ASSIGNMENT 4

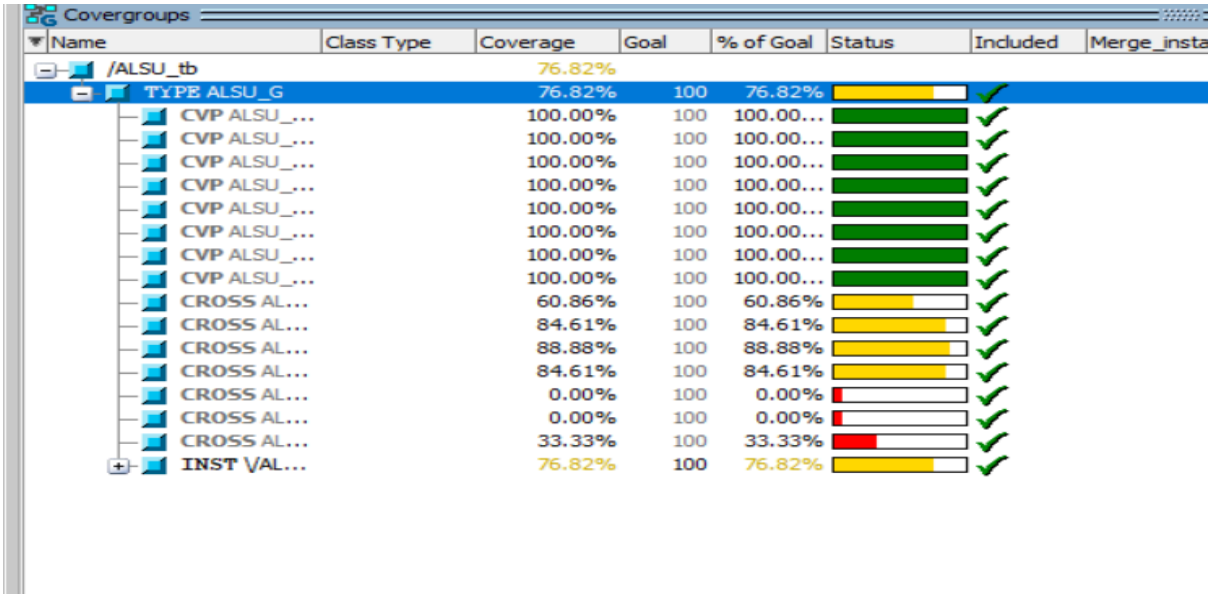
Q1

CODE TB :

```
22      covergroup ALSU_G @(posedge clk);
23
24      label_A : coverpoint A {
25      bins A_data_0 = {0};
26      bins A_data_max = {MAXPOS};
27      bins A_data_min = {MAXNEG};
28      bins A_data_default = default;}
29
30      label_A_red_op : coverpoint A iff (red_op_A) {
31      bins A_data_walking_ones[] = {1,2,4};}
32
33      label_B : coverpoint B {
34      bins B_data_0 = {0};
35      bins B_data_max = {MAXPOS};
36      bins B_data_min = {MAXNEG};
37      bins B_data_default = default;}
38
39      label_B_red_op : coverpoint B iff (red_op_B) {
40      bins B_data_walking_ones[] = {1,2,4};}
41
42      label_opcode : coverpoint opcode {
43      bins bin_shift[] = {SHIFT,ROTATE};
44      bins bin_arith[] = {ADD,MULT};
45      bins bin_bitwise[] = {OR,XOR};
46      //illegal_bins bin_invalid[] = {invalid6,invalid7};
47      bins opcode_tr = (0 => 1 => 2 => 3 => 4 => 5);}
48
49      label_cin : coverpoint cin;
50
51      label_direction : coverpoint direction;
52
53      label_serialin : coverpoint serial_in;
54
55      cross_add_mult : cross  label_A , label_B , label_opcode
56                      { bins add_mult_A_B = binsof (label_opcode) intersect {ADD,MULT} &&
57                                            binsof (label_A) intersect {MAXPOS,MAXNEG,ZERO} &&
58                                            binsof (label_B) intersect {MAXPOS,MAXNEG,ZERO}; }
59
60      cross_add_cin : cross label_opcode , label_cin
61                      { bins add_cin = binsof (label_opcode) intersect {ADD} &&
62                                       binsof (label_cin) intersect {0,1}; }
63
```

```
63
64      cross_shift_dir : cross label_opcode , label_direction
65                        { bins shift_dir = binsof (label_opcode) intersect {SHIFT,ROTATE} &&
66                                           binsof (label_direction) intersect {0,1}; }
67
68      cross_shift_serial : cross label_opcode , label_serialin
69                           { bins shift_serial = binsof (label_opcode) intersect {SHIFT} &&
70                                                 binsof (label_serialin) intersect {0,1}; }
71
72      cross_or_redop_A : cross  label_A , label_B , label_opcode , label_A_red_op
73                         { bins or_redop_A = binsof (label_opcode) intersect {OR,XOR} &&
74                                             binsof (label_A_red_op) intersect {1} &&
75                                             binsof (label_A) intersect {1,2} &&
76                                             binsof (label_B) intersect {0}; }
77
78      cross_or_redop_B : cross  label_A , label_B , label_opcode , label_B_red_op
79                         { bins or_redop_B = binsof (label_opcode) intersect {OR,XOR} &&
80                                             binsof (label_B_red_op) intersect {1} &&
81                                             binsof (label_A) intersect {0} &&
82                                             binsof (label_B) intersect {1,2}; }
83
84      cross_invalid_case : cross label_opcode , label_A_red_op , label_B_red_op
85                           { bins invalid_case = binsof (label_opcode) intersect {invalid6,invalid7} &&
86                                                 binsof (label_A_red_op) intersect {1} ||
87                                                 binsof (label_B_red_op) intersect {1}; }
88
89      endgroup
90
91      ALSU_G my_alu_g;
92
93      initial begin
94        my_alu_g = new;
95        end
96
97       always @(posedge clk) begin
98             if (!rst && !bypass_A && !bypass_B) begin
99             my_alu_g.sample();
100       end
101       end
```

DISPLAY :



Q2

CODE :

assert.sv

```
1   module assert ();
2
3       property test1;
4       (@(posedge clk) a |-> ##2 b);
5       endproperty
6       assert property (test1);
7       cover property (test1);
8
9       property test2;
10      (@(posedge clk) (a&&b) |=>  ##[1:3] c);
11      endproperty
12      assert property (test2);
13      cover property (test2);
14
15      sequence s11b;
16          ##2 !b;
17      endsequence
18
19      property test3;
20      (@(posedge clk) ($onehot(Y) == 1));
21      endproperty
22      assert property (test3);
23      cover property (test3);
24
25      property test4;
26      (@(posedge clk) (d == 4'b0000) |=> ##1 !valid);
27      endproperty
28      assert property (test4);
29      cover property (test4);
30
31  endmodule
```

Q3

CODE TB :

counter_tb.sv

```
1   import counter_pkg ::*;
2   module counter_tb (counter_if.TEST count_in);
3
4       //counter g (count_in);
5       //bind counter counter_sva sva (.*);
6       counter_class counter1 = new;
7
8       always @(*) begin
9           counter1.clk = count_in.clk;
10      end
11
12      initial begin
13
14          count_in.rst_n = 0;
15          @(negedge count_in.clk);
16          count_in.rst_n = 1;
17
18
19      repeat (9999) begin
20          assert (counter1.randomize());
21          count_in.rst_n = counter1.reset;
22          count_in.load_n = counter1.load_n;
23          count_in.ce = counter1.ce;
24          count_in.data_load = counter1.data_load;
25          count_in.up_down = counter1.up_down;
26          counter1.count_out = count_in.count_out;
27          @(negedge count_in.clk);
28
29      end
30      $stop;
31      end
32
33  endmodule
```

CODE PKG :

counter_pkg.sv

```
1   package counter_pkg;
2
3     parameter WIDTH = 4;
4
5   class counter_class ;
6
7     parameter MAX = {WIDTH{1'b1}};
8     parameter MIN = {WIDTH{1'b0}};
9     rand logic reset;
10    rand logic load_n;
11    rand logic ce;
12    rand logic [WIDTH-1:0] data_load;
13    rand logic up_down;
14    bit clk;
15    logic [WIDTH-1:0] count_out;
16
17
18    constraint counter_cst {
19        reset dist {0 := 1, 1:= 99};
20        load_n  dist {0 := 30, 1:= 70};
21        ce dist {0 := 30, 1:= 70};
22    };
23
24    covergroup counter_g @(posedge clk);
25        label1 : coverpoint data_load iff (!load_n && reset);
26        label2 : coverpoint count_out iff (reset && ce && up_down);
27        label3 : coverpoint count_out iff (reset && ce && up_down) {
28          bins overflow = (MAX => MIN);
29        }
30        label4 : coverpoint count_out iff (reset && ce && !up_down);
31        label5 : coverpoint count_out iff (reset && ce && !up_down) {
32          bins overflow = (MIN => MAX);
33        }
34    endgroup
35
36    function new();
37        counter_g = new();
38    endfunction
39
40  endclass
41
42  endpackage
43
```

CODE SVA :

counter_sva.sv

```
import counter_pkg ::*;
module counter_sva (counter_if.DUT count_in);

    always_comb begin
     if(!count_in.rst_n)
     a_reset: assert final(count_in.count_out == 0);
end

    property load_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (!count_in.load_n) |=> (count_in.count_out == $past(count_in.data_load));
    endproperty
    assert property (load_check) else $error(" error in load ");
    cover property (load_check);

    property notchange_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (count_in.load_n && !count_in.ce) |=> (count_in.count_out == $past(count_in.count_out));
    endproperty
    assert property (notchange_check) else $error(" error in not change ");
    cover property (notchange_check);

    property increment_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (count_in.load_n && count_in.ce && count_in.up_down) |=> (count_in.count_out == $past(count_in.count_out) + 1'b1);
    endproperty
    assert property (increment_check) else $error(" error in increment ");
    cover property (increment_check);

    property decrement_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (count_in.load_n && count_in.ce && !count_in.up_down) |=> (count_in.count_out == $past(count_in.count_out) - 1'b1);
    endproperty
    assert property (decrement_check) else $error(" error in decrement ");
    cover property (decrement_check);

    property max_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (count_in.count_out == {WIDTH{1'b1}}) |-> (count_in.max_count);
    endproperty
    assert property (max_check) else $error(" error in max ");
    cover property (max_check);

    property zero_check;
        @(posedge count_in.clk) disable iff (!count_in.rst_n)
            (count_in.count_out == {WIDTH{1'b0}}) |-> (count_in.zero);
    endproperty
    assert property (zero_check) else $error(" error in zero ");
    cover property (zero_check);

endmodule
```

CODE INTERFACE :

counter_interface.sv

```
interface counter_if (clk);
    parameter WIDTH = 4;
     input bit clk;
     logic rst_n;
     logic load_n;
     logic up_down;
     logic ce;
     logic [WIDTH-1:0] data_load;
     logic [WIDTH-1:0] count_out;
     logic max_count;
     logic zero;

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

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

```

counter_top.sv

```
module counter_top ();
    bit clk;
    always #1 clk = ~clk;

    counter_if count_in (clk);
    counter dut (count_in);
    counter_tb tb (count_in);
    bind counter counter_sva sv (count_in);


endmodule
```

counter.do

```
1  vlib work
2  vlog counter.sv counter_pkg.sv counter_sva.sv counter_tb.sv counter_interface.sv counter_top.sv +cover -covercells
3  vsim -voptargs=+acc work.counter_top -cover
4  add wave *
5  coverage save counter_tb.ucdb -onexit
6  run -all
```

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/counter_top/dut/s...	SVA	✓	Off	797	1	Unli...	1	100%		✓	0	0	0 ns	0
/counter_top/dut/s...	SVA	✓	Off	595	1	Unli...	1	100%		✓	0	0	0 ns	0
/counter_top/dut/s...	SVA	✓	Off	2364	1	Unli...	1	100%		✓	0	0	0 ns	0
/counter_top/dut/s...	SVA	✓	Off	2409	1	Unli...	1	100%		✓	0	0	0 ns	0
/counter_top/dut/s...	SVA	✓	Off	2096	1	Unli...	1	100%		✓	0	0	0 ns	0
/counter_top/dut/s...	SVA	✓	Off	2908	1	Unli...	1	100%		✓	0	0	0 ns	0

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_inst
/counter_pkg/coun...		100.00%					
TYPE counter_...		100.00%	100	100.00...		✓	
CVP count...		100.00%	100	100.00...		✓	
CVP count...		100.00%	100	100.00...		✓	
CVP count...		100.00%	100	100.00...		✓	
CVP count...		100.00%	100	100.00...		✓	
CVP count...		100.00%	100	100.00...		✓	
INST \/cou...		100.00%	100	100.00...		✓	
CVP lab...		100.00%	100	100.00...		✓	
CVP lab...		100.00%	100	100.00...		✓	
CVP lab...		100.00%	100	100.00...		✓	
CVP lab...		100.00%	100	100.00...		✓	
CVP lab...		100.00%	100	100.00...		✓	

Q4

1- Error resetting analog_test register o Asserting reset: Expected 16’hABCD and actual 16’hABCC.
2- tmp_sensor0 register is shifted left by 1bit -> bit_0 stuck at 0 o Many different combinations and the output is shifted by 1_bit.
3- digita_config register bit_15 is stuck at 0 o Write 16’hFFFF and observed 16’hFFFE.
4- adc0_reg bit_15 is stuck at 1 o Write 16’h0000 and observed 16’h8000.
5- adc1_reg is rotated by 1 byte o Write 16’h0001 and observed 16’h0100. o Write 16’h1000 and observed 16’h0001.
6- digital_test register output is mapped to the address of the register amp_gain o Write 16’h5 to digital_test address and observed from amp_gain register.
7- amp_gain register output is mapped to the address of the register digital_test o Write 16’h5 to amp_gain address and observed from digital_test register.
8- reset has no effect on tmp_sensor1 register o Value 16’h3 is stored and then reset is asserted and the value of the register didn`t change

CODE TB :

register_tb.sv

```
module register_tb ();
    logic clk;
    logic reset;
    logic write;
    logic [15:0] data_in;
    logic [2:0] address;
    logic [15:0] data_out;

    config_reg d (.*);

    typedef string str;
    typedef logic [15:0] reg_val;
    typedef reg_val assoc_t [str];

    assoc_t reset_assoc;

    typedef enum bit[15:0] {adc0_reg ,
                            adc1_reg ,
                            temp_sensor0_reg ,
                            temp_sensor1_reg ,
                            analog_test ,
                            digital_test ,
                            amp_gain ,
                            digital_config } reg_mame;

      reg_mame reg_addr;

      task automatic init_reset;
        reg_addr = reg_addr.first;

        reset_assoc[reg_addr] = 16'hffff;
        reg_addr = reg_addr.next;

        reset_assoc[reg_addr] = 16'h0000;
        reg_addr = reg_addr.next;

        reset_assoc[reg_addr] = 16'h0000;
        reg_addr = reg_addr.next;

        reset_assoc[reg_addr] = 16'h0000;
        reg_addr = reg_addr.next;

        reset_assoc[reg_addr] = 16'hABCD;
        reg_addr = reg_addr.next;

        reset_assoc[reg_addr] = 16'h0000;
        reg_addr = reg_addr.next;

```

```
48
49              reset_assoc[reg_addr] = 16'h0000;
50              reg_addr = reg_addr.next;
51
52              reset_assoc[reg_addr] = 16'h1;
53              reg_addr = reg_addr.next;
54
55          endtask
56
57          task do_reset;
58              @(negedge clk);
59              reset = 1;
60              @(negedge clk);
61              reset = 0;
62  endtask
63
64  task cheack_result (input assoc_t data_out_ex);
65  @(negedge clk)
66      if (data_out_ex !== data_out) begin
67      $display("Expected: %h, Got: %h", data_out_ex,data_out);
68    end else begin
69      $display("PASS");
70    end
71
72  endtask
73
74              initial begin
75          clk = 0;
76          forever begin
77              #1 clk = ~clk;
78          end
79    end
80
81    initial begin
82        init_reset;
83
84        reset = 0;
85        write = 0;
86        data_in = 0;
87        address = 0;
88
89        do_reset;
90        // reset
91        reg_addr = reg_addr.first;
```

```
91          reg_addr = reg_addr.first;
92          for (int i = 0 ; i < reg_addr.num ; i++) begin
93              address = reg_addr;
94              cheack_result (reset_assoc[reg_addr]);
95              reg_addr = reg_addr.next;
96          end
97          //zero
98          reg_addr = reg_addr.first;
99          for (int i = 0 ; i < reg_addr.num ; i++) begin
100             address = reg_addr;
101             write = 1;
102             data_in = 16'hffff;
103             @(posedge clk);
104             cheack_result (16'hffff);
105             reg_addr = reg_addr.next;
106         end
107         //one
108         reg_addr = reg_addr.first;
109         for (int i = 0 ; i < reg_addr.num ; i++) begin
110             address = reg_addr;
111             write = 1;
112             data_in = 16'h0000;
113             @(posedge clk);
114             cheack_result (16'h0000);
115             reg_addr = reg_addr.next;
116         end
117         //
118         reg_addr = reg_addr.first;
119         for (int i = 0 ; i < reg_addr.num ; i++) begin
120             address = reg_addr;
121             write = 1;
122             data_in = 16'h1;
123             for (int j ; j<16 ; j++) begin
124                 @(posedge clk);
125                 cheack_result(data_in);
126                 data_in = data_in*2;
127             end
128             reg_addr = reg_addr.next;
129         end
130
```

```
130
131         do_reset;
132
133         reg_addr = reg_addr.first;
134         for (int i = 0 ; i < reg_addr.num ; i++) begin
135             address = reg_addr;
136             cheack_result (reset_assoc[reg_addr]);
137             reg_addr = reg_addr.next;
138         end
139     end
140
141 endmodule
```