

Cross Coverage & SVA Assignment
Assignment4

By: Mohamed Sayed Mohamed Soliman

Question 1

➤ ALSU

Add the following cross coverage in the class of the ALSU of the last assignment:

1. When the ALU is addition or multiplication, A and B should have taken all permutations of maxpos, maxneg and zero.
2. When the ALU is addition, c_in should have taken 0 or 1
3. When the ALSU is shifting, then shift_in must take 0 or 1
4. When the ALSU is shifting or rotating, then direction must take 0 or 1
5. When the ALSU is OR or XOR and red_op_A is asserted, then A took all walking one patterns (001, 010, and 100) while B is taking the value 0
6. When the ALSU is OR or XOR and red_op_B is asserted, then B took all walking one patterns (001, 010, and 100) while A is taking the value 0
7. Covering the invalid case: reduction operation is activated while the opcode is not OR or XOR

1. Code Design

```
3  module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
4  parameter INPUT_PRIORITY = "A";
5  parameter FULL_ADDER = "ON";
6  input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
7  input [2:0] opcode;
8  input signed [2:0] A, B;    // first bug [ ]> we must put it signed
9  output reg [15:0] leds;
10 output reg signed [5:0] out; // second bug [ ]> we must put it signed
11
12 reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
13 reg [2:0] opcode_reg;
14 reg signed [2:0] A_reg, B_reg;
15 wire invalid_red_op, invalid_opcode, invalid;
16
17 //Invalid handling
18 assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
19 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
20 assign invalid = invalid_red_op | invalid_opcode;
21
22 //Registering input signals
23 always @(posedge clk or posedge rst) begin
24     if(rst) begin
25         cin_reg <= 0;
26         red_op_B_reg <= 0;
27         red_op_A_reg <= 0;
28         bypass_B_reg <= 0;
29         bypass_A_reg <= 0;
30         direction_reg <= 0;
31         serial_in_reg <= 0;
32         opcode_reg <= 0;
33         A_reg <= 0;
34         B_reg <= 0;
```

```
35     end else begin
36         cin_reg <= cin;
37         red_op_B_reg <= red_op_B;
38         red_op_A_reg <= red_op_A;
39         bypass_B_reg <= bypass_B;
40         bypass_A_reg <= bypass_A;
41         direction_reg <= direction;
42         serial_in_reg <= serial_in;
43         opcode_reg <= opcode;
44         A_reg <= A;
45         B_reg <= B;
46     end
47 end
48
49 //leds output blinking
50 always @(posedge clk or posedge rst) begin
51     if(rst) begin
52         leds <= 0;
53     end else begin
54         if (invalid)
55             leds <= ~leds;
56         else
57             leds <= 0;
58     end
59 end
60
```

```

61 //ALSU output processing
62 always @(posedge clk or posedge rst) begin
63     if(rst) begin
64         out <= 0;
65     end
66     else begin
67         if (bypass_A_reg && bypass_B_reg)
68             out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
69         else if (bypass_A_reg)
70             out <= A_reg;
71         else if (bypass_B_reg)
72             out <= B_reg;
73         else if (invalid)
74             out <= 0;
75     else begin
76         case (opcode_reg) // third bug is to used the opcode_reg not the opcode
77             3'h0: begin
78                 if (red_op_A_reg && red_op_B_reg)
79                     out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg; // third bug is to replace AND with OR
80                 else if (red_op_A_reg)
81                     out <= |A_reg;
82                 else if (red_op_B_reg)
83                     out <= |B_reg;
84                 else
85                     out <= A_reg | B_reg;
86             end
87             3'h1: begin
88                 if (red_op_A_reg && red_op_B_reg)
89                     out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg; // fourth bug is to replace OR with XOR
90                 else if (red_op_A_reg)
91                     out <= ^A_reg;
92                 else if (red_op_B_reg)
93                     out <= ^B_reg;
94                 else
95                     out <= A_reg ^ B_reg;
96             end
97             3'h2:begin
98                 if(FULL_ADDER == "ON") // fifth bug to add Cin operation in case of full adde
99                     out <= A_reg + B_reg + cin_reg ;
100                 else
101                     out <= A_reg + B_reg ;
102                 end
103             3'h3: out <= A_reg * B_reg;
104             3'h4: begin
105                 if (direction_reg)
106                     out <= {out[4:0], serial_in_reg};
107                 else
108                     out <= {serial_in_reg, out[5:1]};
109             end
110             3'h5: begin
111                 if (direction_reg)
112                     out <= {out[4:0], out[5]};
113                 else
114                     out <= {out[0], out[5:1]};
115                 end
116             default : out <= 0 ;
117         endcase
118     end
119 end
120 end
121
122 endmodule

```

2. Verification plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU_1	When the reset is asserted, the output value should be low	Directed at the start of the simulation	-	A checker in the testbench to make sure the output is correct
ALSU_2	When the invalid is asserted, the output value should be low	Randomization	Cover all values of A	A checker in the testbench to make sure the output is correct
ALSU_3	When the bypass_A is asserted, bypass_B is asserted and INPUT_PRIORITY = "A" the output count_out should take the A	Randomization	Cover all values of B	A checker in the testbench to make sure the output is correct
ALSU_4	When the bypass_A is asserted and bypass_B is deasserted the output count_out should take the A	Randomization	Cover all values of opcode, and transition bin from 0=>1=>2=>3=>4=>5	A checker in the testbench to make sure the output is correct
ALSU_5	When the bypass_B is asserted and bypass_A is deasserted the output count_out should take the A	Randomization	-	A checker in the testbench to make sure the output is correct
ALSU_6	Test all value of opcode and show the output in all case	Randomization	-	A checker in the testbench to make sure the output is correct

3. ALSU Package

```

1 package pack_ALSU;
2 typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
3 typedef enum {Or , Xor , Add , Mult , Shift , Rotate} opcode_valid_e ;
4 localparam MAXPOS = 3 ;
5 localparam MAXNEG = -4 ;
6 localparam zero = 0 ;
7
8 class cla_ALSU ;
9     rand logic signed [2:0] A, B ;
10    rand reg_e opcode ;
11    rand logic rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
12    bit clk ;
13    rand opcode_valid_e opcodes_array[6];
14
15    constraint rst_n {rst dist {0:/99 , 1:/1 } ; }
16
17    constraint input_A {
18        if ( (opcode == ADD ) || (opcode == MULT ) )
19        {
20            A dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
21        }
22        else if ( ((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0) )
23        {
24            B == 0 ;
25            A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
26        }
27        else
28        {
29            A inside { [MAXNEG : MAXPOS] } ;
30        }
31    }
32

```

```

32
33 constraint input_B {
34     if ( (opcode == ADD ) || (opcode == MULT ) )
35     {
36         B dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
37     }
38     else if ( ((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
39     {
40         A == 0 ;
41         B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
42     }
43     else
44     {
45         B inside { [MAXNEG : MAXPOS] } ;
46     }
47 }
48
49 constraint input_opcode {opcode dist {[0:3]:/45 , [4:5]:/50 , [6:7]:/1 } ; }
50
51 constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10 } ; }
52 constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10 } ; }
53
54 constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90 } ; }
55 constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90 } ; }
56

```

```

57
58 constraint c_fixed_array {
59     foreach(opcodes_array[i])
60     {
61         foreach(opcodes_array[j])
62         {
63             if(i != j)
64             {
65                 opcodes_array[i] != opcodes_array[j];
66             }
67         }
68     }
69 }

```

```

70
71 covergroup cvr_gp @(posedge clk) ;
72     A_cp : coverpoint A {
73         bins A_data_0 = {0} ;
74         bins A_data_max = {MAXPOS} ;
75         bins A_data_min = {MAXNEG} ;
76         bins A_data_default = default ;
77         bins A_data_walkingones[] = {1, 2, -4} iff (red_op_A && !red_op_B);
78     }
79
80     B_cp : coverpoint B {
81         bins B_data_0 = {0} ;
82         bins B_data_max = {MAXPOS} ;
83         bins B_data_min = {MAXNEG} ;
84         bins B_data_default = default ;
85         bins B_data_walkingones[] = {1, 2, -4} iff (!red_op_A && red_op_B);
86     }
87
88     ALU_cp : coverpoint opcode {
89         bins Bins_shift[] = {SHIFT , ROTATE} ;
90         bins Bins_arith[] = {ADD , MULT} ;
91         bins Bins_bitwise[] = {OR , XOR} ;
92         illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
93         bins Bins_trans = (OR => XOR => ADD => MULT => SHIFT => Rotate);
94     }
95
96     cin_cp : coverpoint cin {
97         bins cin_data = {0 , 1} ;
98     }
99
100     direction_cp : coverpoint direction {
101         bins direction_data = {0 , 1} ;
102     }
103
104     serial_in_cp : coverpoint serial_in {
105         bins serial_in_data = {0 , 1} ;
106     }
107
108     red_op_A_cp : coverpoint red_op_A {
109         bins red_op_A_LOW_data = {0} ;
110         bins red_op_A_HIGH_data = {1} ;
111     }
112
113     red_op_B_cp : coverpoint red_op_B {
114         bins red_op_B_LOW_data = {0} ;
115         bins red_op_B_HIGH_data = {1} ;
116     }
117 }

```

```

add_mult_cp1 : cross A_cp , B_cp , ALU_cp
{
    bins zero_A_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_0) && binsof(B_cp.B_data_0) ;
    bins max_neg_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_min) && binsof(B_cp.B_data_min) ;
    bins max_pos_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_max) && binsof(B_cp.B_data_max) ;
    option.cross_auto_bin_max = 0 ;
}

opcode_cp2 : cross cin_cp , direction_cp , serial_in_cp , ALU_cp , red_op_A_cp , red_op_B_cp
{
    bins cin_add = binsof(cin_cp.cin_data) && binsof(ALU_cp.Bins_arith) intersect {ADD} ;
    bins serialin_shift = binsof(serial_in_cp.serial_in_data) && binsof(ALU_cp) intersect {SHIFT} ;
    bins direction_shift_rota = binsof(direction_cp) && binsof(ALU_cp.Bins_shift) ;
    option.cross_auto_bin_max = 0 ;
}

or_xor_cp3 : cross A_cp , B_cp , ALU_cp , red_op_A_cp , red_op_B_cp
{
    bins or_xor_data_A = binsof(ALU_cp.Bins_bitwise) && binsof(A_cp.A_data_walkingones) && binsof(B_cp.B_data_0) && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(ALU_cp.Bins_bitwise) ;
    bins or_xor_data_B = binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_0) && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(ALU_cp.Bins_bitwise) ;
    option.cross_auto_bin_max = 0 ;
}

INVALID_cp4 : cross ALU_cp , red_op_A_cp , red_op_B_cp
{
    bins Bins_shift_data = binsof(ALU_cp.Bins_shift) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
    bins Bins_arith_data = binsof(ALU_cp.Bins_arith) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
    option.cross_auto_bin_max = 0 ;
}

endgroup
function new() ;
    cvr_gp = new() ;
endfunction

function void print () ;
    $display("A = 0h%0h , B = 0h%0h , opcode = 0h%0h , rst = 0h%0h , cin = 0h%0h " , this.A , this.B , this.opcode , this.rst , this.cin ) ;
endfunction

endclass

endpackage

```

4. ALSU Testbench

```

1  import pack_ALSU::*;
2
3  module ALSU_tb();
4
5      parameter WIDTH = 3;
6      parameter INPUT_PRIORITY = "A";
7      parameter FULL_ADDER = "ON";
8
9      logic signed [2:0] A, B ;
10     logic [2:0] opcode ;
11     logic      clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
12     logic [15:0] leds;
13     logic signed [5:0] out;
14
15     integer i , j;
16     integer correct_counter = 0 ;
17     integer error_counter = 0 ;
18     logic [5:0] old_count_out ;
19     logic invalid ;
20     logic test ;
21
22     opcode_valid_e opcodes_array_tb[6];
23
24     ALSU #(.INPUT_PRIORITY(INPUT_PRIORITY) , .FULL_ADDER(FULL_ADDER) ) dut (.*);
25
26     cla_ALSU trans1 = new() ;
27     initial begin
28         clk = 0 ;
29         forever
30         begin
31             #20 clk = ~clk ;
32             trans1.clk = clk ;
33         end
34     end
35 end

```

<pre> 38 initial begin 39 cin = 0 ; 40 red_op_A = 0 ; 41 red_op_B = 0 ; 42 bypass_A = 0 ; 43 bypass_B = 0 ; 44 direction = 0 ; 45 serial_in = 0 ; 46 A = 0 ; 47 B = 0 ; 48 opcode = 0 ; 49 rest (); 50 #10 ; 51 trans1.c_fixed_array.constraint_mode(0) ; 52 repeat(10000) begin 53 assert(trans1.randomize()); 54 55 A = trans1.A ; 56 B = trans1.B ; 57 cin = trans1.cin ; 58 red_op_A = trans1.red_op_A ; 59 red_op_B = trans1.red_op_B ; 60 bypass_A = trans1.bypass_A ; 61 bypass_B = trans1.bypass_B ; 62 direction = trans1.direction ; 63 serial_in = trans1.serial_in ; 64 opcode = trans1.opcode ; 65 rst = trans1.rst ; 66 67 sample_data () ; 68 golden_model () ; 69 </pre>	<pre> 70 end 71 trans1.constraint_mode(0) ; 72 trans1.c_fixed_array.constraint_mode(1) ; 73 74 rst = 0 ; 75 red_op_A = 0 ; 76 red_op_B = 0 ; 77 bypass_A = 0 ; 78 bypass_B = 0 ; 79 80 repeat(10000) begin 81 assert(trans1.randomize()); 82 83 A = trans1.A ; 84 B = trans1.B ; 85 cin = trans1.cin ; 86 red_op_A = trans1.red_op_A ; 87 red_op_B = trans1.red_op_B ; 88 bypass_A = trans1.bypass_A ; 89 bypass_B = trans1.bypass_B ; 90 direction = trans1.direction ; 91 serial_in = trans1.serial_in ; 92 // opcode = trans1.opcode ; 93 rst = trans1.rst ; 94 for(int j = 0; j < 6 ; j =j+1) begin 95 opcodes_array_tb[j] = trans1.opcodes_array[j]; 96 opcode = opcodes_array_tb[j]; 97 #45 sample_data(); 98 end 99 sample_data(); 100 golden_model(); 101 </pre>
--	---


```

102     end
103     rst      = 0 ;
104     red_op_A  = 0 ;
105     red_op_B  = 0 ;
106     bypass_A  = 0 ;
107     bypass_B  = 0 ;
108
109     trans1.cvr_gp.start();
110     trans1.opcode = OR;
111     opcode = trans1.opcode;
112     #20
113     trans1.cvr_gp.sample();
114     #20
115     trans1.opcode = XOR ;
116     opcode      = trans1.opcode ;
117     #20
118     trans1.cvr_gp.sample();
119     #20
120     trans1.opcode = ADD ;
121     opcode      = trans1.opcode ;
122     #20
123     trans1.cvr_gp.sample();
124     #20
125     trans1.opcode = MULT ;
126     opcode      = trans1.opcode ;
127     #20
128     trans1.cvr_gp.sample();
129     #20
130     trans1.opcode = SHIFT ;
131     opcode      = trans1.opcode ;
132     #20
133     trans1.cvr_gp.sample();
134     #20
135     trans1.opcode = ROTATE ;
136     opcode      = trans1.opcode ;
137     #20
138     trans1.cvr_gp.sample();
139     #40
140     $display ("error_counter = %0d ",error_counter );
141     $display ("correct_counter = %0d ",correct_counter );
142     #100 ;
143
144     $stop ;
145
146 end
147 task check_result (input logic signed [5:0] expected_result_out , input [15:0] expected_result_leds );
148     @(negedge clk );
149     @(negedge clk );
150     if( (expected_result_out != out) && (expected_result_leds != leds) )
151     begin
152         $display (":error");
153         old_count_out = out ;
154         error_counter = error_counter +1 ;
155         test = 1 ;
156     end
157

```

```

158     else
159     begin
160         correct_counter = correct_counter + 1 ;
161         old_count_out = out ;
162         test = 0 ;
163     end
164 endtask
165 task sample_data () ;
166     if(rst || bypass_A || bypass_B)
167     begin
168         trans1.cvr_gp.stop();
169     end
170     else
171     begin
172         trans1.cvr_gp.start();
173         trans1.cvr_gp.sample();
174     end
175
176 endtask

```

```

178 task golden_model ();
179   invalid = (((red_op_A | red_op_B) & (opcode[1] | opcode[2])) | (opcode[1] & opcode[2]) ) ;
180   if (rst)
181     check_result(0,0);
182   else if(bypass_A && bypass_B)
183     begin
184       if (INPUT_PRIORITY == "A")
185         check_result(A,0);
186       else
187         check_result(B,0);
188     end
189   else if (bypass_A)
190     check_result(A,0);
191   else if (bypass_B)
192     check_result(B,0);
193   else if (invalid)
194     check_result(0, 'hffff);
195   else begin
196     case (opcode)
197       3'h0: begin
198         if (red_op_A && red_op_B)
199           begin
200             if (INPUT_PRIORITY == "A")
201               check_result(|A,0);
202             else
203               check_result(|B,0);
204           end
212       3'h1: begin
213         if (red_op_A && red_op_B)
214           begin
215             if (INPUT_PRIORITY == "A")
216               check_result(^A,0);
217             else
218               check_result(^B,0);
219           end
220         else if (red_op_A)
221           check_result(^A,0);
222         else if (red_op_B)
223           check_result(^B,0);
224         else
225           check_result(A^B,0);
226       end
227       3'h2: begin
228         if(FULL_ADDER == "ON")
229           check_result(A+B+cin,0);
230         else
231           check_result(A+B,0);
232       end
233       3'h3: check_result(A*B,0);
234       3'h4: begin
235         if (direction)
236           check_result({old_count_out[4:0], serial_in},0);
237         else
238           check_result({serial_in, old_count_out[5:1]},0);
239       end
240       3'h5: begin
241         if (direction)
242           check_result({old_count_out[4:0], old_count_out[5]},0);
243         else
244           check_result({old_count_out[0], old_count_out[5:1]},0);
245       end
246     endcase
247   end
248 endtask
249 task rest ();
250   rst = 1 ;
251   #20
252   rst = 0 ;
253 endtask
254 endmodule

```

5. Do File

```

1  vlib work
2  vlog ALSU.v ALSU_tb.sv +cover -covercells
3  vsim -voptargs=+acc work.ALSU_tb -cover
4  add wave *
5  coverage save ALSU_tb.ucdb -onexit
6  run -all
7

```

6. Waveform



7. Transcript

```

#
# error_counter = 0
# correct_counter = 1000000

```


8. Code Coverage

```
2
3 =====
4 == File: ALSU.v
5 =====
6 Statement Coverage:
7   Enabled Coverage      Active      Hits      Misses % Covered
8   -----
9   Stmts                49          49          0    100.0
10
11 =====Statement Details=====
12
13 Statement Coverage for file ALSU.v --
14
15   1      module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, by
16   2      parameter INPUT_PRIORITY = "A";
17   3      parameter FULL_ADDER = "ON";
18   4      input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, dir
19   5      input [2:0] opcode;
20   6      input signed [2:0] A, B;
21   7      output reg [15:0] leds;
22   8      output reg signed [5:0] out;
23   9
24  10      reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_
25  11      reg [2:0] opcode_reg, A_reg, B_reg;
26  12
27  13      wire invalid_red_op, invalid_opcode, invalid;
28  14
29  15      //Invalid handling
30  16          1      896366      assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_
31  17          1      856664      assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
32  18          1      494649      assign invalid = invalid_red_op | invalid_opcode;
33  19
34  20      //Registering input signals
35  21          1      1997369      always @(posedge clk or posedge rst) begin
36  22          if(rst) begin
133
136 Branch Coverage:
137   Enabled Coverage      Active      Hits      Misses % Covered
138   -----
139   Branches                30          30          0    100.0
140
141 =====Branch Details=====
142
143 Branch Coverage for file ALSU.v --
144
145 -----IF Branch-----
146   22          1997369      Count coming in to IF
147   22          1      19911      if(rst) begin
148   33          1      1977458      end else begin
149 Branch totals: 2 hits of 2 branches = 100.0%
150
151 -----IF Branch-----
152   49          2009908      Count coming in to IF
153   49          1      29921      if(rst) begin
154   52          1      1513995      if (invalid)
155   54          1      465992      else
156 Branch totals: 3 hits of 3 branches = 100.0%
157
```



```

29
30 FSM Coverage:
31   Enabled Coverage      Active      Hits      Misses % Covered
32   -----
33   FSMs                                     100.0
34   States                0           0         0      100.0
35   Transitions           0           0         0      100.0
36 Toggle Coverage:
37   Enabled Coverage      Active      Hits      Misses % Covered
38   -----
39   Toggle Bins           118          118         0      100.0
40
41 =====Toggle Details=====
42
43 Toggle Coverage for File ALSU.v --
44
45   Line                      Node      1H->0L      0L->1H  "Coverage"
46   -----
47   4                          serial_in  1           1      100.00
48   4                          rst        1           1      100.00

```

9. function Coverage

```

Covergroup instance \pack_ALSU::cla_ALSU::cvr_gp
    covered/total bins:      100.0%      100      Covered
    missing/total bins:      0           36
    % Hit:                   100.0%      100
    Coverpoint A_cp          100.0%      100      Covered
        covered/total bins:      6           6
        missing/total bins:      0           6
        % Hit:                   100.0%      100
        bin A_data_0            1408         1      Covered
        bin A_data_max          1371         1      Covered
        bin A_data_min          1290         1      Covered
        bin A_data_walkingones[-4] 337         1      Covered
        bin A_data_walkingones[1] 261         1      Covered
        bin A_data_walkingones[2] 334         1      Covered
        default bin A_data_default 3780        Occurred
    Coverpoint B_cp          100.0%      100      Covered
        covered/total bins:      6           6
        missing/total bins:      0           6
        % Hit:                   100.0%      100
        bin B_data_0            1457         1      Covered
        bin B_data_max          1373         1      Covered
        bin B_data_min          1368         1      Covered
        bin B_data_walkingones[-4] 364         1      Covered
        bin B_data_walkingones[1] 368         1      Covered
        bin B_data_walkingones[2] 320         1      Covered
        default bin B_data_default 3797        Occurred
    Coverpoint ALU_cp        100.0%      100      Covered
        covered/total bins:      7           7
        missing/total bins:      0           7
        % Hit:                   100.0%      100
        illegal_bin Bins_invalid 2482        Occurred
        bin Bins_shift[SHIFT]    1291         1      Covered
        bin Bins_shift[ROTATE]   1352         1      Covered
        bin Bins_arith[ADD]      1444         1      Covered
        bin Bins_arith[MULT]     1197         1      Covered
        bin Bins_bitwise[OR]     1326         1      Covered
        bin Bins_bitwise[XOR]    1367         1      Covered
        bin Bins_trans           1           1      Covered
    Coverpoint cin_cp        100.0%      100      Covered

```

Coverpoint cin_cp	100.0%	100	Covered
covered/total bins:	1	1	
missing/total bins:	0	1	
% Hit:	100.0%	100	
bin cin_data	10459	1	Covered
Coverpoint direction_cp	100.0%	100	Covered
covered/total bins:	1	1	
missing/total bins:	0	1	
% Hit:	100.0%	100	
bin direction_data	10459	1	Covered
Coverpoint serial_in_cp	100.0%	100	Covered
covered/total bins:	1	1	
missing/total bins:	0	1	
% Hit:	100.0%	100	
bin serial_in_data	10459	1	Covered
Coverpoint red_op_A_cp	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin red_op_A_LOW_data	5420	1	Covered
bin red_op_A_HIGH_data	5039	1	Covered
Coverpoint red_op_B_cp	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin red_op_B_LOW_data	5468	1	Covered
bin red_op_B_HIGH_data	4991	1	Covered
Cross add_mult_cp1	100.0%	100	Covered
covered/total bins:	3	3	
missing/total bins:	0	3	
% Hit:	100.0%	100	
bin zero_A_add	47	1	Covered
bin max_neg_a_add	45	1	Covered
bin max_pos_add	56	1	Covered
bin max_pos_add	56	1	Covered
Cross opcode_cp2	100.0%	100	Covered
covered/total bins:	3	3	
missing/total bins:	0	3	
% Hit:	100.0%	100	
bin cin_add	1444	1	Covered
bin serialin_shift	1291	1	Covered
bin direction_shift_rota	2643	1	Covered
Cross or_xor_cp3	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin or_xor_data_A	64	1	Covered
bin or_xor_data_B	60	1	Covered
Cross INVALID_cp4	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin Bins_shift_data	1842	1	Covered
bin Bins_arith_data	1978	1	Covered
TOTAL COVERGROUP COVERAGE: 100.0% COVERGROUP TYPES: 1			

Question 2

```
1  Q2
2
3  1)
4  property prop1;
5      @(posedge clk)
6      disable iff (reset)
7      a |-> ##2 b
8  endproperty
9
10     dollar1_ass1: assert property (prop1) ;
11     dollar1_cover1: cover property (prop1) ;
12
13  2)
14  property prop2;
15      @(posedge clk)
16      disable iff (reset)
17      a && b |-> ##[1:3] c
18  endproperty
19
20     dollar1_ass2: assert property (prop2) ;
21     dollar1_cover2: cover property (prop2) ;
22
23  3)
24  property prop3;
25      @(posedge clk)
26      disable iff (reset)
27      s11b |-> ##2 !b
28  endproperty
29
30     dollar1_ass3: assert property (prop3) ;
31     dollar1_cover3: cover property (prop3) ;
32
33  4.1)
34
35  property decoder_output_one_bit_high;
36      @(posedge clk)
37      disable iff (reset)
38      (countones(Y) == 1);
39  endproperty
40
41
42  dollar1_ass4: assert property (decoder_output_one_bit_high) ;
43  dollar1_cover4: cover property (decoder_output_one_bit_high) ;
44
45
46  4.2)
47  property priority_encoder_valid_low;
48      @(posedge clk) disable iff (reset) (D == 4'b0000) |=> (valid == 0);
49  endproperty
50
51
52  dollar1_ass5: assert property (priority_encoder_valid_low) ;
53  dollar1_cover5: cover property (priority_encoder_valid_low) ;
```


Question 3_1

➤ priority encoder

1. Code Design

```
1  module priority_enc (  
2      input  clk,  
3      input  rst,  
4      input  [3:0] D,  
5      output reg [1:0] Y,  
6      output reg valid  
7  );  
8  
9      always @(posedge clk) begin  
10         if (rst)  
11             begin  
12                 Y <= 2'b0;  
13                 valid <= 1'b0 ;  
14             end  
15         else  
16             begin  
17                 casex (D)  
18                     4'b1000: Y <= 0;  
19                     4'bX100: Y <= 1;  
20                     4'bXX10: Y <= 2;  
21                     4'bXXX1: Y <= 3;  
22                     default: Y <= 2'b0;  
23                 endcase  
24                 valid <= (~|D)? 1'b0: 1'b1;  
25             end  
26         end  
27     endmodule
```

2. Verification plan

	LABEL	Description	Stimulus Generation	Function check
1				
2	case_1	when the reset is asserted . The output C value must be low && valid must be equal 0	directed at the start of the simulation	A checker in the testbench to make sure the output is correct
3	case_2	when D = 4'b1000 . The output C value must be low && valid must be equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
4	case_3	when D = 4'b0100 . The output C value must be equal 1 && valid must be equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
5	case_4	when D = 4'b1010 . The output C value must be equal 1 && valid must be equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
6	case_5	when D = 4'b0101 . The output C value must be equal 1 && valid must be equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
7	case_6	when D = 4'b1000 . The output C value must be low && valid must be equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
8	case_7	when D = 4'b0000 . The output C value must be low "default case" && valid must be equal 0	directed during the simulation	A checker in the testbench to make sure the output is correct

3. priority encoder Testbench

```

1  module priority_enc_tb ();
2      logic      clk      ;
3      logic      rst      ;
4      logic [3:0] D       ;
5      logic [1:0] Y       ;
6      logic      valid    ;
7      integer correct_counter = 0 ;
8      integer error_counter = 0 ;
9      priority_enc dut (
10         .clk(clk),
11         .rst(rst),
12         .D(D),
13         .Y(Y),
14         .valid(valid)
15     );
16     always #20 clk = ~clk ;
17     property prop1 ;
18         @(posedge clk )
19         disable iff (rst)
20         ( D === 4'b1000 ) | => (Y === 0) && valid;
21     endproperty
22     dollar1_ass: assert property (prop1) ;
23     dollar1_cover: cover property (prop1) ;
24     property prop2 ;
25         @(posedge clk )
26         disable iff (rst)
27         (( D === 4'b0100 ) || ( D === 4'b1100 ) ) | => (Y === 1)&& valid;
28     endproperty
29
30     dollar2_ass: assert property (prop2) ;
31     dollar2_cover: cover property (prop2) ;

```

```

58     initial                                     88
59         begin                                     89
60             clk = 0 ;                             90
61             D = 0 ;                               91
62             rest () ;                             92
63                                                     93
64             D = 4'b1000 ;                         94
65             @(negedge clk )                      95
66                                                     96
67             D = 4'b0100 ;                         97
68             @(negedge clk )                      98
69                                                     99
70             D = 4'b1010 ;                        100
71             @(negedge clk )                      101
72                                                     102
73             D = 4'b1000 ;                        103
74             @(negedge clk )                      104
75                                                     105
76             D = 4'b0101 ;                        106
77             @(negedge clk )                      107
78                                                     108
79             D = 4'b1000 ;                        109
80             @(negedge clk )                      110
81                                                     111
82                                                     112
83             D = 4'b0000 ;                        113
84             @(negedge clk )                      114
85                                                     115
86             D = 4'b1000 ;                        116
87             @(negedge clk )                      117
88
89             D = 4'b1101 ;
90             @(negedge clk )
91
92             D = 4'b0011 ;
93             @(negedge clk )
94
95             D = 4'b1011 ;
96             @(negedge clk )
97
98             D = 4'b0111 ;
99             @(negedge clk )
100
101             D = 4'b1111 ;
102             @(negedge clk )
103
104             $display ("error_counter = %0d " ,error_counter );
105             $display ("correct_counter = %0d " ,correct_counter );
106             $stop ;
107
108         end
109         task rest () ;
110             rst = 0 ;
111             #1
112             rst = 1 ;
113             check_result(2'b0 , 0 );
114             rst = 0 ;
115
116     endtask
117
118 endmodule

```

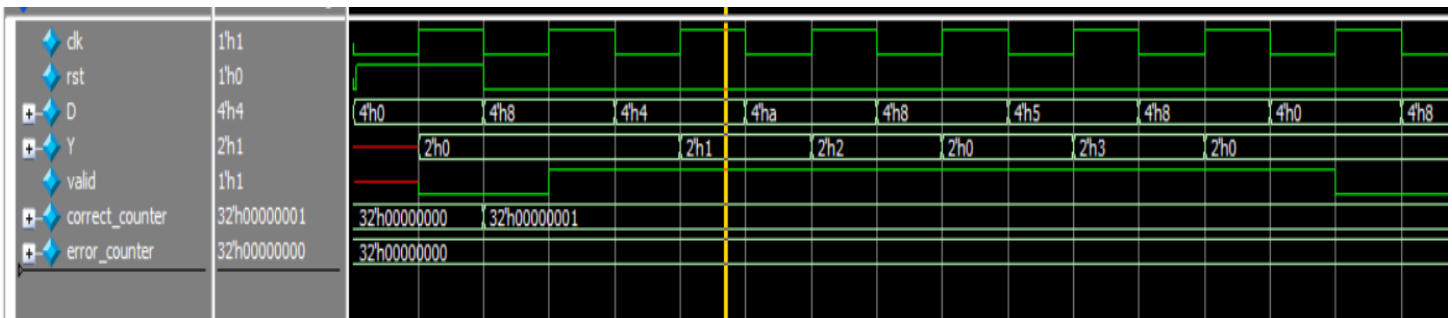
4. Do File

```






1 vlib work
2 vlog priority_enc.v priority_enc_tb.svh +cover -covercells
3 vsim -voptargs=+acc work.priority_enc_tb -cover
4 add wave *
5 coverage save priority_enc_tb.ucdb -onexit
6 run -all

```

5. Waveform



6. assertion

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Index
 /priority_enc_tb/do... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...)	✓	
 /priority_enc_tb/do... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...)	✓	
 /priority_enc_tb/do... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...)	✓	
 /priority_enc_tb/do... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...)	✓	
 /priority_enc_tb/do... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...)	✓	

7. Assertion coverage

Name	Language	Enabled	Log	Success	Unli...	Weight	Simple C...	Simple Graph	Release	Time /	Simple /	Simple /	Simple /
▶ /priority_enc_tb/do... SVA	✓	Off	4	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
▶ /priority_enc_tb/do... SVA	✓	Off	3	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
▶ /priority_enc_tb/do... SVA	✓	Off	5	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
▶ /priority_enc_tb/do... SVA	✓	Off	8	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
▶ /priority_enc_tb/do... SVA	✓	Off	1	1	Unli...	1	100%	100%	✓	0	0	0 ns	0

8. Code Coverage

Statement Coverage:

Enabled Coverage	Active	Hits	Misses	% Covered
-----	-----	-----	-----	-----
Stmts	9	9	0	100.0

====Statement Details=====

Statement Coverage for file priority_enc.v --

```
1      module priority_enc (
2          input  clk,
3          input  rst,
4          input  [3:0] D,
5          output reg [1:0] Y,
6          output reg valid
7      );
8
9          1      always @(posedge clk) begin
10              if (rst)
11                  begin
12                      1      Y <= 2'b0;
13                      1      valid <= 1'b0 ;
14                  end
15              else
16                  begin
17                      casex (D)
18                          1      4      4'b1000: Y <= 0;
19                          1      3      4'bX100: Y <= 1;
20                          1      5      4'bXX10: Y <= 2;
21                          1      9      4'bXXX1: Y <= 3;
22                          1      default: Y <= 2'b0;
23                      endcase
24                      1      valid <= (~|D)? 1'b0: 1'b1;
25                  end
26              end
```

Branch Coverage:

Enabled Coverage	Active	Hits	Misses	% Covered
-----	-----	-----	-----	-----
Branches	7	7	0	100.0

====Branch Details=====

Branch Coverage for file priority_enc.v --

-----IF Branch-----

Line	Count	Branch
10	23	Count coming in to IF
10	1	if (rst)
15	22	else

Branch totals: 2 hits of 2 branches = 100.0%

-----CASE Branch-----

Line	Count	Branch
17	22	Count coming in to CASE
18	4	4'b1000: Y <= 0;
19	3	4'bX100: Y <= 1;
20	5	4'bXX10: Y <= 2;
21	9	4'bXXX1: Y <= 3;
22	1	default: Y <= 2'b0;

Branch totals: 5 hits of 5 branches = 100.0%

Transitions	0	0	0	100.0
Toggle Coverage:				
Enabled Coverage	Active	Hits	Misses	% Covered

Toggle Bins	18	18	0	100.0
=====Toggle Details=====				
Toggle Coverage for File <u>priority_enc.v</u> --				
Line	Node	1H->0L	0L->1H	"Coverage"

2	clk	1	1	100.00
3	rst	1	1	100.00
4	D[3]	1	1	100.00
4	D[2]	1	1	100.00
4	D[1]	1	1	100.00
4	D[0]	1	1	100.00
5	Y[1]	1	1	100.00
5	Y[0]	1	1	100.00
6	valid	1	1	100.00

Question 3_2

➤ Alu

1. Code Design

```

1  module ALU_4_bit (
2
3
4      localparam      Add          = 3'b00; // A + B
5      localparam      Sub          = 3'b01; // A - B
6      localparam      Not_A        = 3'b10; // ~A
7      localparam      ReductionOR_B = 3'b11; // |B
8
9      // Do the operation
10     always @* begin
11         case (Opcode)
12             Add:      Alu_out = A + B;
13             Sub:      Alu_out = A - B;
14             Not_A:    Alu_out = ~A;
15             ReductionOR_B: Alu_out = |B;
16             default:  Alu_out = 5'b0;
17         endcase
18     end // always @ *
19
20     // Register output C
21     always @(posedge clk or posedge reset) begin
22         if (reset)
23             C <= 5'b0;
24         else
25             C <= Alu_out;
26     end
27
28 endmodule

```

2. Verification plan

LABEL	Description	Stimulus Generation	Function check
case_1	when the reset is asserted . The output C value must be low	directed at the start of the simulation	A checker in the testbench to make sure the output is correct
case_2	verifying maximum negative value on A and maximum negative value on B && Opcode equal 0	directed during the simulation	A checker in the testbench to make sure the output is correct
case_3	verifying maximum negative value on A and ZERO value on B && Opcode equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
case_4	verifying maximum negative value on A and maximum positive value on B && Opcode equal 2	directed during the simulation	A checker in the testbench to make sure the output is correct
case_5	verifying ZERO value on A and maximum negative value on B && Opcode equal 3	directed during the simulation	A checker in the testbench to make sure the output is correct
case_6	verifying maximum positive value on A and ZERO value on B && Opcode equal 0	directed during the simulation	A checker in the testbench to make sure the output is correct
case_7	verifying ZERO value on A and ZERO value on B && Opcode equal 1	directed during the simulation	A checker in the testbench to make sure the output is correct
case_8	verifying maximum positive value on A and ZERO value on B && Opcode equal 2	directed during the simulation	A checker in the testbench to make sure the output is correct

3. Alu Testbench

```

1  module ALU_4_bit_tb ();
2      localparam MAXPOS = 7 ;
3      localparam MAXNEG = -8 ;
4      logic clk ;
5      logic reset ;
6      logic [1:0] Opcode;
7      logic signed [3:0] A ;
8      logic signed [3:0] B ;
9      logic signed [4:0] C ;
10     localparam      Add      = 3'b00; // A + B
11     localparam      Sub      = 3'b01; // A - B
12     localparam      Not_A    = 3'b10; // ~A
13     localparam      ReductionOR_B = 3'b11; // |B|
14
15     ALU_4_bit dut (
16         .clk(clk),
17         .reset(reset),
18         .Opcode(Opcode),
19         .A(A),
20         .B(B),
21         .C(C)
22     );
23
24     always #20 clk = ~clk ;
25
26     property prop1_Add;
27         @(posedge clk)
28             disable iff (reset)
29             (Opcode === Add)  | =>  (C === ($past(A)+$past(B)));
30     endproperty
31
32     assert_add: assert property (prop1_Add) ;
33     cover_add: cover property (prop1_Add) ;
34

```

```

36     property prop1_sub;
37         @(posedge clk)
38         disable iff (reset)
39         (Opcode === Sub)    | =>  (C === ($past(A)-$past(B)));
40     endproperty
41
42     assert_sub: assert property (prop1_sub) ;
43     cover_sub:  cover property (prop1_sub) ;
44
45     property prop1_not_A;
46         @(posedge clk)
47         disable iff (reset)
48         (Opcode === Not_A)   | =>  (C === (~$past(A)));
49     endproperty
50
51     assert_not_A: assert property (prop1_not_A) ;
52     cover_not_A:  cover property (prop1_not_A) ;
53
54
55     property prop1_Reduc;
56         @(posedge clk)
57         disable iff (reset)
58         (Opcode === ReductionOR_B)    | =>  (C === ($past(B)));
59     endproperty
60
61     assert_Reduc: assert property (prop1_Reduc) ;
62     cover_Reduc:  cover property (prop1_Reduc) ;

```

```

63
64     property prop1_reset;
65         @(posedge clk)
66         reset | =>  (C === 0);
67     endproperty
68
69     assert_reset: assert property (prop1_reset) ;
70     cover_reset:  cover property (prop1_reset) ;
71
72     initial
73     begin
74         clk = 0 ;
75         A = 0 ;
76         B = 0 ;
77         Opcode = 0 ;
78         rest();
79
80         reset = 0 ;
81         A = MAXNEG ;
82         B = MAXNEG ;
83
84         @(negedge clk );
85
86         A = MAXNEG ;
87         B = MAXPOS ;
88
89         @(negedge clk );
90
91         B = MAXNEG ;
92         A = MAXPOS ;
93
94         @(negedge clk );
95

```


96	A = MAXPOS ;	123			A = MAXNEG ;
97	B = MAXPOS ;	124			B = MAXPOS ;
98		125			
99	@(negedge clk);	126			@(negedge clk);
100		127			
101	A = MAXNEG ;	128			B = MAXNEG ;
102	B = 0 ;	129			A = MAXPOS ;
103		130			
104	@(negedge clk);	131			@(negedge clk);
105		132			
106	A = 0 ;	133			A = MAXPOS ;
107	B = 0 ;	134			B = MAXPOS ;
108		135			
109	@(negedge clk);	136			@(negedge clk);
110		137			
111	A = MAXPOS ;	138			A = MAXNEG ;
112	B = 0 ;	139			B = 0 ;
113	@(negedge clk);	140			
114		141			@(negedge clk);
115	reset = 0 ;	142			
116		143			A = 0 ;
117	@(negedge clk);	144			B = 0 ;
118	Opcode = 1 ;	145			
119	A = MAXNEG ;	146			@(negedge clk);
120	B = MAXNEG ;	147			
121		148			A = MAXPOS ;
122	@(negedge clk);	149			B = 0 ;
		150			
		151			@(negedge clk);
		152			
156	@(negedge clk);	188			A = MAXPOS ;
157	Opcode = 2 ;	189			B = 0 ;
158	A = MAXNEG ;	190			
159	B = MAXNEG ;	191			@(negedge clk);
160		192			
161	@(negedge clk);	193			reset = 0 ;
162		194			
163	A = MAXNEG ;	195			@(negedge clk);
164	B = MAXPOS ;	196			
165		197			reset = 0 ;
166	@(negedge clk);	198			
167		199			@(negedge clk);
168	B = MAXNEG ;	200			Opcode = 3 ;
169	A = MAXPOS ;	201			A = MAXNEG ;
170		202			B = MAXNEG ;
171	@(negedge clk);	203			
172		204			@(negedge clk);
173	A = MAXPOS ;	205			
174	B = MAXPOS ;	206			A = MAXNEG ;
175		207			B = MAXPOS ;
176	@(negedge clk);	208			
177		209			@(negedge clk);
178	A = MAXNEG ;	210			
179	B = 0 ;	211			B = MAXNEG ;
180		212			A = MAXPOS ;
181	@(negedge clk);	213			
182		214			@(negedge clk);
183	A = 0 ;	215			
184	B = 0 ;	216			A = MAXPOS ;
185		217			B = MAXPOS ;
186	@(negedge clk);				

```

218         A = MAXNEG ;
219         B = 0 ;
220         @(negedge clk );
221         A = 0 ;
222         B = 0 ;
223         @(negedge clk );
224
225         A = MAXPOS ;
226         B = 0 ;
227
228         @(negedge clk );
229         Opcode = 0 ;
230         @(negedge clk );
231         $stop();
232     end
233     task rest ();
234         reset = 0 ;
235         #1
236         reset = 1 ;
237         #40;
238         reset = 0 ;
239     endtask
240
241
242
243     endmodule

```

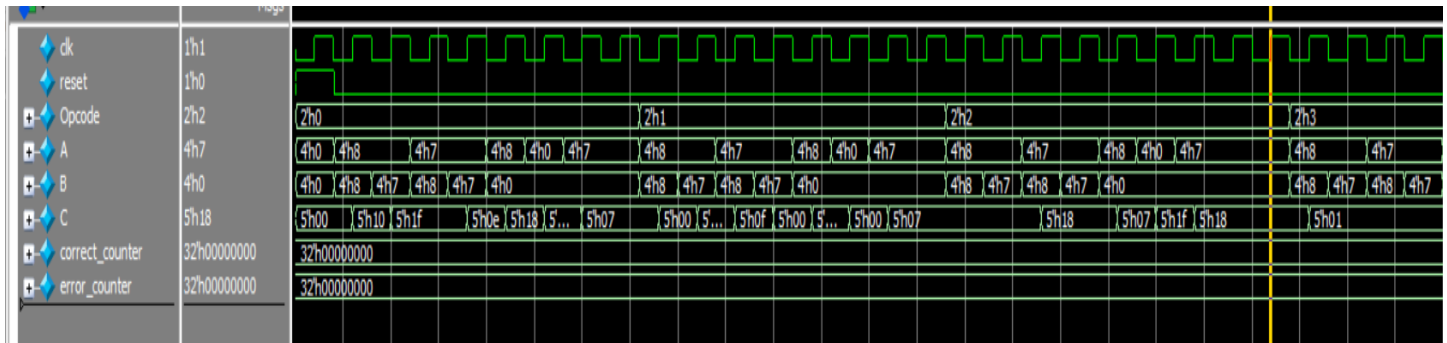
1. Do File

```

1  vlib work
2  vlog ALU.v alu_tb.sv +cover -covercells
3  vsim -voptargs=+acc work.ALU_4_bit_tb -cover
4  add wave *
5  coverage save alu_tb.ucdb -onexit
6  run -all

```

3. Waveform



4. Assertion

Name	Assertion Type	Language	Enabled	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	Any	Assertion Expression	OK
/ALU_4_bit_tb/ass...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...)	0	0	0 ns	0	assert(@(posedge clk) disable iff (...)	✓
/ALU_4_bit_tb/ass...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...)	0	0	0 ns	0	assert(@(posedge clk) disable iff (...)	✓
/ALU_4_bit_tb/ass...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...)	0	0	0 ns	0	assert(@(posedge clk) disable iff (...)	✓
/ALU_4_bit_tb/ass...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...)	0	0	0 ns	0	assert(@(posedge clk) disable iff (...)	✓
/ALU_4_bit_tb/ass...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) (reset) =>...	0	0	0 ns	0	assert(@(posedge clk) (reset) =>...	✓

5. Assertion coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/ALU_4_bit_tb/cov...	SVA	✓	Off	8	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
/ALU_4_bit_tb/cov...	SVA	✓	Off	8	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
/ALU_4_bit_tb/cov...	SVA	✓	Off	9	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
/ALU_4_bit_tb/cov...	SVA	✓	Off	7	1	Unli...	1	100%	100%	✓	0	0	0 ns	0
/ALU_4_bit_tb/cov...	SVA	✓	Off	1	1	Unli...	1	100%	100%	✓	0	0	0 ns	0

6. Code Coverage

```
Statement Coverage:
Enabled Coverage      Active      Hits      Misses % Covered
-----
Stmts                8          8          0      100.0

=====Statement Details=====

Statement Coverage for file ALU.v --

1          module ALU_4_bit (
2              input  clk,
3              input  reset,
4              input  [1:0] Opcode, // The opcode
5              input  signed [3:0] A, // Input data A in 2's complement
6              input  signed [3:0] B, // Input data B in 2's complement
7
8              output reg signed [4:0] C // ALU output in 2's complement
9
10             );
11
12             reg signed [4:0]      Alu_out; // ALU output in 2's complement
13
14             localparam      Add      = 3'b00; // A + B
15             localparam      Sub      = 3'b01; // A - B
16             localparam      Not_A    = 3'b10; // ~A
17             localparam      ReductionOR_B = 3'b11; // |B
18
19             // Do the operation
20             1          30      always @* begin
21                 case (Opcode)
22                     1          9          Add:      Alu_out = A + B;
23                     1          7          Sub:      Alu_out = A - B;
24                     1          7          Not_A:    Alu_out = ~A;
25                     1          7          ReductionOR_B: Alu_out = |B;
26                     1          E          default: Alu_out = 5'b0;
27
28             endcase
29
30             end
31
32             if (reset)
33                 C = 5'b0;
34             else
35                 C = Alu_out;
36
37             end
38
39             endmodule
```

```
Branch Coverage:
Enabled Coverage      Active      Hits      Misses % Covered
-----
Branches              6          6          0      100.0

=====Branch Details=====

Branch Coverage for file ALU.v --

-----CASE Branch-----
21          30      Count coming in to CASE
22          1          9          Add:      Alu_out = A + B;
23          1          7          Sub:      Alu_out = A - B;
24          1          7          Not_A:    Alu_out = ~A;
25          1          7          ReductionOR_B: Alu_out = |B;
26          1          E          default: Alu_out = 5'b0;
Branch totals: 4 hits of 4 branches = 100.0%

-----IF Branch-----
32          29      Count coming in to IF
32          1          2          if (reset)
34          1          27         else
Branch totals: 2 hits of 2 branches = 100.0%
```

Toggle Coverage:

Enabled Coverage	Active	Hits	Misses	% Covered
Toggle Bins	44	44	0	100.0

=====toggle Details=====

Toggle Coverage for File ALU.v --

Line	Node	1H->0L	0L->1H	"Coverage"
2	<u>clk</u>	1	1	100.00
3	reset	1	1	100.00
4	Opcode[1]	1	1	100.00
4	Opcode[0]	1	1	100.00
5	A[3]	1	1	100.00
5	A[2]	1	1	100.00
5	A[1]	1	1	100.00
5	A[0]	1	1	100.00
6	B[3]	1	1	100.00
6	B[2]	1	1	100.00
6	B[1]	1	1	100.00
6	B[0]	1	1	100.00
8	C[4]	1	1	100.00
8	C[3]	1	1	100.00
8	C[2]	1	1	100.00
8	C[1]	1	1	100.00
8	C[0]	1	1	100.00
12	<u>Alu_out</u> [4]	1	1	100.00
12	<u>Alu_out</u> [3]	1	1	100.00
12	<u>Alu_out</u> [2]	1	1	100.00
12	<u>Alu_out</u> [1]	1	1	100.00
12	<u>Alu_out</u> [0]	1	1	100.00

DIRECTIVE COVERAGE:

Name	Design Unit	Design UnitType	Lang	File(Line)	Count	Status
/ALU_4_bit_tb/ <u>cover_add</u>	ALU_4_bit_tb	Verilog	SVA	alu_tb.sv(41)	8	Covered
/ALU_4_bit_tb/ <u>cover_sub</u>	ALU_4_bit_tb	Verilog	SVA	alu_tb.sv(51)	8	Covered
/ALU_4_bit_tb/ <u>cover_not_A</u>	ALU_4_bit_tb	Verilog	SVA	alu_tb.sv(60)	9	Covered
/ALU_4_bit_tb/ <u>cover_Reduc</u>	ALU_4_bit_tb	Verilog	SVA	alu_tb.sv(70)	7	Covered
/ALU_4_bit_tb/ <u>cover_reset</u>	ALU_4_bit_tb	Verilog	SVA	alu_tb.sv(78)	1	Covered

TOTAL DIRECTIVE COVERAGE: 100.0% COVERS: 5

Question 4

➤ Counter

1. Code Design

```
7 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
8 module counter(counter_inter.DUT counter_if);
9
10 always @(posedge counter_if.clk , negedge counter_if.rst_n ) begin
11     if (!counter_if.rst_n)
12         counter_if.count_out <= 0;
13     else if (!counter_if.load_n)
14         counter_if.count_out <= counter_if.data_load;
15     else if (counter_if.ce)
16         if (counter_if.up_down)
17             counter_if.count_out <= counter_if.count_out + 1;
18         else
19             counter_if.count_out <= counter_if.count_out - 1;
20 end
21
22 assign counter_if.max_count = (counter_if.count_out == {counter_if.WIDTH{1'b1}})? 1:0;
23 assign counter_if.zero = (counter_if.count_out == 0)? 1:0;
24
25 endmodule
```

2. Interface

```
1 interface counter_inter(clk) ;
2
3     parameter WIDTH = 4;
4     input clk;
5     logic rst_n;
6     logic load_n;
7     logic up_down;
8     logic ce;
9     logic [WIDTH-1:0] data_load;
10    logic [WIDTH-1:0] count_out;
11    logic max_count;
12    logic zero;
13
14    modport TEST (output rst_n, load_n, up_down , ce , data_load , input count_out , max_count , zero , clk ) ;
15
16    modport DUT (input rst_n, load_n, up_down , clk , ce , data_load , output count_out , max_count , zero ) ;
17
18    modport MONITOR (input rst_n, load_n, up_down , clk , ce , data_load , count_out , max_count , zero ) ;
19
20 endinterface
```

3. Top module

```
1  module TOP_counter ();
2
3      bit clk ;
4      initial begin
5          clk = 0;
6          forever
7              #1 clk = ~clk;
8      end
9
10     counter_inter  counter_if (clk) ;
11     counter  DUT (counter_if);
12     bind counter  sva_t  sva (counter_if);
13     counter_tb  test (counter_if);
14
15
16 endmodule
```

4. Testbench

```
1  import pack_count::*;
2  module counter_tb(counter_inter.TEST counter_if);
3      integer i ;
4      logic [counter_if.WIDTH:0] old_count_out ;
5      cla_count trans1 = new() ;
6      initial begin
7          forever
8          begin
9              #20 ;
10             trans1.clk = counter_if.clk ;
11         end
12     end
13     initial begin
14         rest () ;
15         @(posedge counter_if.clk );
16         for(i=0; i<1000 ;i=i+1) begin
17             trans1.randomize();
18             //@(negedge counter_if.clk );
19             //trans1.print() ;
20             counter_if.data_load = trans1.data_load ;
21             counter_if.ce = trans1.ce ;
22             counter_if.up_down = trans1.up_down ;
23             counter_if.load_n = trans1.load_n ;
24             counter_if.rst_n = trans1.rst_n ;
25             @(negedge counter_if.clk );
26             trans1.count_out = counter_if.count_out ;
27         end
28         $stop ;
29     end
30     task rest () ;
31         counter_if.rst_n = 1 ;
32         #20
33         counter_if.rst_n = 0 ;
34     endtask
35 endmodule
```


5. Package

```
1 package pack_count;
2   class cla_count ;
3     parameter WIDTH = 4;
4     rand logic [WIDTH-1:0] data_load ;
5     rand logic          ce ;
6     rand logic          up_down ;
7     rand logic          load_n ;
8     rand logic          rst_n ;
9     bit                clk ;
10    logic [WIDTH-1:0] count_out ;
11    parameter max_value = {{WIDTH{1'b1}}} ;
12    parameter zero = 0 ;
13    constraint enable {ce dist {1:/70 , 0:/30 } ; }
14    constraint load {load_n dist {1:/30 , 0:/70 } ; }
15    constraint rst {rst_n dist {1:/99 , 0:/1 } ; }
16    covergroup covgr @(posedge clk) ;
17      load_n_cp : coverpoint load_n {
18        bins cove_asserted = {0} ;
19      }
20    count_out_cp1 : coverpoint count_out iff (rst_n&&ce&&up_down) ;
21    count_out_cp2 : coverpoint count_out iff(rst_n && ce && up_down)
22    {
23      bins max_zero = (max_value => zero) ;
24    }
25    count_out_cp3 : coverpoint count_out iff(rst_n && ce && !up_down) ;
26    count_out_cp4 : coverpoint count_out iff(rst_n && ce && !up_down)
27    {
28      bins zero_max = ( zero => max_value ) ;
29    }
30  endgroup
31  function new() ;
32    covgr = new();
33  endfunction
34  endclass
35 endpackage
36
```

6. SVA

```
1 module sva_t(counter_inter.DUT counter_if);
2
3     property prop1 ;
4         @(posedge counter_if.clk )
5             disable iff (!counter_if.rst_n)
6                 (!counter_if.load_n) |>= (counter_if.count_out === $past(counter_if.data_load)) ;
7     endproperty
8     dollar1_ass: assert property (prop1) ;
9     dollar1_cover: cover property (prop1) ;
10
11     property prop2 ;
12         @(posedge counter_if.clk )
13             disable iff (!counter_if.rst_n)
14                 (counter_if.load_n && !counter_if.ce) |>= counter_if.count_out === $past(counter_if.count_out) ;
15     endproperty
16     dollar2_ass: assert property (prop2) ;
17     dollar2_cover: cover property (prop2) ;
18
19     property prop3 ;
20         @(posedge counter_if.clk )
21             disable iff (!counter_if.rst_n)
22                 (counter_if.load_n && counter_if.ce && counter_if.up_down) |>= (counter_if.count_out === $past(counter_if.count_out) + 1'b1 ) ;
23     endproperty
24
25     dollar3_ass: assert property (prop3) ;
26     dollar3_cover: cover property (prop3) ;
27
28     property prop4 ;
29         @(posedge counter_if.clk )
30             disable iff (!counter_if.rst_n)
31                 (counter_if.load_n && counter_if.ce && !counter_if.up_down) |>= (counter_if.count_out === $past(counter_if.count_out) - 1'b1 ) ;
32     endproperty
33     dollar4_ass: assert property (prop4) ;
34
35
36     property prop5;
37     @(posedge counter_if.clk) (!counter_if.rst_n) |>= (counter_if.count_out === 0) ;
38     endproperty
39
40     dollar5_ass: assert property (prop5) ;
41     dollar5_cover: cover property (prop5) ;
42
43     property prop6;
44     @(posedge counter_if.clk) ( counter_if.count_out === {counter_if.WIDTH{1'b1}} )|>= counter_if.max_count ;
45     endproperty
46
47     dollar6_ass: assert property (prop6) ;
48     dollar6_cover: cover property (prop6) ;
49
50     property prop7;
51     @(posedge counter_if.clk) ( counter_if.count_out === 0 )|>= counter_if.zero ;
52     endproperty
53
54     dollar7_ass: assert property (prop7) ;
55     dollar7_cover: cover property (prop7) ;
56
57 endmodule
```

7. Do File

```

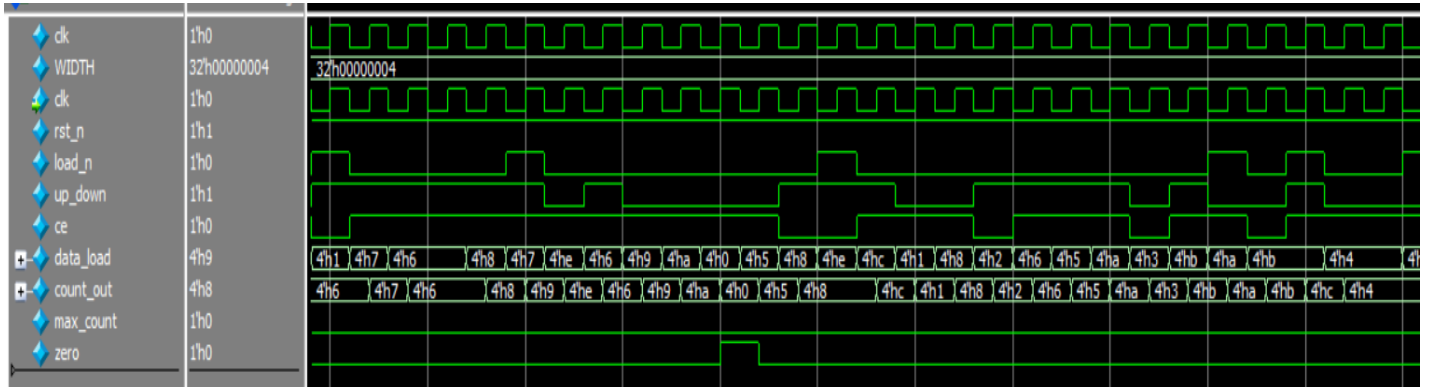
1  vlib work
2  vlog counter.sv counter_tb.sv sva_t.sv TOP_counter.sv counter_inter.sv pack_count.sv +cover -covercells
3  vsim -voptargs=+acc work.TOP_counter -cover
4  add wave *
5  coverage save TOP_counter.ucdb -onexit
6  add wave -position insertpoint \
7  sim:/TOP_counter/DUT/counter_if/WIDTH \
8  sim:/TOP_counter/DUT/counter_if/clk \
9  sim:/TOP_counter/DUT/counter_if/rst_n \
10 sim:/TOP_counter/DUT/counter_if/load_n \
11 sim:/TOP_counter/DUT/counter_if/up_down \
12 sim:/TOP_counter/DUT/counter_if/ce \
13 sim:/TOP_counter/DUT/counter_if/data_load \
14 sim:/TOP_counter/DUT/counter_if/count_out \
15 sim:/TOP_counter/DUT/counter_if/max_count \
16 sim:/TOP_counter/DUT/counter_if/zero
17
18 run -all
19

```

8. Verification plan

	A	B	C	D	E	
1	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check	Co
2	COUNTER_1	When the reset is asserted, the output counter value should be low	Directed at the start of the sim, then randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the testbench to make sure the output is correct	
3	COUNTER_2	When the load is asserted, the output count_out should take the value of the load_data input	Randomization under constraints on the load signal to be off 70% of the time	Cover all values of load data	A checker in the testbench to make sure the output is correct	
4	COUNTER_3	When the load/reset is deasserted, the output count_out increment as long as the enable is active and up_down is high	Randomization for up_down, and randomization for enable to be 70% of the time	Cover all values of count_out, and transition bin from max to zero	A checker in the testbench to make sure the output is correct	
5	COUNTER_4	When the load/reset is deasserted, the output count_out increment as long as the enable is active and up_down is low	Randomization for up_down, and randomization for enable to be 70% of the time	Cover all values of count_out, and transition bin from zero to max	A checker in the testbench to make sure the output is correct	

9. Waveform



10.Assertion

name	language	source	log	cover	status	init	target	init res	target res	init time	target time	init off	target off	assert(@posedge counter_if.clk) ...
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓
/TOP_counter/DUT... Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@posedge counter_if.clk) ...	0 ns	0 off	assert(@posedge counter_if.clk) ...	✓

11.Assertion coverage

name	language	source	log	cover	status	init	target	init res	target res	init time	target time	init off	target off	assert(@posedge counter_if.clk) ...
/TOP_counter/DUT... SVA	✓	Off	672	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	80	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	129	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	93	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	13	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	53	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0
/TOP_counter/DUT... SVA	✓	Off	110	1	Unli...	1	100%	✓	0	0	0 ns	0	0	0

12. Code Coverage

```
3 =====
4 == File: counter.v
5 =====
6 Statement Coverage:
7   Enabled Coverage      Active   Hits   Misses % Covered
8   -----
9   Stmts                7        7      0    100.0
10
11 Statement Details
12
13 Statement Coverage for file counter.v --
14
15 1 ///////////////////////////////////////////////////////////////////
16 2 // Author: Kareem Waseem
17 3 // Course: Digital Verification using SV & UVM
18 4 //
19 5 // Description: Counter Design
20 6 //
21 7 ///////////////////////////////////////////////////////////////////
22 8 module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
23 9 parameter WIDTH = 4;
24 10 input clk;
25 11 input rst_n;
26 12 input load_n;
27 13 input up_down;
28 14 input ce;
29 15 input [WIDTH-1:0] data_load;
30 16 output reg [WIDTH-1:0] count_out;
31 17 output max_count;
32 18 output zero;
33 19
```

```

51 Branch Coverage:
52   Enabled Coverage      Active      Hits      Misses % Covered
53   -----
54   Branches              10        10         0    100.0
55
56 =====Branch Details=====
57
58 Branch Coverage for file counter.v --
59
60 -----IF Branch-----
61   21                      1000      Count coming in to IF
62   21          1           7          if (!rst_n)
63   23          1          707        else if (!load_n)
64   25          1          196        else if (ce)
65                                     90      All False Count
66 Branch totals: 4 hits of 4 branches = 100.0%
67
68 -----IF Branch-----
69   26                      196      Count coming in to IF
70   26          1          116        if (up_down)
71   28          1           80        else
72 Branch totals: 2 hits of 2 branches = 100.0%
73
74 -----IF Branch-----
75   32                      906      Count coming in to IF
76   32          1           5      assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
77   32          2          901      assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
78 Branch totals: 2 hits of 2 branches = 100.0%
79
80 -----IF Branch-----
81   33                      906      Count coming in to IF
82   33          1           11      assign zero = (count_out == 0)? 1:0;
83   33          2          895      assign zero = (count_out == 0)? 1:0;
84 Branch totals: 2 hits of 2 branches = 100.0%
85

```

```

101 Toggle Coverage:
102   Enabled Coverage      Active      Hits      Misses % Covered
103   -----
104   Toggle Bins           46        46         0    100.0
105
106 =====Toggle Details=====
107
108 Toggle Coverage for File counter.v --
109
110   Line      Node      1H->0L      0L->1H      "Coverage"
111   -----
112   10      clk      1          1          100.00
113   11      rst_n      1          1          100.00
114   12      load_n      1          1          100.00
115   13      up_down      1          1          100.00
116   14      ce      1          1          100.00
117   15      data_load[7]      1          1          100.00
118   15      data_load[6]      1          1          100.00
119   15      data_load[5]      1          1          100.00
120   15      data_load[4]      1          1          100.00
121   15      data_load[3]      1          1          100.00
122   15      data_load[2]      1          1          100.00
123   15      data_load[1]      1          1          100.00
124   15      data_load[0]      1          1          100.00
125   16      count_out[7]      1          1          100.00
126   16      count_out[6]      1          1          100.00
127   16      count_out[5]      1          1          100.00
128   16      count_out[4]      1          1          100.00
129   16      count_out[3]      1          1          100.00
130   16      count_out[2]      1          1          100.00
131   16      count_out[1]      1          1          100.00
132   16      count_out[0]      1          1          100.00
133   17      max_count      1          1          100.00
134   18      zero      1          1          100.00
135

```

13.function Coverage

```

547         bin_max_zero                3143           1      Covered
548     Coverpoint count_out_cp3         100.0%        100      Covered
549         covered/total bins:           16          16
550         missing/total bins:           0           16
551         % Hit:                        100.0%        100
552         bin auto[0]                   24240          1      Covered
553         bin auto[1]                   21882          1      Covered
554         bin auto[2]                   21698          1      Covered
555         bin auto[3]                   21219          1      Covered
556         bin auto[4]                   21348          1      Covered
557         bin auto[5]                   21608          1      Covered
558         bin auto[6]                   21505          1      Covered
559         bin auto[7]                   21246          1      Covered
560         bin auto[8]                   21160          1      Covered
561         bin auto[9]                   21547          1      Covered
562         bin auto[10]                  21228          1      Covered
563         bin auto[11]                  21532          1      Covered
564         bin auto[12]                  21783          1      Covered
565         bin auto[13]                  21157          1      Covered
566         bin auto[14]                  21535          1      Covered
567         bin auto[15]                  21905          1      Covered
568     Coverpoint count_out_cp4         100.0%        100      Covered
569         covered/total bins:           1           1
570         missing/total bins:           0           1
571         % Hit:                        100.0%        100
572         bin_zero_max                  3320           1      Covered
573
574     TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1
575

```

14.assertion coverage

DIRECTIVE COVERAGE:						
Name	Design Unit	Design UnitType	Lang	File(Line)	Count	Status
/TOP_counter/DUT/sva/dollar1_cover	sva_t	Verilog	SVA	sva_t.sv(9)	672	Covered
/TOP_counter/DUT/sva/dollar2_cover	sva_t	Verilog	SVA	sva_t.sv(19)	80	Covered
/TOP_counter/DUT/sva/dollar3_cover	sva_t	Verilog	SVA	sva_t.sv(29)	129	Covered
/TOP_counter/DUT/sva/dollar4_cover	sva_t	Verilog	SVA	sva_t.sv(38)	93	Covered
/TOP_counter/DUT/sva/dollar5_cover	sva_t	Verilog	SVA	sva_t.sv(45)	13	Covered
/TOP_counter/DUT/sva/dollar6_cover	sva_t	Verilog	SVA	sva_t.sv(52)	53	Covered
/TOP_counter/DUT/sva/dollar7_cover	sva_t	Verilog	SVA	sva_t.sv(59)	110	Covered
TOTAL DIRECTIVE COVERAGE: 100.0% COVERS: 7						