

## Assignment 3

Q1

CODE TB :

alu\_tb.sv

```
1  import testing_pkg ::*;
2  module alu_tb ();
3      byte operand1, operand2, out;
4      bit clk, rst;
5      opcode_e opcode;
6      byte out_ex;
7
8      alu_seq o (.*);
9
10     transaction test1 = new;
11
12     int error_count = 0;
13     int correct_count = 0;
14
15     initial begin
16         clk = 0;
17         forever begin
18             #1 clk = ~clk;
19             test1.clk = clk;
20         end
21     end
22
23     initial begin
24         cheack_reset;
25
26         repeat (50000) begin
27             assert(test1.randomize);
28             opcode = test1.opcode;
29             operand1 = test1.operand1;
30             operand2 = test1.operand2;
31             @(negedge clk);
32             cheack_result;
33         end
34
35         cheack_reset;
36
37
38
39         $display("error = %d correct = %d", error_count, correct_count);
40         $stop;
41     end
42 end
```

```

42
43 task cheack_result ;
44     golden_model ;
45     if (rst !== 1) begin
46         @(negedge clk);
47         if (out_ex !== out)
48             error_count++;
49         else
50             correct_count++;
51     end
52     else begin
53         cheack_result_rst;
54     end
55 endtask
56
57 task golden_model;
58     if (rst)
59         out_ex <= 0;
60     else
61         case (opcode)
62             ADD: out_ex <= operand1 + operand2;
63             SUB: out_ex <= operand1 - operand2;
64             MULT: out_ex <= operand1 * operand2;
65             DIV: out_ex <= operand1 / operand2;
66             default: out_ex <= 0;
67         endcase
68     endtask
69
70 task cheack_reset ;
71     rst = 1;
72     cheack_result_rst;
73     rst = 0;
74
75 endtask
76
77 task cheack_result_rst ;
78     @(negedge clk);
79     if (out !== 0)
80         error_count++;
81
82     else
83         correct_count++;
84
85 endtask
86
87 endmodule

```

## CODE PKG :

```
1 package testing_pkg;
2
3 typedef enum {ADD,SUB,MULT,DIV,OR} opcode_e;
4
5 class transaction;
6     rand opcode_e opcode;
7     rand byte operand1;
8     rand byte operand2;
9     bit clk;
10
11     covergroup covcode @(posedge clk);
12         opcode_label : coverpoint opcode {
13             bins add_sub = {ADD,SUB};
14             bins add_sub = (ADD => SUB);
15             illegal_bins div = {DIV};
16         }
17         operand_label : coverpoint operand1 {
18             bins maxneg = {-128};
19             bins zero = {0};
20             bins maxpos = {127};
21             bins opbin = default;
22         }
23         a : cross opcode_label , operand_label
24             { bins add_sub_max = binsof(opcode_label.add_sub) && binsof(operand_label.maxneg);
25               bins add_sub_neg = binsof(opcode_label.add_sub) && binsof(operand_label.maxpos);
26               option.weight=5;
27             //option.cross_auto_bin_max=0;
28         }
29     endgroup
30
31     function new();
32         covcode = new();
33     endfunction
34
35 endclass //className
36
37
38 endpackage
39
40 endpackage
```

## DO FILE :

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

## DISPLAY :

```
Time: 199961 ns Iteration: 0 Region: /testing_pkg::transa
error = 0 correct = 50002
** Note: $stop : alu_tb.sv(40)
Time: 200004 ns Iteration: 1 Instance: /alu_tb
Break in Module alu_tb at alu_tb.sv line 40
```

## COVERGROUP :

Covergroups							
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_i
/testing_pkg/trans...		100.00%					
TYPE covcode		100.00%	100	100.00...		✓	
CVP covco...		100.00%	100	100.00...		✓	
CVP covco...		100.00%	100	100.00...		✓	
CROSS co...		100.00%	100	100.00...		✓	
INST Vtes...		100.00%	100	100.00...		✓	
CVP op...		100.00%	100	100.00...		✓	
illeg...		2021	-	-		✓	
bin ...		39797	1	100.00...		✓	
bin ...		2021	1	100.00...		✓	
CVP op...		100.00%	100	100.00...		✓	
bin ...		378	1	100.00...		✓	
bin ...		383	1	100.00...		✓	
bin ...		382	1	100.00...		✓	
def...		98859	-	-		✓	
CROSS...		100.00%	100	100.00...		✓	
bin ...		136	1	100.00...		✓	
bin ...		134	1	100.00...		✓	
bin ...		5	1	100.00...		✓	
bin ...		6	1	100.00...		✓	
bin ...		4	1	100.00...		✓	
bin ...		165	1	100.00...		✓	

COVERAGR :

=====				
Branch Coverage:				
Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Branches	7	7	0	100.00%

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

Toggle Coverage:				
Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Toggles	57	57	0	100.00%

Q2

Verification plan :

1	label	description	stimulus generation	functionality cheack	functional coverage
2	counter_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
3	counter_2	when load asserted the data store in data_load	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
4	counter_3	when the rst is asserted the output dout value must be low	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
5	counter_4	when ce is enable you can up or down the data	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	
6	counter_5	verifying randomize and constraint input	directed at the start of the simulation	cheacker in the testbench to make sure the output is correct	cover all value for input and output

CODE TB :

counter\_tb.sv

```
1  import counter_pkg::*;
2  module counter_tb ();
3
4      parameter WIDTH = 4;
5      logic clk;
6      logic rst_n;
7      logic load_n;
8      logic up_down;
9      logic ce;
10     logic [WIDTH-1:0] data_load;
11     logic [WIDTH-1:0] count_out;
12     logic max_count;
13     logic zero;
14
15     logic [WIDTH-1:0] count_out_ex;
16     int error_counter = 0;
17     int correct_counter = 0;
18
19     counter #(.WIDTH(WIDTH)) g (.*);
20
21     counter_class counter1 = new;
22
23     initial begin
24         clk = 0;
25         forever begin
26             #1 clk = ~clk;
27             counter1.clk = clk;
28         end
29     end
30
31     task cheack_result (input [WIDTH-1:0] x);
32
33         @(negedge clk);
34         if (x != count_out) begin
35             $display("%t error", $time);
36             error_counter++;
37         end
38         else
39             correct_counter++;
40
41     endtask
42
43     task cheack_reset ;
44         rst_n = 0;
45         @(negedge clk);
46         cheack_result (0);
47         rst_n = 1;
48
```

```

51     task cheack_max ;
52         @(negedge clk);
53         if (max_count)
54             correct_counter++;
55         else begin
56             $display("%t error",$time);
57             correct_counter++;
58         end
59     endtask
60
61     task cheack_zero ;
62         @(negedge clk);
63         if (zero)
64             correct_counter++;
65         else begin
66             $display("%t error",$time);
67             correct_counter++;
68         end
69     endtask
70
71     initial begin
72         ///label1/////
73         cheack_reset;
74         /////label2/////
75         load_n = 0;
76         data_load = 4'b0001;
77         cheack_result (data_load);
78         load_n = 1;
79         ///label3/////
80         cheack_reset;
81         /////label4/////
82         ce = 1;
83         up_down = 0;
84         cheack_result (count_out-1);
85         up_down = 1;
86         cheack_result(count_out +1);
87         /////label5/////
88         ce = 0;
89         load_n = 0;
90         data_load = 4'b0000;
91         cheack_result (data_load);
92         cheack_zero ;
93         load_n = 1;

```



```

93     load_n = 1;
94     //label6////
95     ce = 1;
96     load_n = 0;
97     data_load = 4'b0000;
98     cheack_result (data_load);
99     cheack_zero ;
100    load_n = 1;
101    //label7/////
102    up_down = 0;
103    cheack_result (count_out-1);
104    up_down = 1;
105    cheack_result(count_out +1);
106    //label8////
107    load_n = 0;
108    data_load = 4'b0100;
109    cheack_result (data_load);
110    load_n = 1;
111    //label9/////
112    up_down = 0;
113    cheack_result (count_out-1);
114    up_down = 1;
115    cheack_result(count_out +1);
116    //label10////
117    load_n = 0;
118    data_load = 4'b1111;
119    cheack_result (data_load);
120    cheack_max ;
121    load_n = 1;
122    //label11////
123    load_n = 0;
124    data_load = 4'b0010;
125    cheack_result (data_load);
126    load_n = 1;
127    //label12////
128    load_n = 0;
129    data_load = 4'b1100;
130    cheack_result (data_load);
131    load_n = 1;
132    //label13/////
133    cheack_reset;
134
135    /// 14
136

```

```

136
137     repeat (9000) begin
138         assert (counter1.randomize());
139         rst_n = counter1.reset;
140         load_n = counter1.load_n;
141         ce = counter1.ce;
142         data_load = counter1.data_load;
143         up_down = counter1.up_down;
144         counter1.count_out = count_out;
145         golden_model ;
146         cheack_out ;
147     end
148 end
149
150 $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
151 $stop;
152 end
153
154 task golden_model ;
155     if (!rst_n)
156         count_out_ex <= 0;
157     else if (!load_n)
158         count_out_ex <= data_load;
159     else if (ce)
160         if (up_down)
161             count_out_ex <= count_out_ex + 1;
162         else
163             count_out_ex <= count_out_ex - 1;
164     endtask
165
166 task cheack_out ;
167
168     @(negedge clk);
169     if (count_out_ex != count_out) begin
170         $display("%t error", $time);
171         error_counter++;
172     end
173     else
174         correct_counter++;
175 endtask
176
177 endmodule
178

```

CODE PKG :

```

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

```

DO FILE :

```

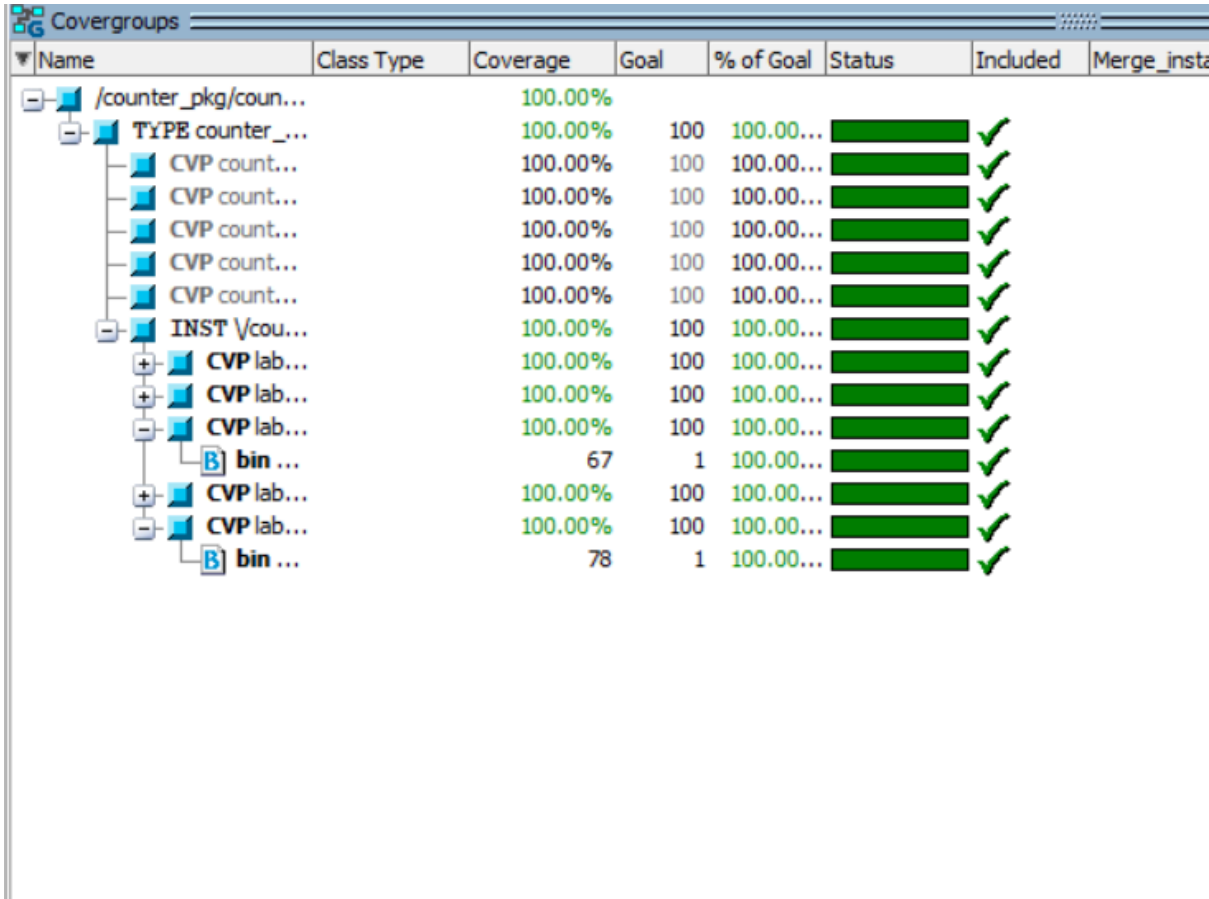
1  counter.do
2  vlib work
3  vlog counter.v counter_tb.sv counter_pkg.sv +cover -covercells
4  vsim -voptargs=+acc work.counter_tb -cover
5  add wave *
6  coverage save counter_tb.ucdb -onexit
7  run -all

```

DISPLAY :

```
# Loading work.counter(fast)
# error_counter =          0  correct_counter =          9019
# ** Note: $stop      : counter_tb.sv(151)
#   Time: 18044 ns  Iteration: 1  Instance: /counter_tb
# Break in Module counter tb at counter tb.sv line 151
```

COVERGROUP :



CODE COVERAGE :

Branch Coverage:				
Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Branches	10	10	0	100.00%

Condition Coverage:				
Enabled Coverage	Bins	Covered	Misses	Coverage
-----	----	----	-----	-----
Conditions	2	2	0	100.00%

### Statement Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Statements	7	7	0	100.00%

### Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Toggles	30	30	0	100.00%

## Q3

### Verification plan :

label	description	stimulus generation	functionality check	functional coverage
ALSU_1	when the rst is asserted the output dout value must be low	directed at the start of the simulation	checker in the testbench to make sure the output is correct	
ALSU_2	turn off constraintB and randomize the input and check output	directed at the start of the simulation	checker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_3	when the rst is asserted the output dout value must be low	directed at the start of the simulation	checker in the testbench to make sure the output is correct	
ALSU_4	turn on constraintB and turn off all constraint randomize the input and check output	directed at the start of the simulation	checker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_5	when the rst is asserted the output dout value must be low	directed at the start of the simulation	checker in the testbench to make sure the output is correct	cover all value for input and output
ALSU_6	make opcode equal zero and increment	directed at the start of the simulation	checker in the testbench to make sure the output is correct	cover bins transition
ALSU_7	make opcode equal valid array and randomize with in line constraint	directed at the start of the simulation	checker in the testbench to make sure the output is correct	

## CODE DESIGN :

```
1 module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
2 parameter INPUT_PRIORITY = "A";
3 parameter FULL_ADDER = "ON";
4 input clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
5 input [2:0] opcode;
6 input signed [2:0] A, B;
7 output reg [15:0] leds;
8 output reg signed [5:0] out;
9
10 reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
11 reg signed cin_reg;
12 reg [2:0] opcode_reg;
13 reg signed [2:0] A_reg, B_reg;
14
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
```

```

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 //ALU 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)
77                 3'h0: begin
78                     if (red_op_A_reg && red_op_B_reg)
79                         out <= (INPUT_PRIORITY == "A")? /A_reg: /B_reg;
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;
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;

```

```

94         else
95             out <= A_reg ^ B_reg;
96         end
97     3'h2: begin
98         if (FULL_ADDER == "ON")
99             out <= A_reg+B_reg+cin_reg;
100         else if (FULL_ADDER == "OFF")
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     endcase
117 end
118 end
119 end
120
121 endmodule
122

```

# GOLDEN MODEL :

```
V ALSU_golden_model.v
1  module ALSU_golden_model #( parameter INPUT_PRIORITY = "A",
2      parameter FULL_ADDER = "ON")
3      ( output reg [5:0] out,
4        output reg [15:0] leds,
5        input [2:0] A,B,opcode,
6        input clk,rst,cin,serial_in,red_op_A,red_op_B,bypass_A,bypass_B,direction);
7
8  reg red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
9  reg signed cin_reg;
10 reg [2:0] opcode_reg;
11 reg signed [2:0] A_reg, B_reg;
12
13 wire invalid_red_op, invalid_opcode, invalid;
14
15 assign invalid_red_op = (red_op_A_reg / red_op_B_reg) & (opcode_reg[1] / opcode_reg[2]);
16 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
17 assign invalid = invalid_red_op / invalid_opcode;
18
19 always @(posedge clk or posedge rst) begin
20     if(rst) begin
21         cin_reg <= 0;
22         red_op_B_reg <= 0;
23         red_op_A_reg <= 0;
24         bypass_B_reg <= 0;
25         bypass_A_reg <= 0;
26         direction_reg <= 0;
27         serial_in_reg <= 0;
28         opcode_reg <= 0;
29         A_reg <= 0;
30         B_reg <= 0;
31     end else begin
32         cin_reg <= cin;
33         red_op_B_reg <= red_op_B;
34         red_op_A_reg <= red_op_A;
35         bypass_B_reg <= bypass_B;
36         bypass_A_reg <= bypass_A;
37         direction_reg <= direction;
38         serial_in_reg <= serial_in;
39         opcode_reg <= opcode;
40         A_reg <= A;
41         B_reg <= B;
42     end
43 end
44
```



```

45 always @(posedge clk or posedge rst) begin
46     if(rst) begin
47         leds <= 0;
48     end else begin
49         if (invalid)
50             leds <= ~leds;
51         else
52             leds <= 0;
53     end
54 end
55
56 always @(posedge clk , posedge rst) begin
57     if (rst) begin
58         out <= 0;
59     end
60     else begin
61         if (bypass_A_reg && bypass_B_reg)
62             out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
63         else if (bypass_A_reg)
64             out <= A_reg;
65         else if (bypass_B_reg)
66             out <= B_reg;
67         else if (invalid)
68             out <= 0;
69         else begin
70
71             case (opcode)
72                 3'b000 : begin
73                     if (INPUT_PRIORITY == "A") begin
74                         if (red_op_A_reg)
75                             out <= /A_reg;
76                         else
77                             out <= A_reg/B_reg;
78                     end
79                     else if (INPUT_PRIORITY == "B") begin
80                         if (red_op_B_reg)
81                             out <= /B_reg;
82                         else
83                             out <= A_reg/B_reg;
84                     end
85                 end

```

```

86         3'b001 : begin
87             if (red_op_A_reg && red_op_B_reg)
88                 out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
89             else if (red_op_A_reg)
90                 out <= ^A_reg;
91             else if (red_op_B_reg)
92                 out <= ^B_reg;
93             else
94                 out <= A_reg ^ B_reg;
95         end
96         3'b010 : begin
97             if (FULL_ADDER == "ON")
98                 out <= A_reg + B_reg + cin_reg;
99             else if (FULL_ADDER == "OFF")
100                 out <= A_reg + B_reg;
101         end
102         3'b011 : out <= A_reg * B_reg;
103         3'b100 : begin
104             if (direction_reg)
105                 out <= {out[4:0],serial_in_reg};
106             else
107                 out <= {serial_in_reg,out[5:1]};
108         end
109         3'b101 : begin
110             if (direction_reg)
111                 out <= {out[4:0],out[5]};
112             else
113                 out <= {out[0],out[5:1]};
114         end
115     endcase
116 end
117 end
118 end
119
120 endmodule
121

```

# CODE TB :

```
1  ALSU_tb.sv
2
3  import ALSU_pkg::*;
4  module ALSU_tb ();
5      parameter INPUT_PRIORITY = "A";
6      parameter FULL_ADDER = "ON";
7      logic clk, cin, rst, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
8      logic [2:0] opcode;
9      logic signed [2:0] A, B;
10     logic [15:0] leds;
11     bit [15:0] leds_ex;
12     logic [5:0] out;
13     bit [5:0] out_ex;
14
15     int error_counter = 0;
16     int correct_counter = 0;
17
18     ALSU o1 (A,B,cin,serial_in,red_op_A,red_op_B,opcode,bypass_A,bypass_B,clk,rst,direction,leds,out);
19     ALSU_golden_model 04 (out_ex,leds_ex,A,B,opcode,clk,rst,cin,serial_in,red_op_A,red_op_B,bypass_A,bypass_B,direction);
20
21     ALU_class MY_ALU = new;
22
23     covergroup ALSU_G @(posedge clk);
24
25     label_A : coverpoint A {
26         bins A_data_0 = {0};
27         bins A_data_max = {MAXPOS};
28         bins A_data_min = {MAXNEG};
29         bins A_data_default = default;
30
31     label_A_red_op : coverpoint A iff (red_op_A) {
32         bins A_data_walking_ones[] = {1,2,4};
33
34     label_B : coverpoint B {
35         bins B_data_0 = {0};
36         bins B_data_max = {MAXPOS};
37         bins B_data_min = {MAXNEG};
38         bins B_data_default = default;
39
40     label_B_red_op : coverpoint B iff (red_op_B) {
41         bins B_data_walking_ones[] = {1,2,4};
42
43     label_opcode : coverpoint opcode {
44         bins bin_shift[] = {SHIFT,ROTATE};
45         bins bin_arith[] = {ADD,MULT};
46         bins bin_bitwise[] = {OR,XOR};
47         illegal_bins bin_invalid[] = {invalid6,invalid7};
48         bins opcode_tr = {0 => 1 => 2 => 3 => 4 => 5};
49     }
```

```

40         label_6_red_op = coverpoint B {j} (red_op_B);
52     ALU_6 my_alu_g;
53
54     initial begin
55         my_alu_g = new;
56     end
57
58     always @(posedge clk) begin
59         if (!rst && !bypass_A && !bypass_B) begin
60             my_alu_g.sample();
61         end
62     end
63
64
65     initial begin
66         clk = 0;
67         forever begin
68             #1 clk = ~clk;
69         end
70     end
71
72     initial begin
73         //1
74         cheack_reset;
75
76         //2
77         MY_ALU.OP_cns.constraint_mode(0);
78         repeat [90000] begin
79             assert (MY_ALU.randomize());
80             rst = MY_ALU.rst;
81             red_op_A = MY_ALU.red_op_A;
82             red_op_B = MY_ALU.red_op_B;
83             bypass_A = MY_ALU.bypass_A;
84             bypass_B = MY_ALU.bypass_B;
85             direction = MY_ALU.direction;
86             cin = MY_ALU.cin;
87             serial_in = MY_ALU.serial;
88             A = MY_ALU.A;
89             B = MY_ALU.B;
90             opcode = MY_ALU.my_opcode;
91             cheack_result;
92         end
93
94         //3
95         cheack_reset;
96

```

```

95         cheack_reset;
96
97         //4
98         MY_ALU.OP_cns.constraint_mode(1);
99         MY_ALU.ALU_cns.constraint_mode(0);
100         rst = 0;
101         red_op_A = 0;
102         red_op_B = 0;
103         bypass_A = 0;
104         bypass_B = 0;
105         repeat (900000) begin
106             assert (MY_ALU.randomize());
107             cheack_result ;
108         end
109
110         //5
111         cheack_reset;
112         //
113         opcode = 0;
114         repeat(6) begin
115             @(negedge clk)
116                 opcode++;
117         end
118
119         //
120         foreach (MY_ALU.op[i]) begin
121             opcode = MY_ALU.op[i];
122             repeat (100) begin
123                 assert (MY_ALU.randomize() with { my_opcode == opcode; });
124                 cheack_result;
125             end
126         end
127
128
129
130         $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
131         $stop;
132     end
133

```

```

133
134     task cheack_result ;
135         @(negedge clk);
136         if ((out_ex != out) || (leds_ex != leds)) begin
137             $display("%t error",$time);
138             error_counter++;
139         end
140         else
141             correct_counter++;
142     endtask
143
144
145     task cheack_reset ;
146         rst = 0;
147         @(negedge clk);
148         cheack_result ;
149         rst = 1;
150     endtask
151
152
153
154 endmodule
155
156

```

CODE PKG :

```

1  package ALSU_pkg;
2
3  typedef enum bit [2:0] { OR,XOR,ADD,MULT,SHIFT,ROTATE,invalid6,invalid7 } opcode_e;
4
5  localparam MAXPOS = 3, ZERO = 0, MAXNEG = -4;
6
7  class ALU_class ;
8      rand logic rst;
9      rand logic signed [2:0] A,B;
10     rand logic red_op_A,red_op_B;
11     rand opcode_e my_opcode;
12     rand logic bypass_A, bypass_B;
13     rand logic direction,cin;
14     rand logic serial;
15     rand opcode_e op[6];
16
17     constraint ALU_cns {
18         //Label1
19         rst dist { 0:=99 , 1:=1};
20         //Label2
21         (my_opcode == ADD || my_opcode == MULT) -> A dist { MAXNEG,MAXPOS,ZERO :/ 60};
22         (my_opcode == ADD || my_opcode == MULT) -> B dist { MAXNEG,MAXPOS,ZERO :/ 60};
23         //Label3
24         (my_opcode == OR || my_opcode == XOR && red_op_A ==1) -> A dist { 1,2,4 :/ 80};
25         (my_opcode == OR || my_opcode == XOR && red_op_A ==1) -> B dist { 0 := 100};
26         //Label4
27         (my_opcode == OR || my_opcode == XOR && red_op_B ==1) -> B dist { 1,2,4 :/ 80};
28         (my_opcode == OR || my_opcode == XOR && red_op_B ==1) -> A dist { 0 := 100};
29         //Label5
30         my_opcode dist { [invalid6:invalid7] := 20,[OR:ROTATE] := 80};
31         //Label6
32         bypass_A dist { 0:=80 , 1:=20};
33         bypass_B dist { 0:=80 , 1:=20};
34     }
35     constraint OP_cns {
36         foreach (op[i])
37             op[i] inside {OR,XOR,ADD,MULT,SHIFT,ROTATE};
38     }
39
40 endclass

```

DO FILE :

```

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

```

DISPLAY :

```

# Time: 179985 ns Iteration: 0 Region: /ALSU_tb/ALSU_G
# error_counter = 0 correct_counter = 990603
# ** Note: $stop : ALSU_tb.sv(131)
# Time: 1981224 ns Iteration: 1 Instance: /ALSU_tb
# Break in Module ALSU_tb at ALSU_tb.sv line 131

```

## COVERGROUP :

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_
/ALSU_tb		100.00%					
TYPE ALSU_G		100.00%	100	100.00...	<div></div>	✓	
CVP ALSU_...		100.00%	100	100.00...	<div></div>	✓	
CVP ALSU_...		100.00%	100	100.00...	<div></div>	✓	
CVP ALSU_...		100.00%	100	100.00...	<div></div>	✓	
CVP ALSU_...		100.00%	100	100.00...	<div></div>	✓	
CVP ALSU_...		100.00%	100	100.00...	<div></div>	✓	
INST VAL...		100.00%	100	100.00...	<div></div>	✓	
CVP lab...		100.00%	100	100.00...	<div></div>	✓	
bin ...	B	1868362	1	100.00...	<div></div>	✓	
bin ...	B	11501	1	100.00...	<div></div>	✓	
bin ...	B	11441	1	100.00...	<div></div>	✓	
def...	B	56713	-	-		✓	
CVP lab...		100.00%	100	100.00...	<div></div>	✓	
bin ...	B	6327	1	100.00...	<div></div>	✓	
bin ...	B	6365	1	100.00...	<div></div>	✓	
CVP lab...		100.00%	100	100.00...	<div></div>	✓	
bin ...	B	1868056	1	100.00...	<div></div>	✓	
bin ...	B	11580	1	100.00...	<div></div>	✓	
bin ...	B	11246	1	100.00...	<div></div>	✓	
def...	B	57135	-	-		✓	
CVP lab...		100.00%	100	100.00...	<div></div>	✓	
bin ...	B	6446	1	100.00...	<div></div>	✓	
bin ...	B	6472	1	100.00...	<div></div>	✓	
CVP lab...		100.00%	100	100.00...	<div></div>	✓	
illeg...	B	5637	-	-		✓	

## CODE COVERAGE :



Toggle Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
-----	----	----	-----	-----
Toggles	118	118	0	100.00%

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

Toggle Coverage for instance /ALSU\_tb/o1 --

Node	1H->0L	0L->1H	"Coverage"
-----	-----	-----	-----
A[0-2]	1	1	100.00
A_reg[2-0]	1	1	100.00
B[0-2]	1	1	100.00
B_reg[2-0]	1	1	100.00
bypass_A	1	1	100.00
bypass_A_reg	1	1	100.00
bypass_B	1	1	100.00
bypass_B_reg	1	1	100.00
cin	1	1	100.00
cin_reg	1	1	100.00
clk	1	1	100.00
direction	1	1	100.00
direction_reg	1	1	100.00
invalid	1	1	100.00
invalid_opcode	1	1	100.00
invalid_red_op	1	1	100.00
leds[15-0]	1	1	100.00
opcode[0-2]	1	1	100.00
opcode_reg[2-0]	1	1	100.00
out[5-0]	1	1	100.00
red_op_A	1	1	100.00
red_op_A_reg	1	1	100.00
red_op_B	1	1	100.00
red_op_B_reg	1	1	100.00
rst	1	1	100.00
serial_in	1	1	100.00
serial_in_reg	1	1	100.00

## Q4

### Verification plan :

label	description	stimulus generation	functionality check
ram_1	call task stimulus gen and make write asserted	directed at the start of the simulation	checker in the testbench to make sure the output is correct
ram_2	make write disasserted and read asserted and call task golden model and cheack out	directed at the start of the simulation	checker in the testbench to make sure the output is correct
ram_3	call queue array and display data out	directed at the start of the simulation	checker in the testbench to make sure the output is correct

### DESIGN :

```
ram.sv
1  module my_mem(
2    input clk,
3    input write,
4    input read,
5    input [7:0] data_in,
6    input [15:0] address,
7    output reg [7:0] data_out
8  );
9    // Declare a 9-bit associative array using the logic data type & the key of int datatype
10   reg [7:0] mem_array [0:65535];
11   always @(posedge clk) begin
12     if (write)
13       mem_array[address] <= {~^data_in, data_in};
14     else if (read)
15       data_out <= mem_array[address];
16   end
17 endmodule
```

### CODE TB :

ram\_tb.sv

```
1  module ram_tb ();
2      localparam TESTS = 100;
3      logic clk;
4      logic write;
5      logic read;
6      logic [7:0] data_in;
7      logic [15:0] address;
8      logic [7:0] data_out;
9      logic [7:0] data_out_ex;
10
11     int address_array[TESTS];
12     int data_to_write_array[TESTS];
13     int data_read_expect_assoc [int];
14     int data_read_queue[$];
15
16     int error_counter = 0;
17     int correct_counter = 0;
18     int i = 0;
19
20     my_mem t (.*);
21
22     initial begin
23         clk = 0;
24         forever begin
25             #1 clk = ~clk;
26         end
27     end
28
29     task golden_model;
30         data_out_ex = data_read_expect_assoc[address_array[i]];
31     endtask
32
33     task stimulus_gen ;
34         for (i=0 ; i<TESTS ; i++) begin
35             address_array[i] = $urandom_range(0,65535);
36             data_to_write_array[i] = $urandom_range(0,255);
37             data_read_expect_assoc[address_array[i]] = {~^data_to_write_array[i], data_to_write_array[i]};
38         end
39     endtask
40
41     task check_result ;
42         @(negedge clk);
43         if (data_out_ex != data_out) begin
44             $display("%t error",$time);
45             error_counter++;
46         end
47         else
48             correct_counter++;
```

```

47     else
48         correct_counter++;
49
50     data_read_queue.push_back (data_out);
51
52     endtask
53
54     initial begin
55         write = 0;
56         read = 0;
57         data_in = 0;
58         address = 0;
59
60     stimulus_gen;
61
62     for (i=0 ; i<TESTS ; i++) begin
63         @(negedge clk);
64         address = address_array[i];
65         data_in = data_to_write_array[i];
66         read = 0;
67         write = 1;
68     end
69
70     @(negedge clk);
71     write = 0;
72
73     for (i=0 ; i<TESTS ; i++) begin
74         @(negedge clk);
75         address = address_array[i];
76         read = 1;
77         write = 0;
78         golden_model;
79         @(posedge clk);
80         cheack_result;
81     end
82
83     read = 0;
84
85     // display output
86
87     while (data_read_queue.size() > 0) begin
88         $display (" data = %0h ",data_read_queue.pop_front());
89     end
90
91     $display("error_counter = %d  correct_counter = %d",error_counter,correct_counter);
92     $stop;

```

DISPLAY :

```

# data = 80
# data = 5
# data = e0
# data = fd
# data = 2c
# error_counter =          0  correct_counter =          100
# ** Note: $stop      : ram_tb.sv(92)
#   Time: 602 ns  Iteration: 1  Instance: /ram_tb
# Break in Module ram_tb at ram_tb.sv line 92

```