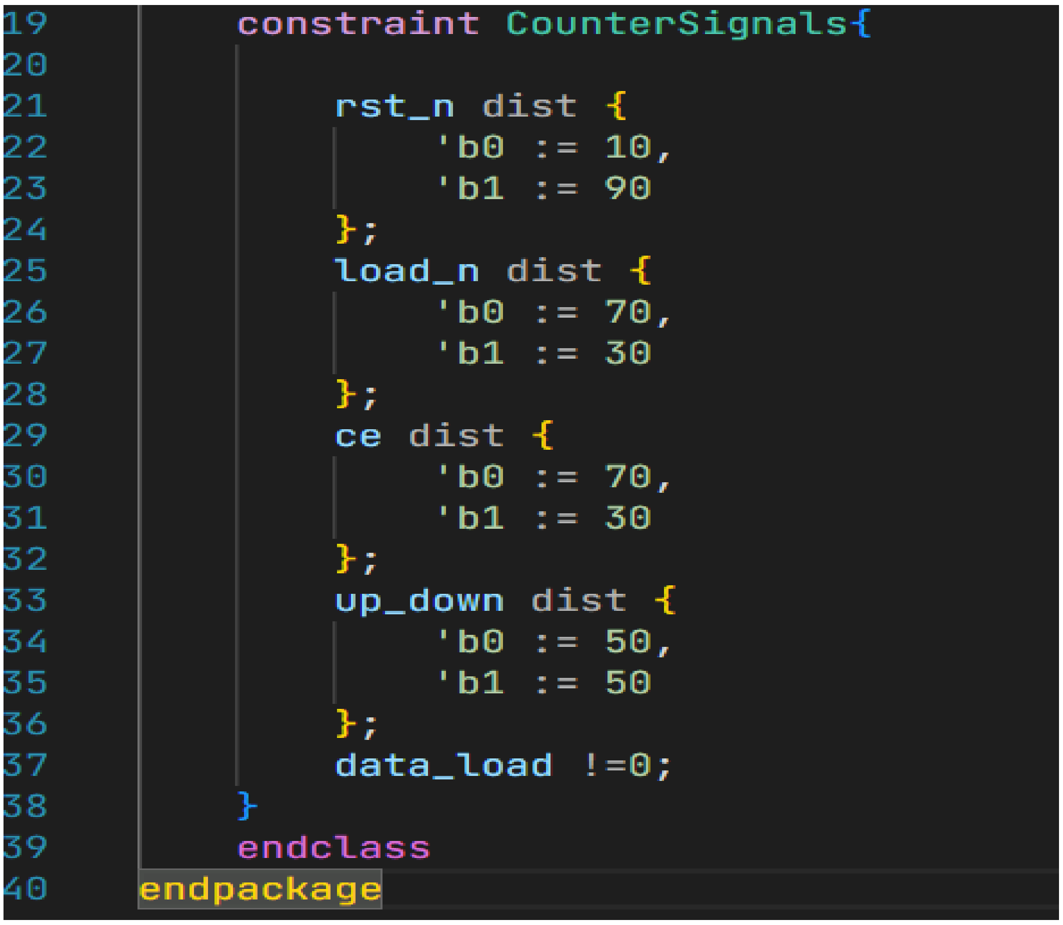
❖ Verification Plan:

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 constraint to be inactive 90% of time during the simulation	-	A checker in the testbench to make sure the output is correct
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	-	Output Checked against golden model
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	-	Output Checked against golden model
Counter4	If rst_n is disabled and ce is enabeled, if: **up_down = o → decrement** **Up_down = 1 → increment**	Randomization with constraint on **up_down** to be high 50% of simulation time	-	Output Checked against golden model
Counter5	Check that when the bus **count_out** value equals the max possible value, **max_count** should be high	Randomized	-	Output Checked against golden model

❖ Constraints class:

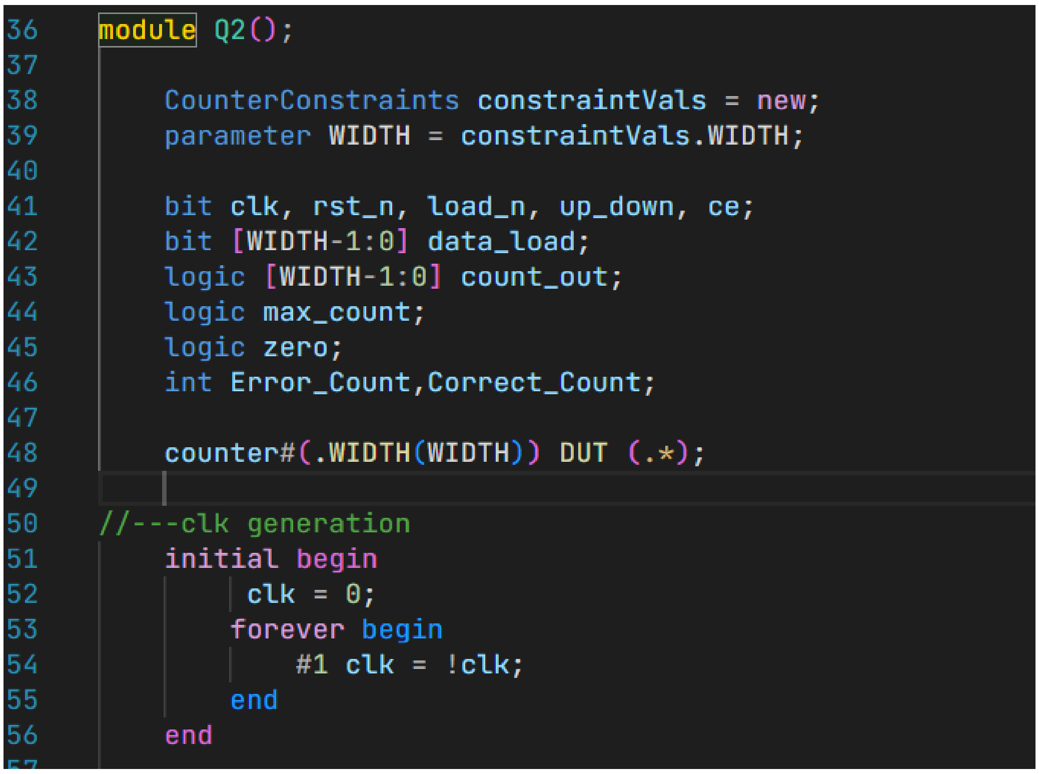


```
package A2;
class CounterConstraints;
//Counter

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


```

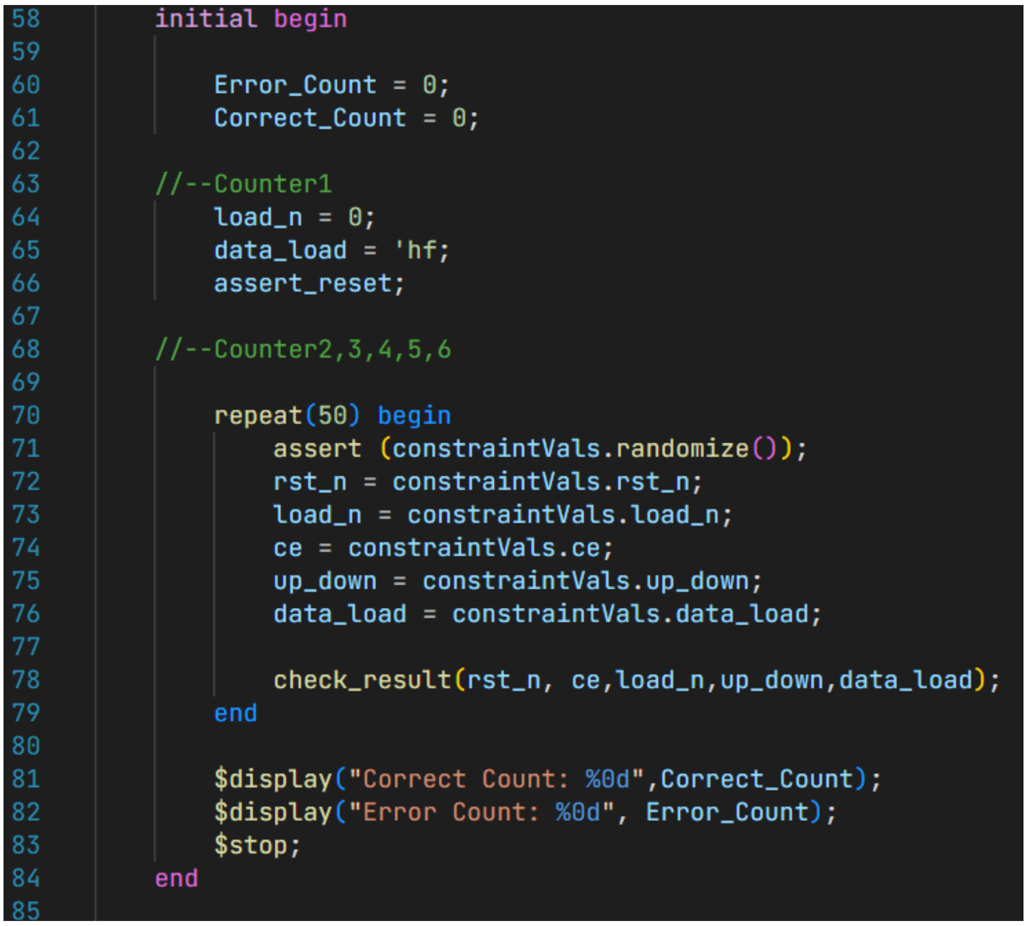


```
        constraint CounterSignals{

            rst_n dist {
                'b0 := 10,
                'b1 := 90
            };
            load_n dist {
                'b0 := 70,
                'b1 := 30
            };
            ce dist {
                'b0 := 70,
                'b1 := 30
            };
            up_down dist {
                'b0 := 50,
                'b1 := 50
            };
            data_load !=0;
        }
        endclass
endpackage
```

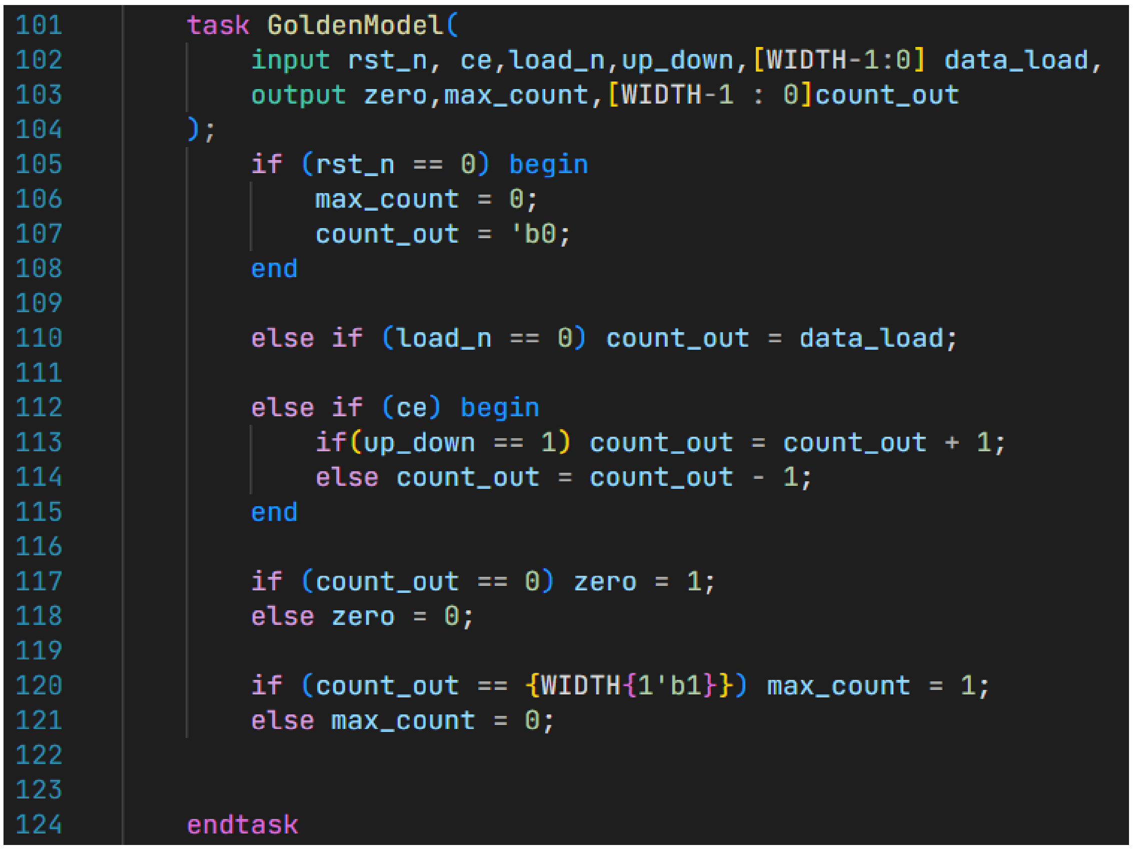
❖ **Test bench:**



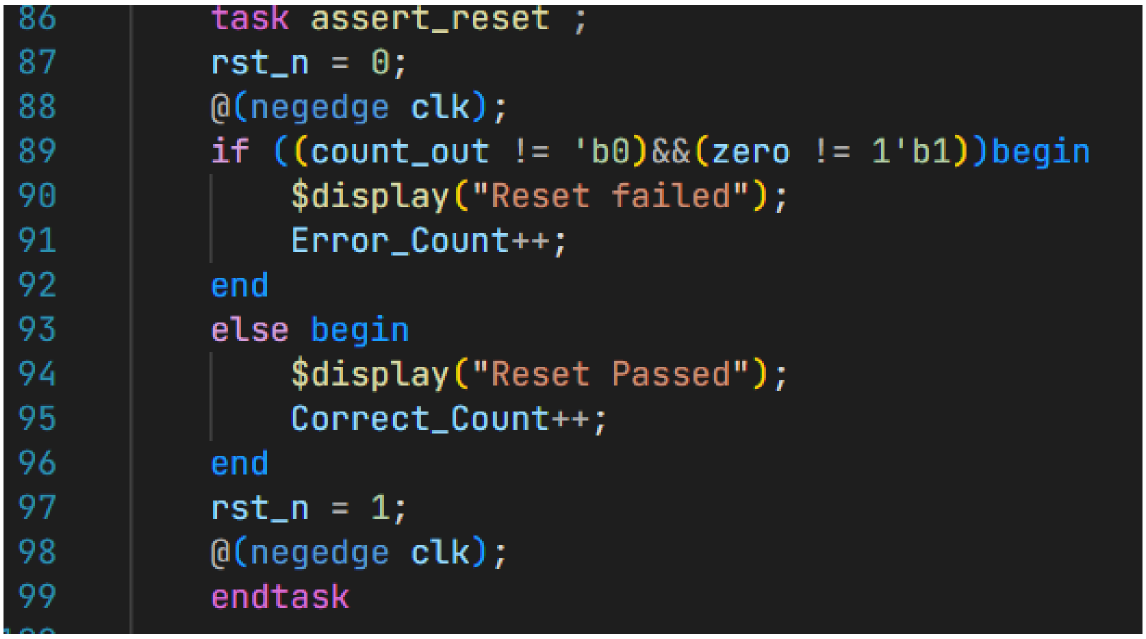
```systemverilog
module Q2();

    CounterConstraints constraintVals = new;
    parameter WIDTH = constraintVals.WIDTH;

    bit clk, rst_n, load_n, up_down, ce;
    bit [WIDTH-1:0] data_load;
    logic [WIDTH-1:0] count_out;
    logic max_count;
    logic zero;
    int Error_Count,Correct_Count;

    counter#(.WIDTH(WIDTH)) DUT (.*);

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



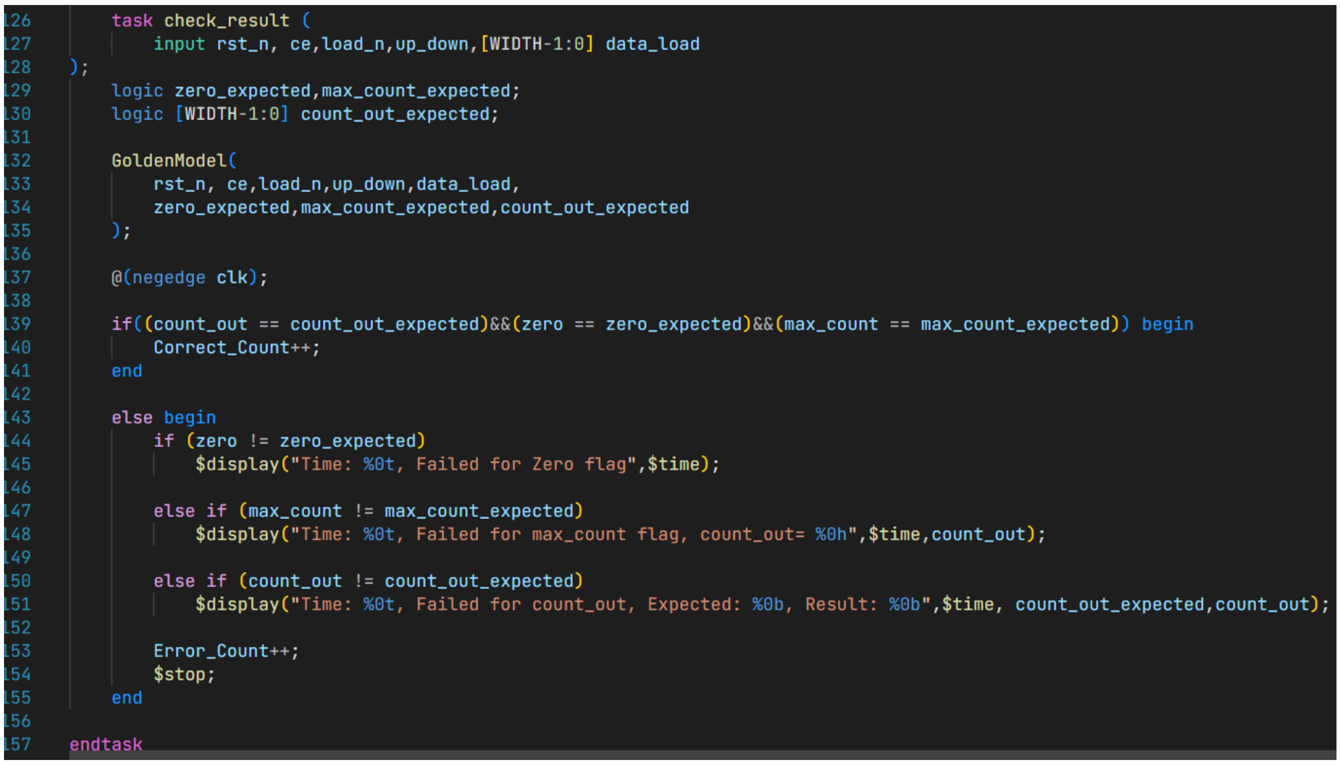
```systemverilog
    initial begin

        Error_Count = 0;
        Correct_Count = 0;

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

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

        repeat(50) begin
            assert (constraintVals.randomize());
            rst_n = constraintVals.rst_n;
            load_n = constraintVals.load_n;
            ce = constraintVals.ce;
            up_down = constraintVals.up_down;
            data_load = constraintVals.data_load;

            check_result(rst_n, ce,load_n,up_down,data_load);
        end

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



```verilog
task GoldenModel(
    input rst_n, ce,load_n,up_down,[WIDTH-1:0] data_load,
    output zero,max_count,[WIDTH-1 : 0]count_out
);
    if (rst_n == 0) begin
        max_count = 0;
        count_out = 'b0;
    end

    else if (load_n == 0) count_out = data_load;

    else if (ce) begin
        if(up_down == 1) count_out = count_out + 1;
        else count_out = count_out - 1;
    end

    if (count_out == 0) zero = 1;
    else zero = 0;

    if (count_out == {WIDTH{1'b1}}) max_count = 1;
    else max_count = 0;


endtask
```

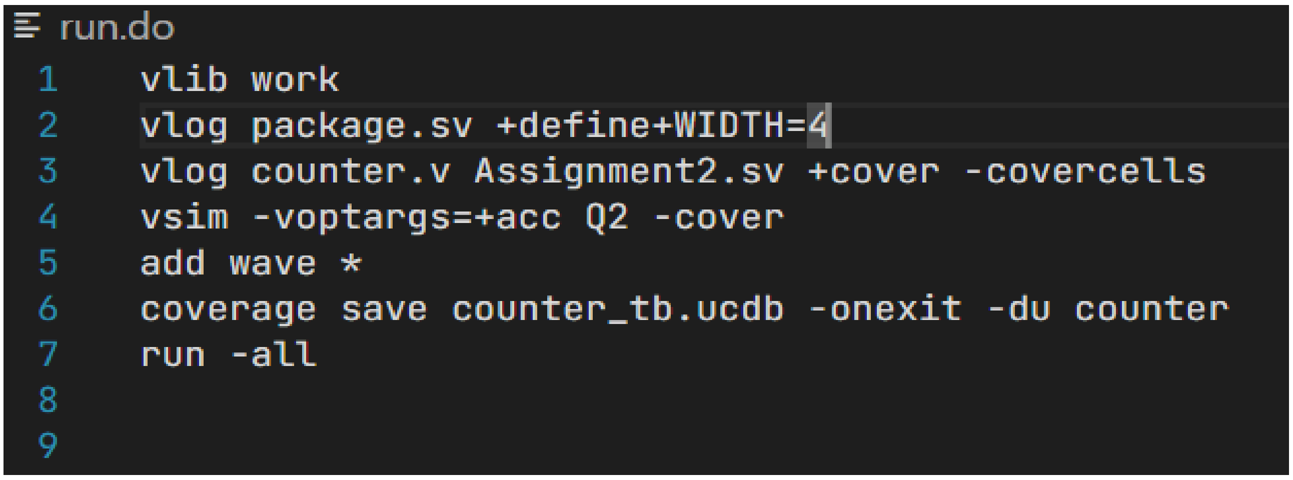


```verilog
task assert_reset ;
rst_n = 0;
@(negedge clk);
if ((count_out != 'b0)&&(zero != 1'b1))begin
    $display("Reset failed");
    Error_Count++;
end
else begin
    $display("Reset Passed");
    Correct_Count++;
end
rst_n = 1;
@(negedge clk);
endtask
```



```
    task check_result (
        input rst_n, ce,load_n,up_down,[WIDTH-1:0] data_load
);
    logic zero_expected,max_count_expected;
    logic [WIDTH-1:0] count_out_expected;

    GoldenModel(
        rst_n, ce,load_n,up_down,data_load,
        zero_expected,max_count_expected,count_out_expected
    );

    @(negedge clk);

    if((count_out == count_out_expected)&&(zero == zero_expected)&&(max_count == max_count_expected)) begin
        Correct_Count++;
    end

    else begin
        if (zero != zero_expected)
            $display("Time: %0t, Failed for Zero flag",$time);

        else if (max_count != max_count_expected)
            $display("Time: %0t, Failed for max_count flag, count_out= %0h",$time,count_out);

        else if (count_out != count_out_expected)
            $display("Time: %0t, Failed for count_out, Expected: %0b, Result: %0b",$time, count_out_expected,count_out);

        Error_Count++;
        $stop;
    end

endtask
```

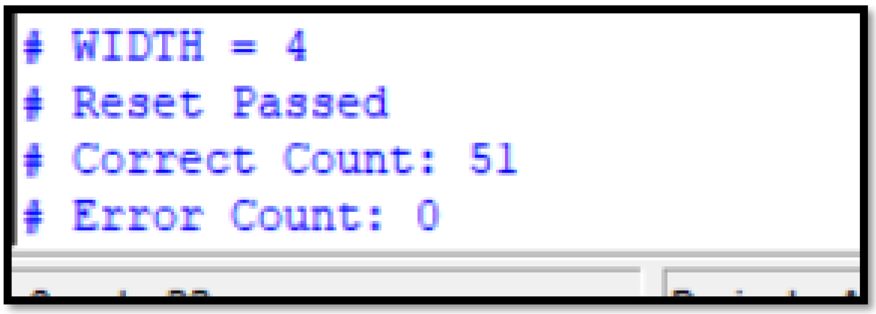
❖ <u>Do file:</u>



run.do

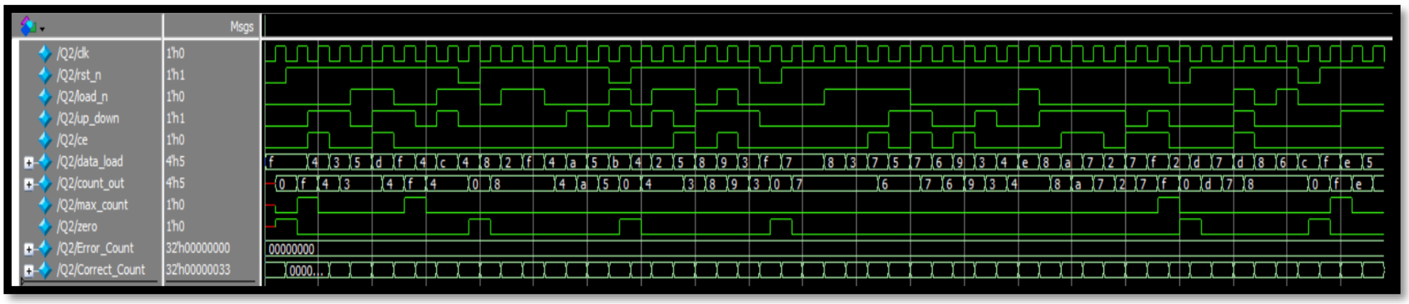
```
vlib work
vlog package.sv +define+WIDTH=4
vlog counter.v Assignment2.sv +cover -covercells
vsim -voptargs=+acc Q2 -cover
add wave *
coverage save counter_tb.ucdb -onexit -du counter
run -all
```

❖ **Result:**



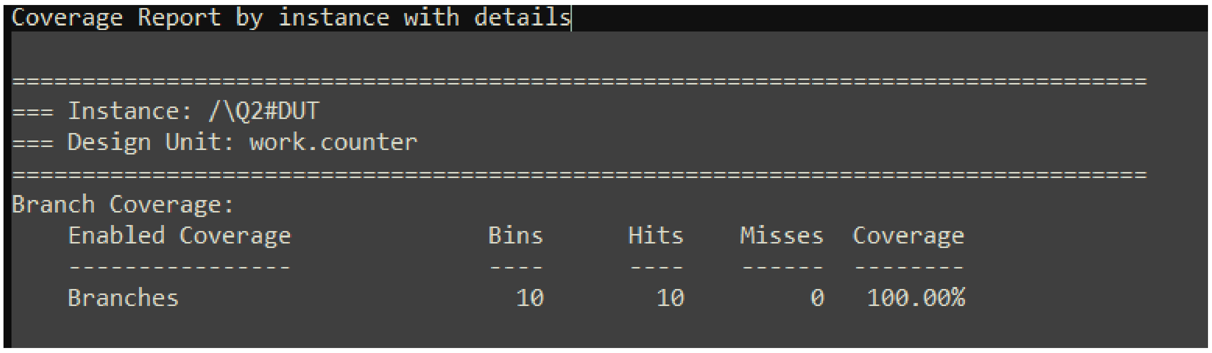
```
# WIDTH = 4
# Reset Passed
# Correct Count: 51
# Error Count: 0
```

Waveform:

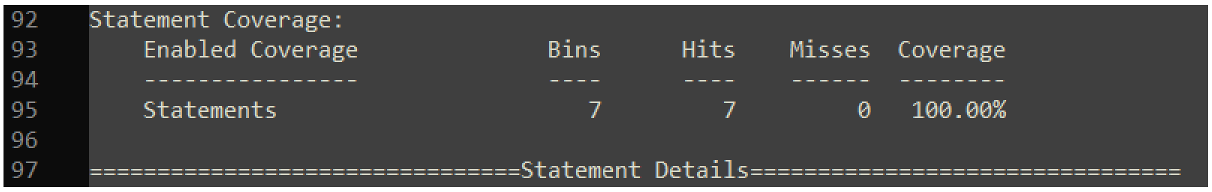


```
                        Msgs
/Q2/clk                 1'h0
/Q2/rst_n               1'h1
/Q2/load_n              1'h0
/Q2/up_down             1'h1
/Q2/ce                  1'h0
/Q2/data_load           4'h5
/Q2/count_out           4'h5
/Q2/max_count           1'h0
/Q2/zero                1'h0
/Q2/Error_Count         32'h00000000
/Q2/Correct_Count       32'h00000033
```

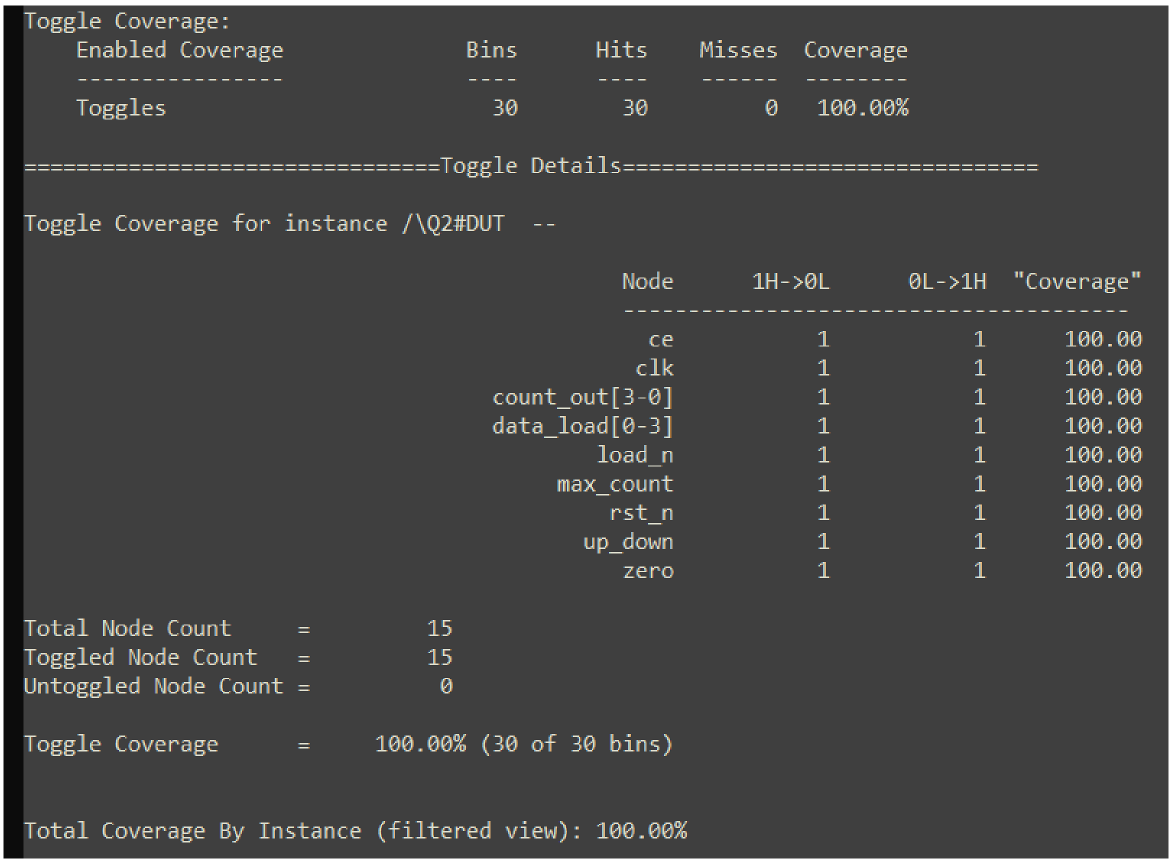
❖ **Coverage Report:**



```
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================================
```



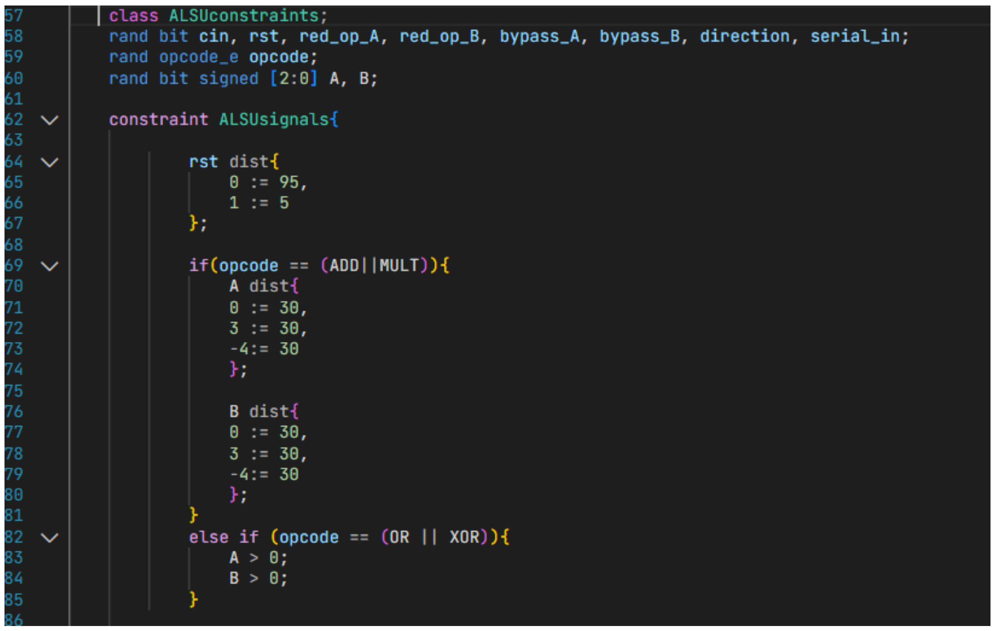
```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         30        30         0   100.00%

================================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%
```

Q3) ALSU

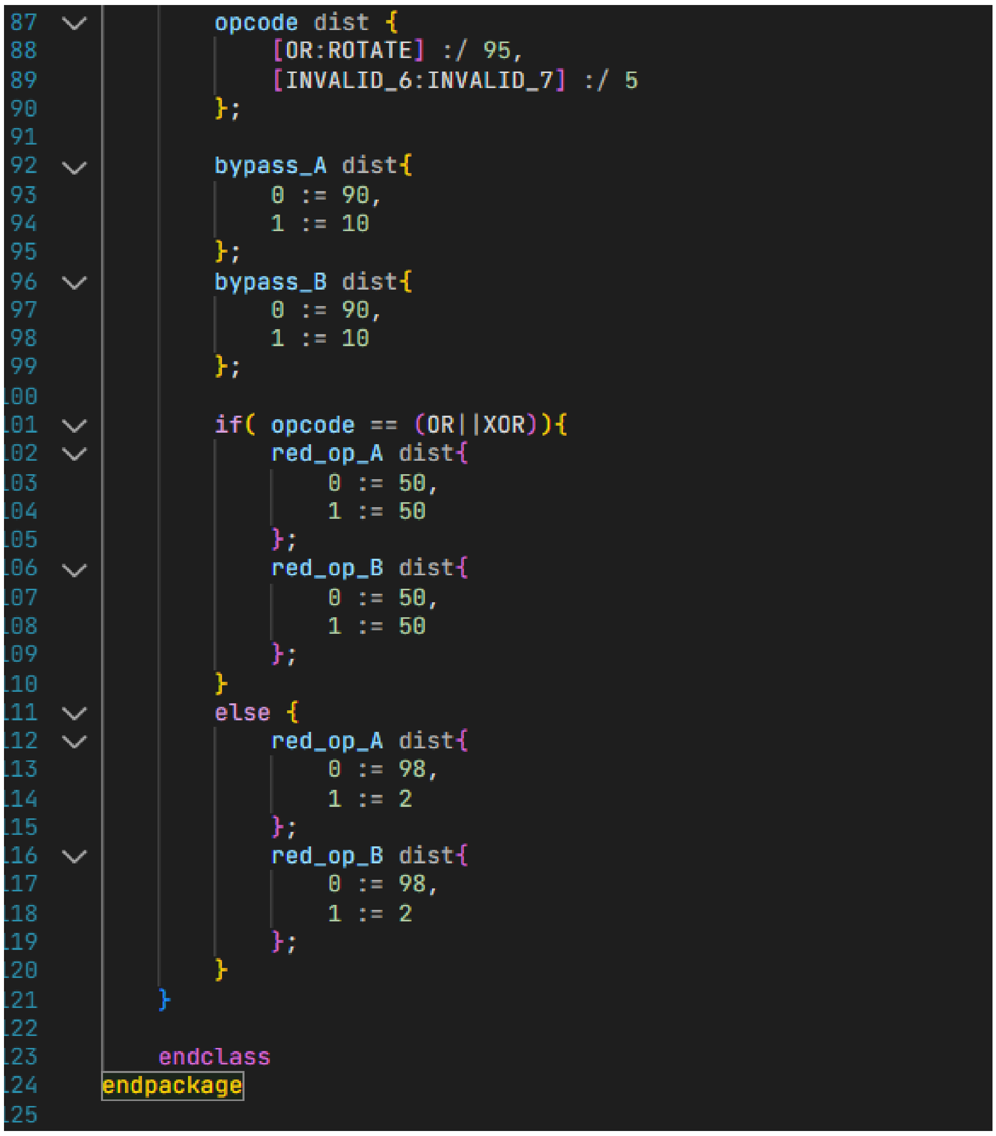
❖ Verification Plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU1	when reset is asserted, Outputs should be Low for at least one clk cycle.	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
ALSU2	In case opcode = ADD/MULT, & no invalid case, **LEDs** should = 0 &**out** should = A + B or A*B	Randomized under constraint of having A and B equals to(Max., Min., Zero) most of the time,	-	Comparing results to a refrence golden model
ALSU3	If **opcode = OR/XOR** and no invalid cases, if both red_op_A/B are low, **out = A OR/XOR B**, else if red_op_A is high, **out = redOR/XOR A**, else if red_op_A is low and red_op_b is high, **out = redOR/XOR B**	Randomized under constraint of having A/B having a high bit is reduction operation	-	Comparing results to a refrence golden model
ALSU4	verify Shift/ Rotate operations	Random during the simulation	-	Comparing results to a refrence golden model
ALSU5	Verifying correct behaviour if invalid operation happens, **LEDs** should blink and **out** should be low, if bypass → out = the bypassed signal with higher priority for A	Randomized under constraints of having less invalid operations, and low probabity of having a bypass signal high	-	Comparing results to a refrence golden model

❖ **Constraints class:**

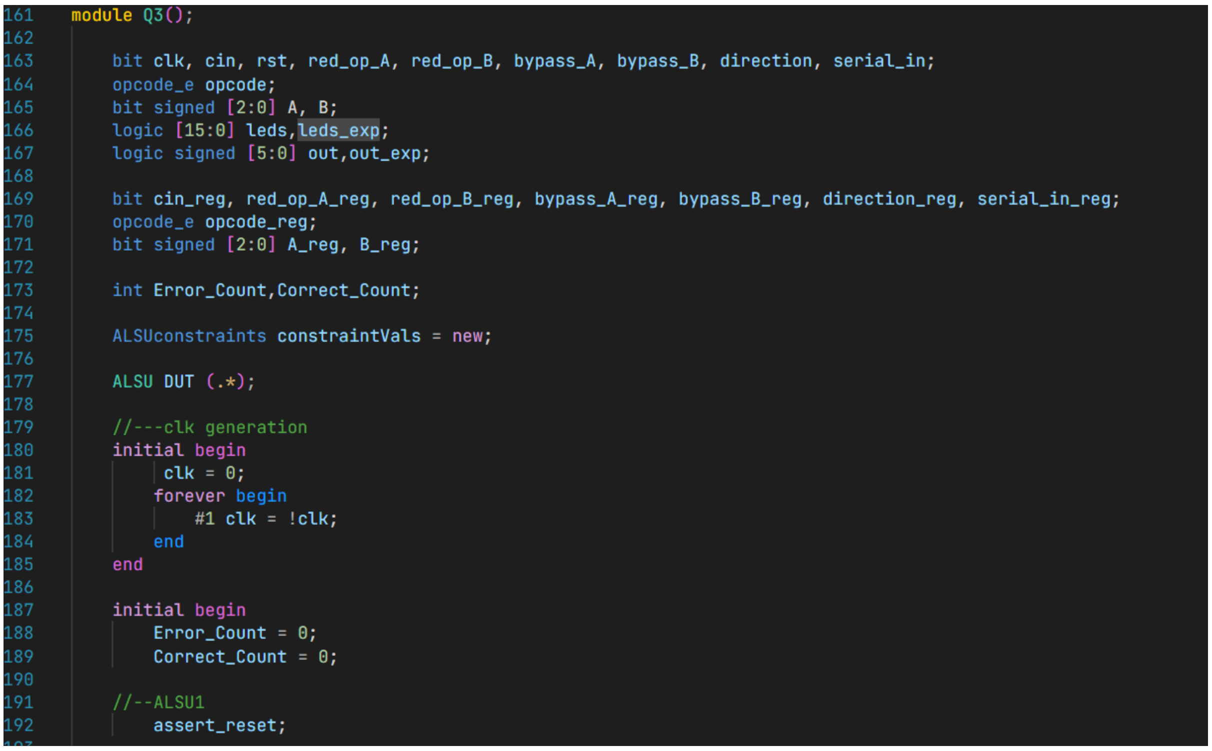


```
    class ALSUconstraints;
    rand bit cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    rand opcode_e opcode;
    rand bit signed [2:0] A, B;

    constraint ALSUsignals{

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

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

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

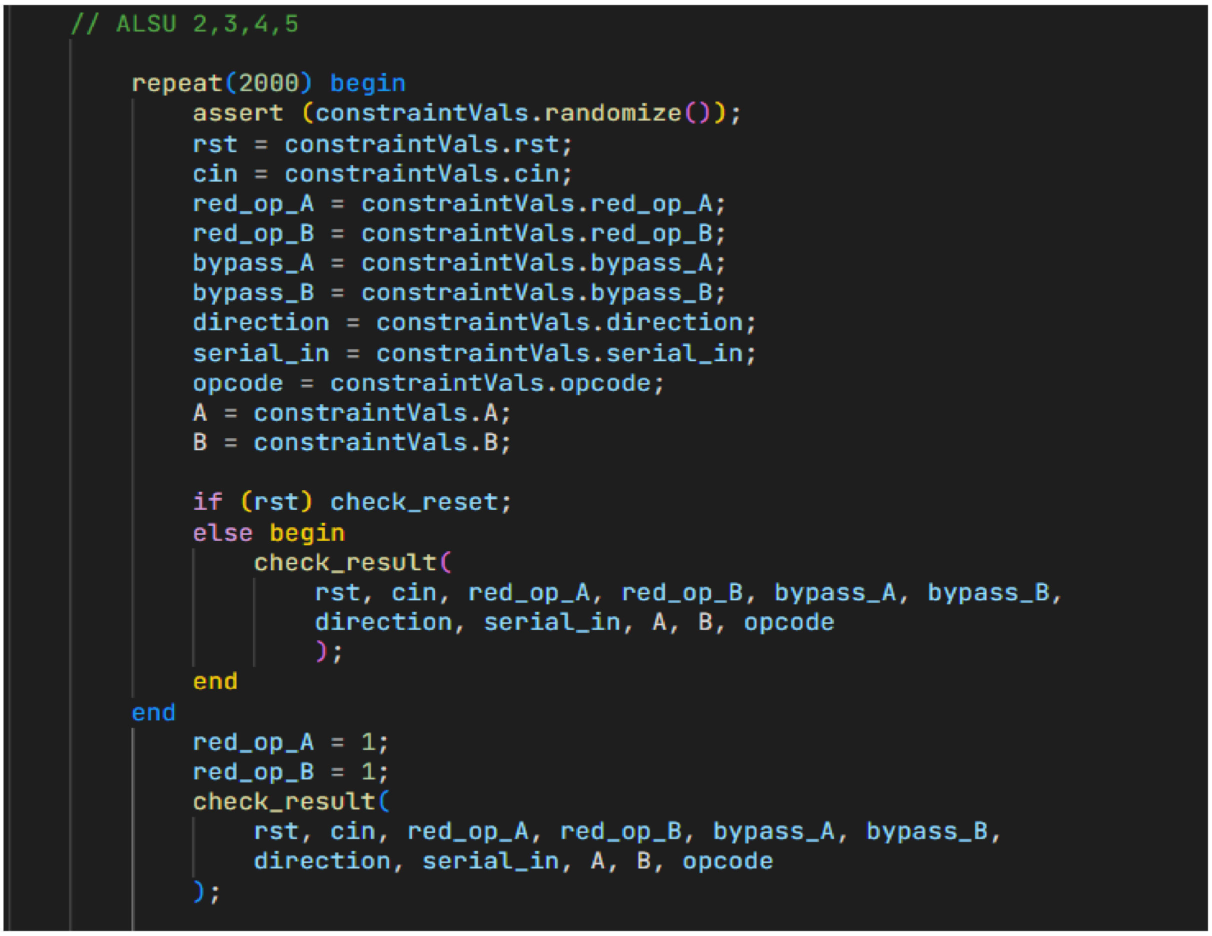


```
            opcode dist {
                [OR:ROTATE] :/ 95,
                [INVALID_6:INVALID_7] :/ 5
            };

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

            if( opcode == (OR||XOR)){
                red_op_A dist{
                    0 := 50,
                    1 := 50
                };
                red_op_B dist{
                    0 := 50,
                    1 := 50
                };
            }
            else {
                red_op_A dist{
                    0 := 98,
                    1 := 2
                };
                red_op_B dist{
                    0 := 98,
                    1 := 2
                };
            }
        }

        endclass
endpackage
```

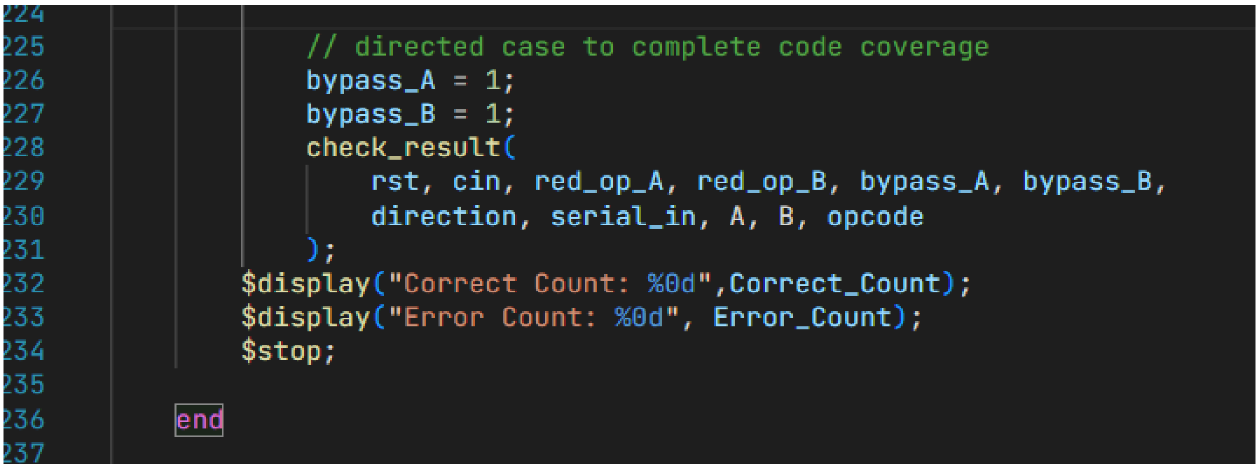
❖ TestBench:



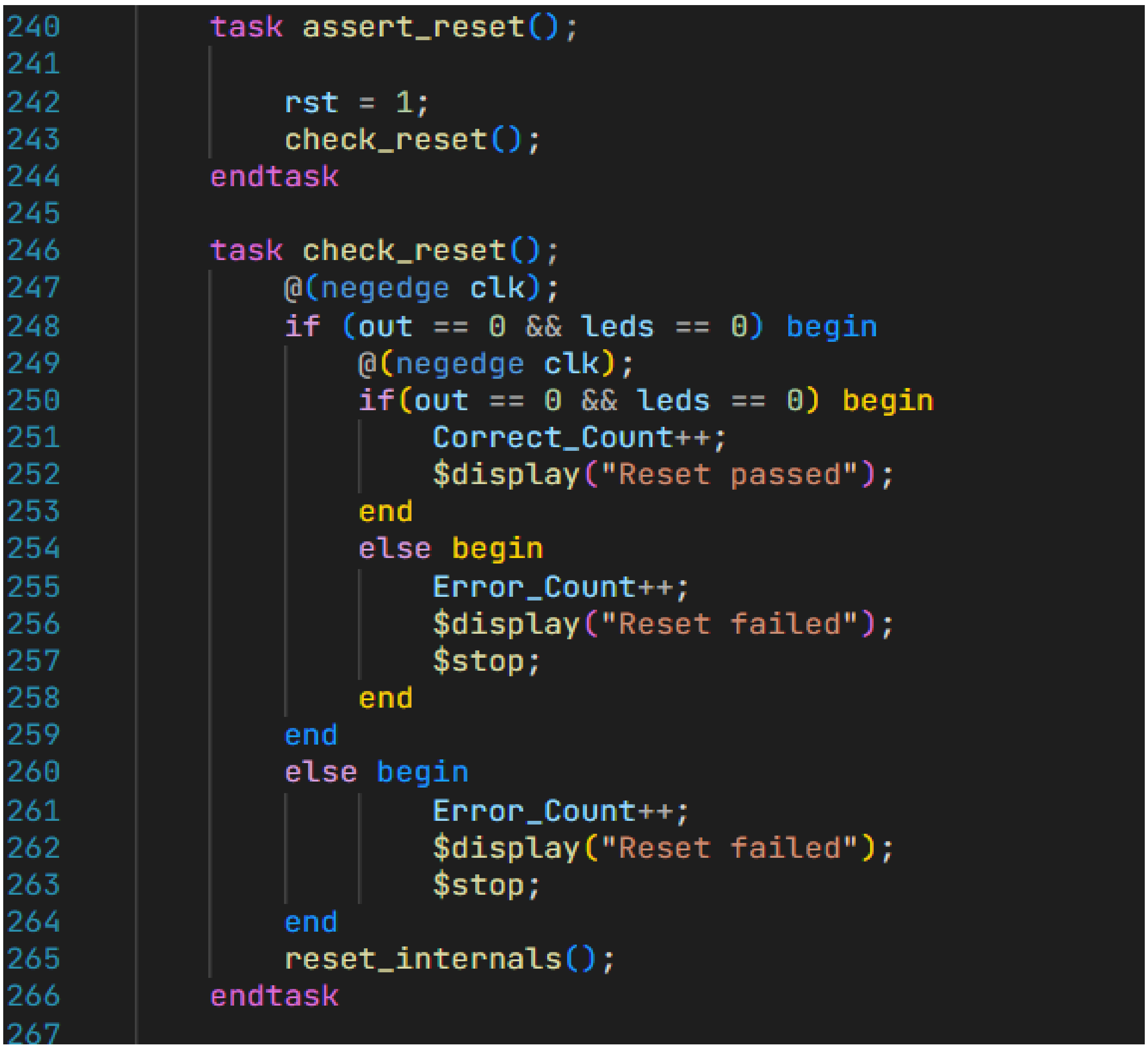
```
module Q3();

    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;

    int Error_Count,Correct_Count;

    ALSUconstraints constraintVals = new;

    ALSU DUT (.*);

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

    initial begin
        Error_Count = 0;
        Correct_Count = 0;

    //--ALSU1
        assert_reset;
```



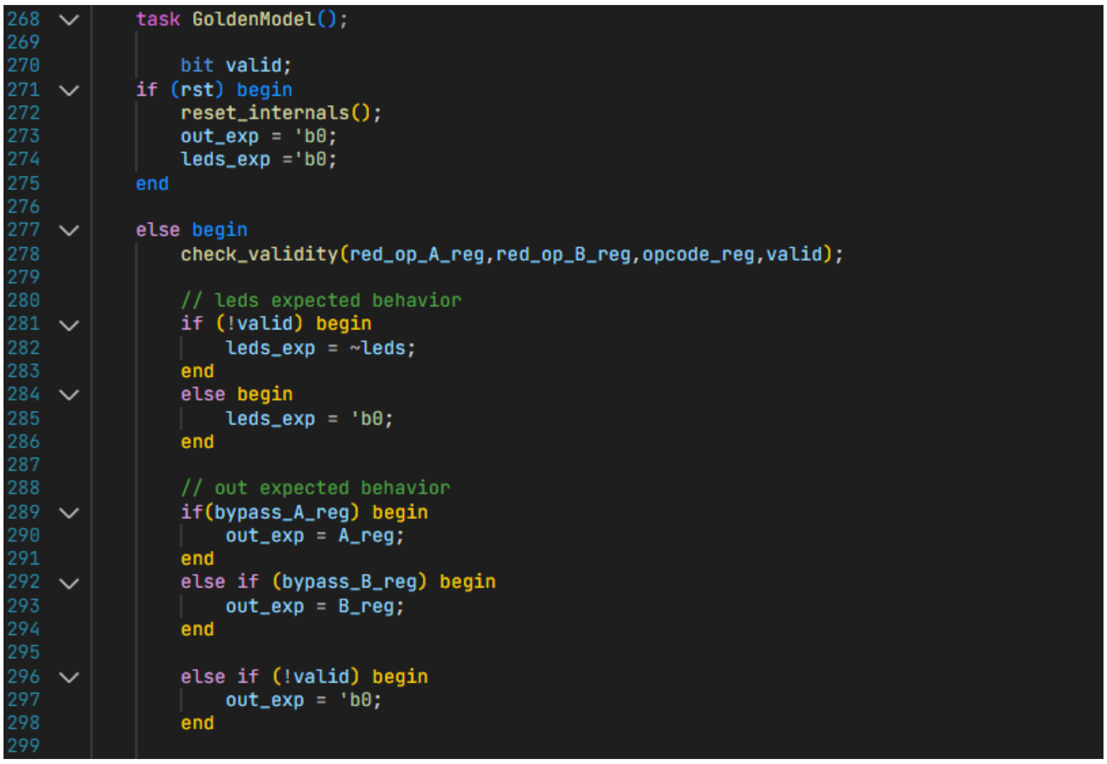
```
    // ALSU 2,3,4,5

        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(
                    rst, cin, red_op_A, red_op_B, bypass_A, bypass_B,
                    direction, serial_in, A, B, opcode
                    );
            end
        end
            red_op_A = 1;
            red_op_B = 1;
            check_result(
                rst, cin, red_op_A, red_op_B, bypass_A, bypass_B,
                direction, serial_in, A, B, opcode
            );
```



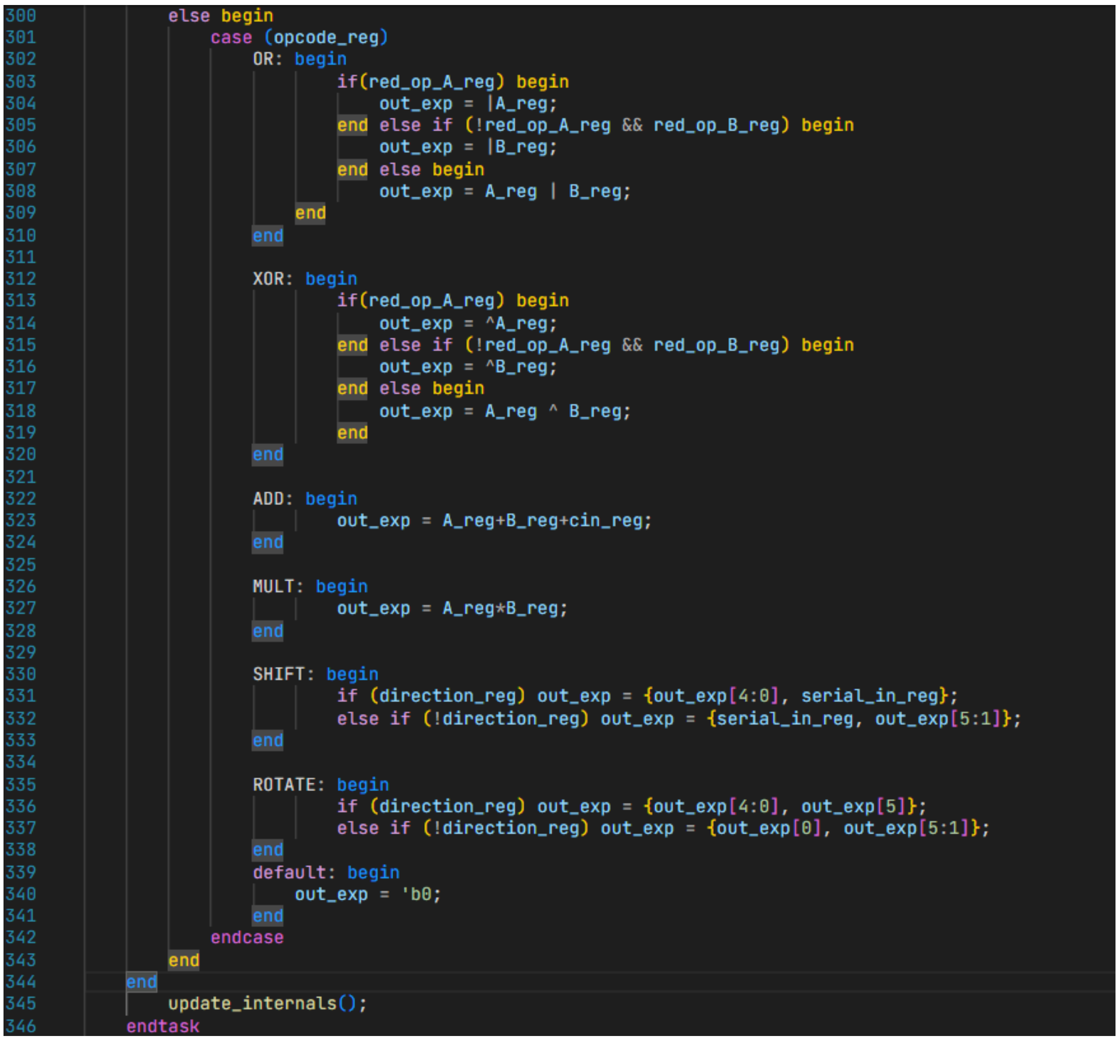
```verilog
            // directed case to complete code coverage
            bypass_A = 1;
            bypass_B = 1;
            check_result(
                rst, cin, red_op_A, red_op_B, bypass_A, bypass_B,
                direction, serial_in, A, B, opcode
            );
        $display("Correct Count: %0d",Correct_Count);
        $display("Error Count: %0d", Error_Count);
        $stop;

    end

```



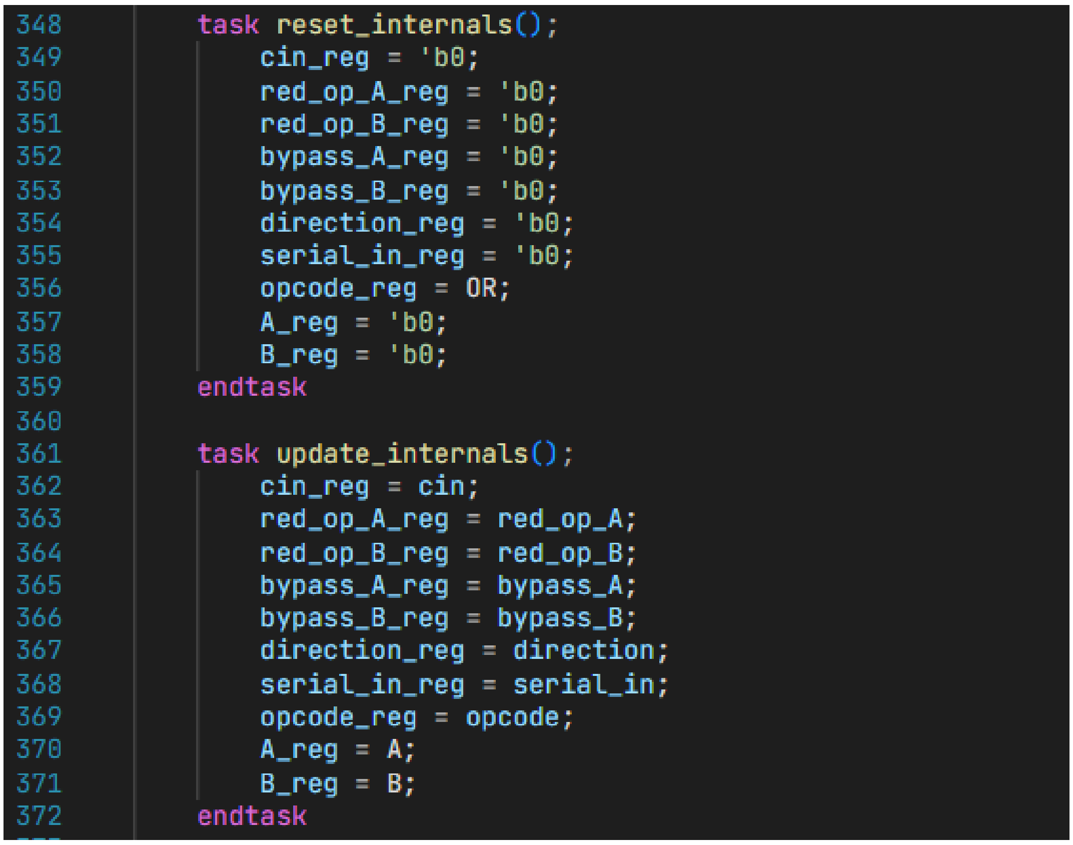
```verilog
    task assert_reset();

        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

```



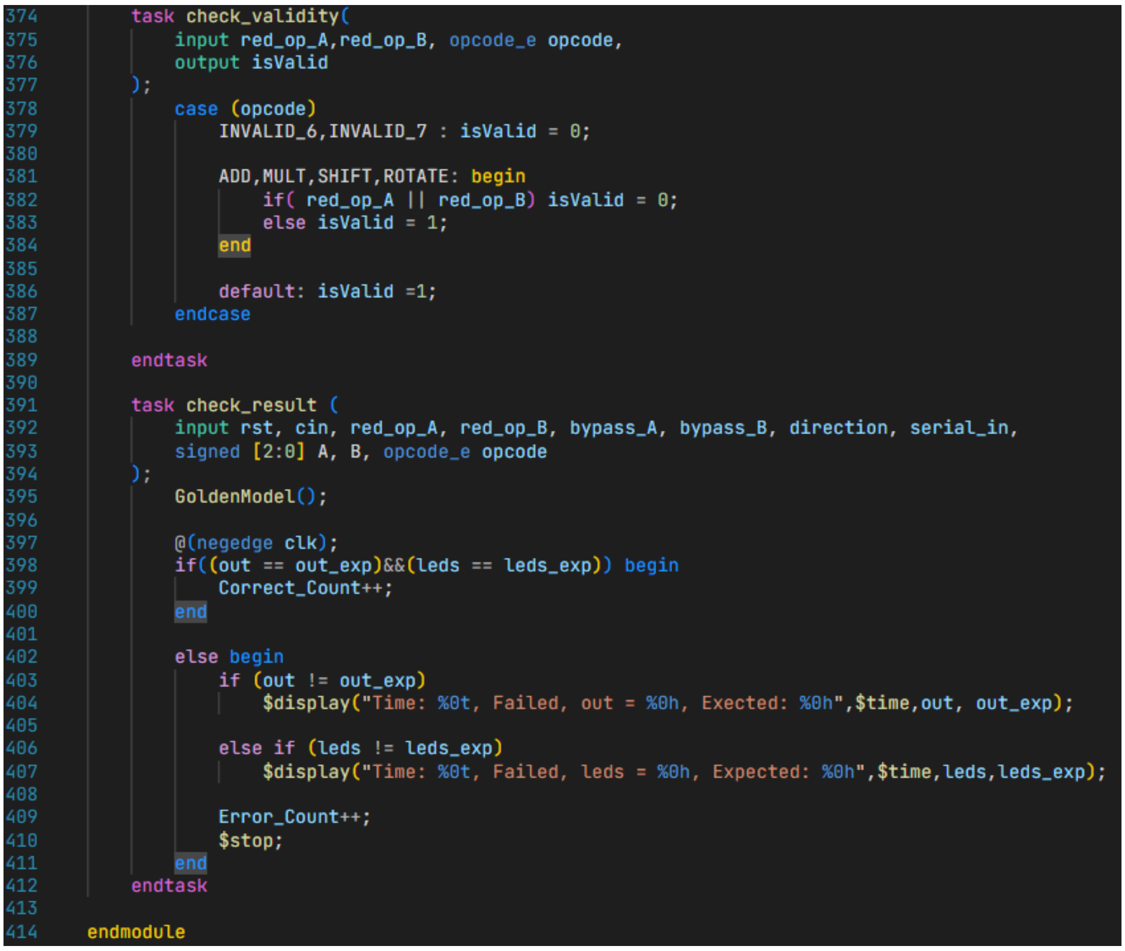
```
task GoldenModel();

    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
```

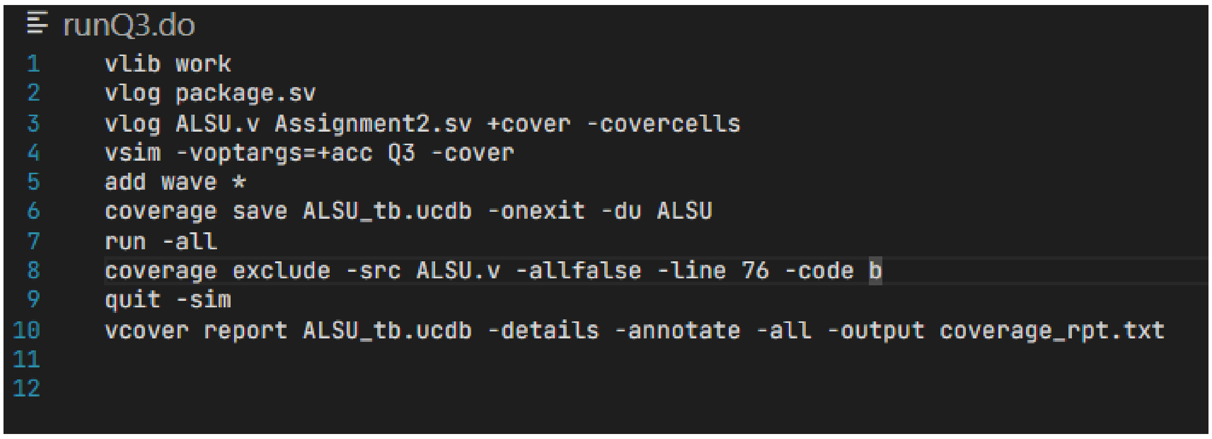


```
    task reset_internals();
        cin_reg = 'b0;
        red_op_A_reg = 'b0;
        red_op_B_reg = 'b0;
        bypass_A_reg = 'b0;
        bypass_B_reg = 'b0;
        direction_reg = 'b0;
        serial_in_reg = 'b0;
        opcode_reg = OR;
        A_reg = 'b0;
        B_reg = 'b0;
    endtask

    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
```

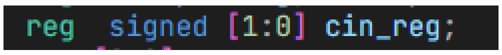
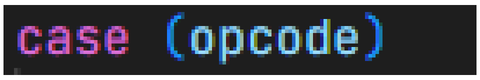
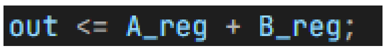
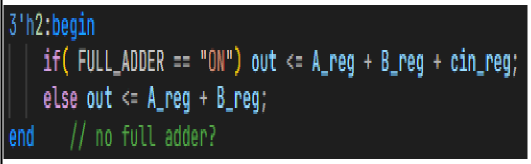
❖ Do file:



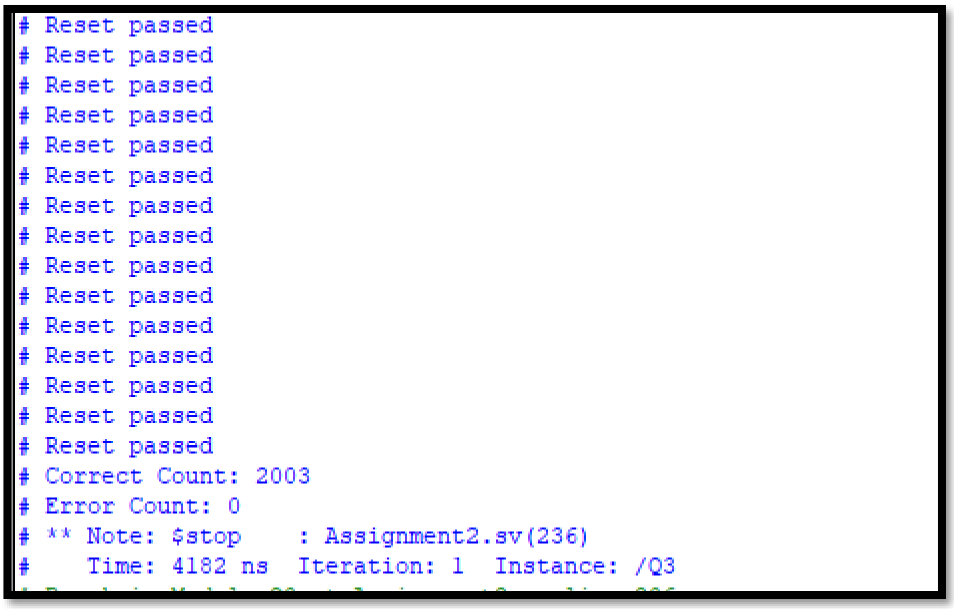
runQ3.do

```
vlib work
vlog package.sv
vlog ALSU.v Assignment2.sv +cover -covercells
vsim -voptargs=+acc Q3 -cover
add wave *
coverage save ALSU_tb.ucdb -onexit -du ALSU
run -all
coverage exclude -src ALSU.v -allfalse -line 76 -code b
quit -sim
vcover report ALSU_tb.ucdb -details -annotate -all -output coverage_rpt.txt
```

❖ Bugs found:

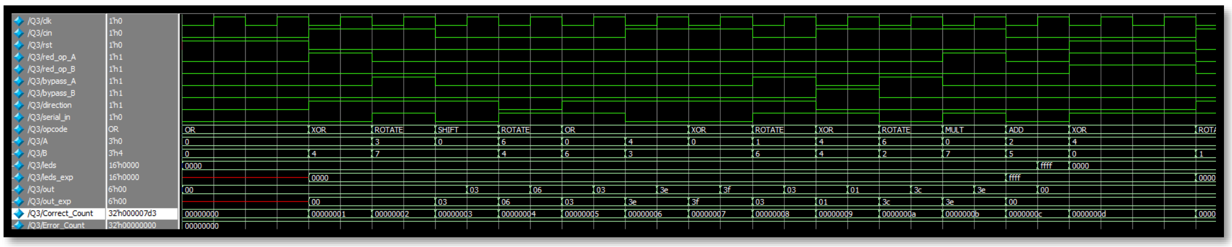
Bug Description	Wrong code	Correction
Internal wire (cin_reg) was defined to be signed, which causes sign extension in the addition processes.	 `reg signed [1:0] cin_reg;`	Cin_reg should be unsigned, and no need to have a width more than one bit
The case statement should use the value of opcode_reg	 `case (opcode)`	Opcode → opcode_reg
Missing FULL_ADDER parameter check when opcode = add, hence, cin wasn't taken into consideration	 `out <= A_reg + B_reg;`	 `3'h2:begin` `if( FULL_ADDER == "ON") out <= A_reg + B_reg + cin_reg;` `else out <= A_reg + B_reg;` `end // no full adder?`

❖ Results:

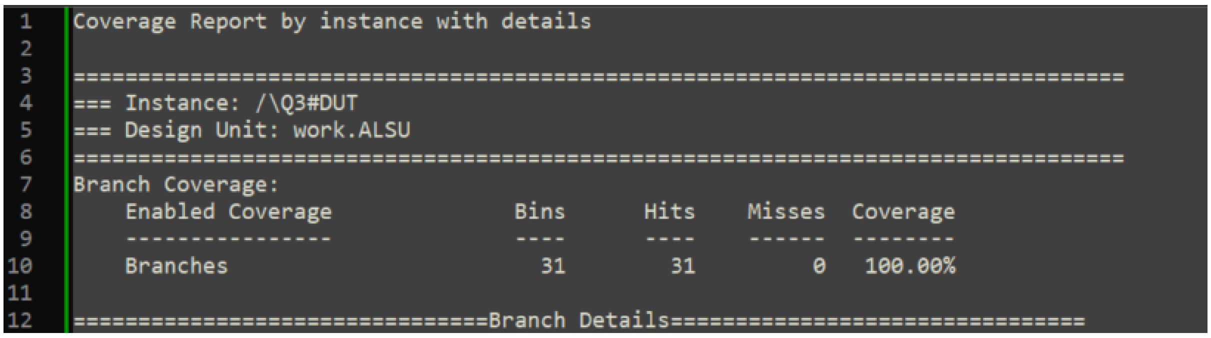


```
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Reset passed
# Correct Count: 2003
# Error Count: 0
# ** Note: $stop    : Assignment2.sv(236)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

Waveform:

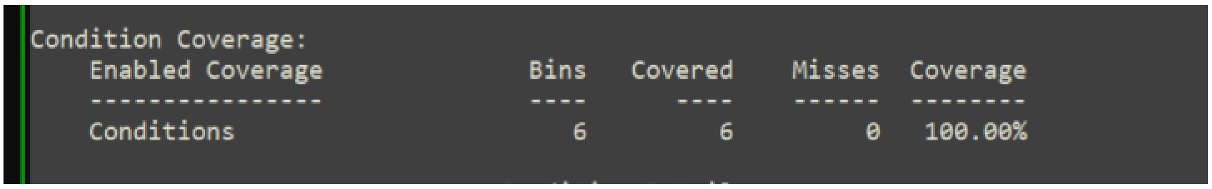


❖ Coverage Report:

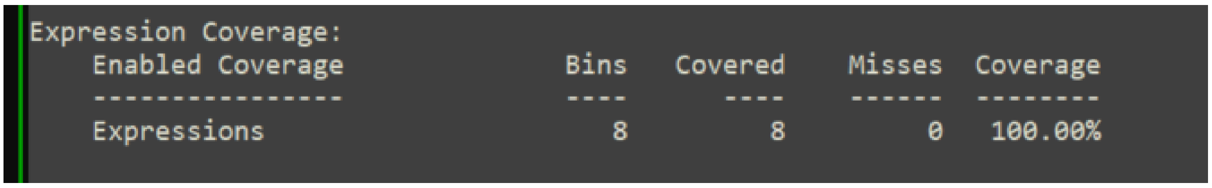


```
Coverage Report by instance with details

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

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

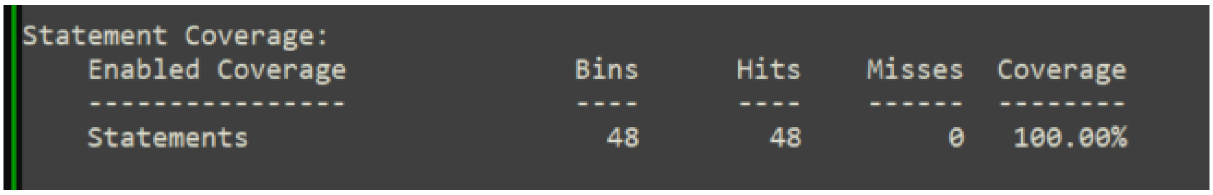
- Excluded **all False case** in the case statement, since invalid opcodes are handled by the flag (**invalid_opcode**)



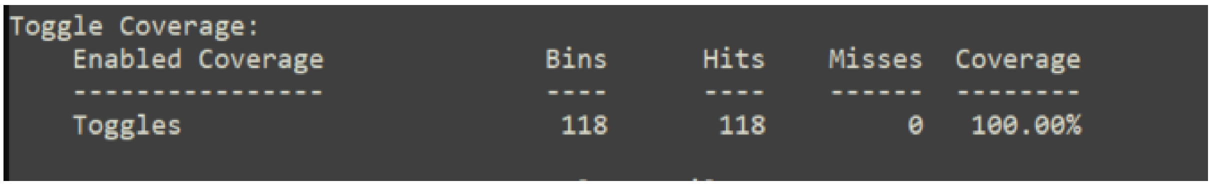
```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                       6         6         0   100.00%
```



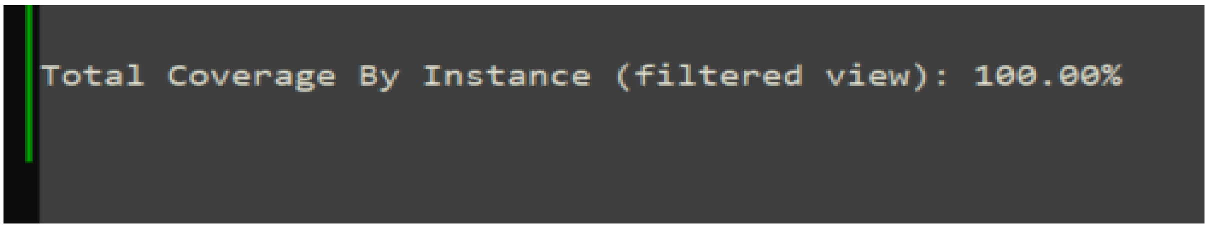
```
Expression Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Expressions                      8         8         0   100.00%
```



```
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%
```



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