ASSIGMENT 2

Example1

```
1   module arrays ();
2
3   int dyn_arr1[];
4   int dyn_arr2[]='{9,1,8,3,4,4};
5
6   initial begin
7       dyn_arr1=new[6];
8       foreach(dyn_arr1[i])
9           dyn_arr1[i]=i;
10      $display("%p %0d",dyn_arr1,dyn_arr1.size());
11      dyn_arr1.delete();
12
13      dyn_arr2.reverse();
14      $display("%p",dyn_arr2);
15      dyn_arr2.sort();
16      $display("%p",dyn_arr2);
17      dyn_arr2.rsort();
18      $display("%p",dyn_arr2);
19      dyn_arr2.shuffle();
20      $display("%p",dyn_arr2);
21  end
22
23  endmodule
24
```

```
VSIM 3> run -all
# '{0, 1, 2, 3, 4, 5} 6
# '{4, 4, 3, 8, 1, 9}
# '{1, 3, 4, 4, 8, 9}
# '{9, 8, 4, 4, 3, 1}
# '{8, 4, 9, 1, 3, 4}
```

Counter

Design

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

always @(posedge clk) begin
    if (!rst_n)
        count_out <= 0;
    else if (!load_n)
        count_out <= data_load;
    else if (ce)
        if (up_down)
            count_out <= count_out + 1;
        else
            count_out <= count_out - 1;
end

assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
assign zero = (count_out == 0)? 1:0;

endmodule
```

testbench

```
package all_item;
    parameter WIDTH=4;
    class transaction;

    rand bit rst_n,load_n,up_down,ce;
    rand bit [WIDTH-1:0] data_load;

    constraint x {
        rst_n dist {0:=5,1:=95};
        load_n dist {0:=70,1:=30};
        ce dist {1:=70,0:=30};
    }

    /*function printing;
        $display("count_out=%0d  , max_count=%0d, zero=%0d",count_out,max_count,zero);
    endfunction*/

endclass
endpackage

int error=0,correct =0;
import all_item::*;
```

```
module counter_tb();

logic clk=1,rst_n,load_n,up_down,ce;
logic  [WIDTH-1:0] data_load;
logic  [WIDTH-1:0] count_out;
logic max_count,zero;
bit[3:0] last_value=0;
counter co(.*);

always #5 clk=!clk;

transaction tr=new();

initial begin
    rst_n=0;
    test();
    repeat(100)begin
        assert (tr.randomize());
        rst_n=tr.rst_n;
        load_n=tr.load_n;
        up_down=tr.up_down;
        ce=tr.ce;
        data_load=tr.data_load ;
        test();
      // tr.printing();
    end
    $display("number of correct =%0d ,error=%0d",correct,error);
$finish;
end
```

```
task test();
   if (rst_n==0) begin
       if(count_out!=0 || zero!=1 || max_count!=0)begin
           $display( @%0t there is problem in reset ,$time);error++;
       end
       else correct++;
   end
   else begin
       if (load_n==0)begin //check on load case
           if(data_load==0  && (count_out!=0 || max_count!=0 || zero!=1))begin
               $display("@%0t there is problem in zero case ",$time); error++;
           end
           else if (data_load==15 && (count_out!=15 || max_count!=1 || zero!=0) )begin
                   $display("@%0t there is problem in max case ",$time);error++;
           end
           else if (count_out!=data_load )begin
                   $display("@%0t there is problem in loading ",$time);error++;
           end
           else  correct++;
       end
       else begin
           if (ce)begin
               if ((up_down==1 && count_out!=last_value+4'b0001) ||
                   (up_down==1 && last_value==14 && max_count!=1) || (up_down==1 && last_value==15 && zero!=1) )begin
                       $display("@%0t there is problem in count up ",$time);error++;
               end
               else if ((up_down==0 && count_out!=last_value-4'b0001) ||
                           (up_down==0 && last_value==0 && max_count!=1) || (up_down==0 && last_value==1 && zero!=1) )begin
                           $display("@%0t there is problem in count down and value =%0d",$time,(last_value-4'b0001));error++;
               end
               else correct++;
```

```
63      else begin
76          else begin
77              if (ce)begin
82                  else if ((up_down==0 && count_out!=last_value-4'b0001) ||
85                  end
86                  else correct++;
87              end
88              else
89                  if(!ce )begin
90                      if(count_out!=last_value)begin
91                          $display("@%0t there is problem in enaple ",$time);error++;
92                      end
93                      else
94                      begin
95                          correct++;
96                      end
97                  end
98          end
99      end
00
01      last_value=count_out;
02
03      #1;
04  endtask
05
06
07
08  endmodule
09
```

1	Label	Description	Stimulus Generation	Functionality Check
2	COUNTER_1	When the reset is asserted, the output counter value should be low then deassert reset	Directed at the start of the simulation then it randomized with cosntraint to be of high 95 % from time	A checker in the testbench to make sure the output is correct by test function
3	COUNTER_2	when load_n is asserted to low -> the output count out should take same value of load data	Randomization with constrain that Load_n should be low(active) for 70% of time,we can check this on time 75 ns	A checker in the testbench to make sure the output is correct by test function
4	COUNTER_3	when load_n is asserted to low and load data is equal to 15 -> the output count out should take 15 and max count output should be 1	Randomization,we can check this on time 495 ns	A checker in the testbench to make sure the output is correct by test function
5	COUNTER_4	when load_n is asserted to low and load data is equal to 0 -> the output count out should take 0 and zero output should be 1	Randomization,we can check this on time 535 ns	A checker in the testbench to make sure the output is correct by test function
6	COUNTER_5	we assert load_n to high and we assert enaple input to high and count up -> the output count _out shoult increase every clock cycle	Randomization we can check this in most of simulation	A checker in the testbench to make sure the output is correct by test function in entire simulation
7	COUNTER_6	we assert load_n to high and we assert enaple input to high and count down-> the output count _out shoult decrease every clock cycle	Randomization under constrain that ec should be HIgh(active) for 70% of time,	A checker in the testbench to make sure the output is correct by test function in entire simulation
8	COUNTER_7	we check on state when counter reach 15 through counting and check output max count if =1	Randomization	A checker in the testbench to make sure the output is correct by test function in entire simulation
9	COUNTER_8	we check on state when counter reach 0 through counting and check output Zero count if =1	Randomization,we can check this on time 790 ns	A checker in the testbench to make sure the output is correct by test function in entire simulation
10	COUNTER_9	we check when reset is high and load_n is high and enaple is low the output shouldnt change and keep previous value	Randomization through simmulation	A checker in the testbench to make sure the output is correct by test function in entire simulation

```
vlib work
vlog counter.v counter_tb.sv  +cover -covercells
vsim -voptargs=+acc work.counter_tb -cover
add wave *
coverage save counter_tb.ucdb -onexit -du work.counter
run -all
quit -sim

vcover report counter_tb.ucdb -details -all -output coverage_report.txt
```

```
# number of correct =101 ,error=0
# ** Note: $finish    : E:/study/kareem wassem/ASSIGMENTS/2/counter/counter_tb.sv(51)
#    Time: 1006 ns  Iteration: 0  Instance: /counter_tb
# 1
```

Coverage report

```
=== File: counter.v
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            7         7         0     100.0

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

Statement Coverage for file counter.v --

    1                                       ////////////////////////////////////////////////////////////////////////////////
    2                                       // Author: Kareem Waseem
    3                                       // Course: Digital Verification using SV & UVM
    4                                       //
    5                                       // Description: Counter Design
    6                                       //
    7                                       ////////////////////////////////////////////////////////////////////////////////
    8
    9                                       module counter (clk ,rst_n, load_n, up_down, ce, data_load, count_out, max_count, zero);
    10                                      parameter WIDTH = 4;
    11                                      input clk;
    12                                      input rst_n;
    13                                      input load_n;
    14                                      input up_down;
    15                                      input ce;
    16                                      input [WIDTH-1:0] data_load;
    17                                      output reg [WIDTH-1:0] count_out;
    18                                      output max_count;
    19                                      output zero;
    20
    21              1                  100  always @(posedge clk) begin
    22                                          if (!rst_n)
```

```
    22                                          if (!rst_n)
    23              1                    5          count_out <= 0;
    24                                          else if (!load_n)
    25              1                   74          count_out <= data_load;
    26                                          else if (ce)
    27                                              if (up_down)
    28              1                    7              count_out <= count_out + 1;
    29                                              else
    30              1                    6              count_out <= count_out - 1;
    31                                      end
    32
    33              1                   89      assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
    34              1                   89      assign zero = (count_out == 0)? 1:0;
    35
    36                                          endmodule
```

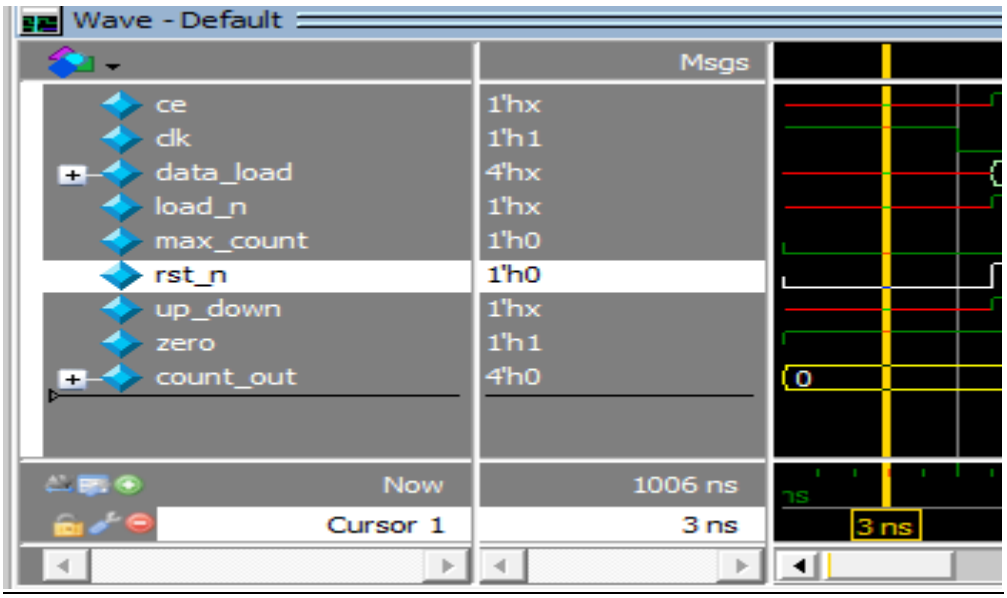
```
Branch Coverage:
    Enabled Coverage           Active      Hits    Misses % Covered
    ----------------           ------      ----    ------ ---------
    Branches                       10        10         0     100.0

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

Branch Coverage for file counter.v --

------------------------------------IF Branch------------------------------------
    22                                    100     Count coming in to IF
    22              1                       5         if (!rst_n)
    24              1                      74         else if (!load_n)
    26              1                      13         else if (ce)
                                            8     All False Count
Branch totals: 4 hits of 4 branches = 100.0%

------------------------------------IF Branch------------------------------------
    27                                     13     Count coming in to IF
    27              1                       7             if (up_down)
    29              1                       6             else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    33                                     88     Count coming in to IF
    33              1                       2     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
    33              2                      86     assign max_count = (count_out == {WIDTH{1'b1}})? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    34                                     88     Count coming in to IF
    34              1                      11     assign zero = (count_out == 0)? 1:0;
    34              2                      77     assign zero = (count_out == 0)? 1:0;
Branch totals: 2 hits of 2 branches = 100.0%
```

```
Condition Coverage:
    Enabled Coverage           Active   Covered    Misses % Covered
    ----------------           ------      ----    ------ ---------
    FEC Condition Terms             0         0         0     100.0
Expression Coverage:
    Enabled Coverage           Active   Covered    Misses % Covered
    ----------------           ------      ----    ------ ---------
    FEC Expression Terms            0         0         0     100.0
FSM Coverage:
    Enabled Coverage           Active      Hits    Misses % Covered
    ----------------           ------      ----    ------ ---------
    FSMs                                                      100.0
        States                      0         0         0     100.0
        Transitions                 0         0         0     100.0
```

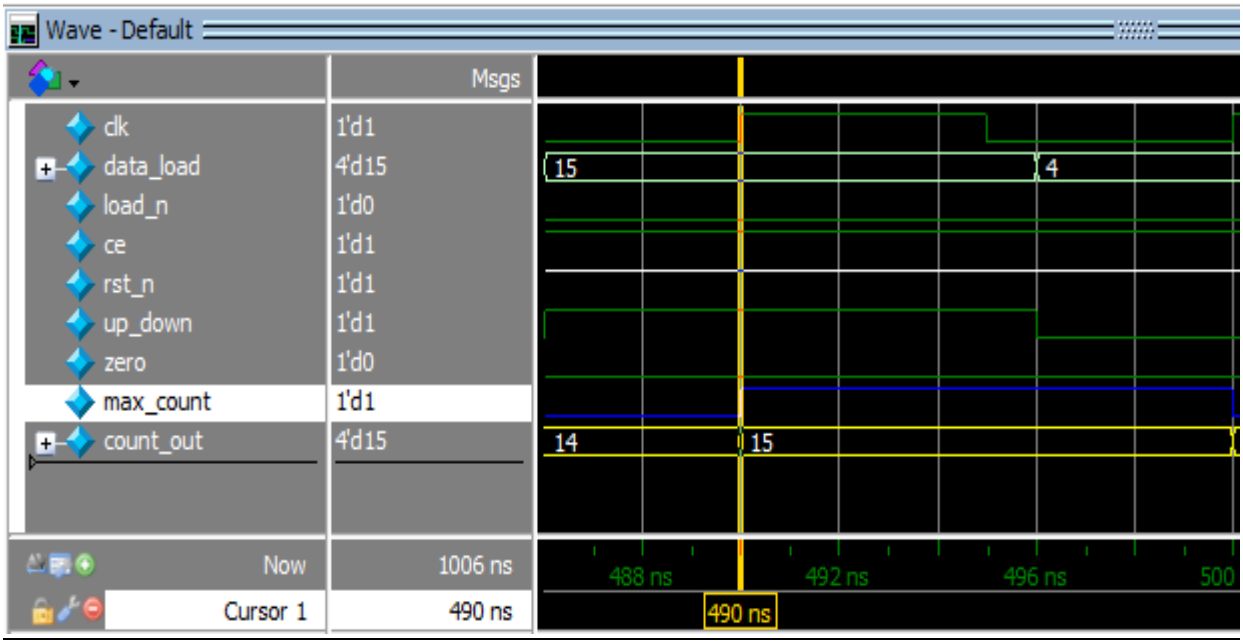
```
Toggle Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Toggle Bins                     30        30         0     100.0

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

Toggle Coverage for File counter.v --

       Line                                  Node      1H->0L      0L->1H  "Coverage"
--------------------------------------------------------------------------------------
         11                                   clk           1           1      100.00
         12                                 rst_n           1           1      100.00
         13                                load_n           1           1      100.00
         14                              up_down            1           1      100.00
         15                                    ce           1           1      100.00
         16                          data_load[3]           1           1      100.00
         16                          data_load[2]           1           1      100.00
         16                          data_load[1]           1           1      100.00
         16                          data_load[0]           1           1      100.00
         17                          count_out[3]           1           1      100.00
         17                          count_out[2]           1           1      100.00
         17                          count_out[1]           1           1      100.00
         17                          count_out[0]           1           1      100.00
         18                             max_count           1           1      100.00
         19                                  zero           1           1      100.00

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

Toggle Coverage      =      100.0% (30 of 30 bins)


Total Coverage By File (code coverage only, filtered view): 100.0%
```

Simulation

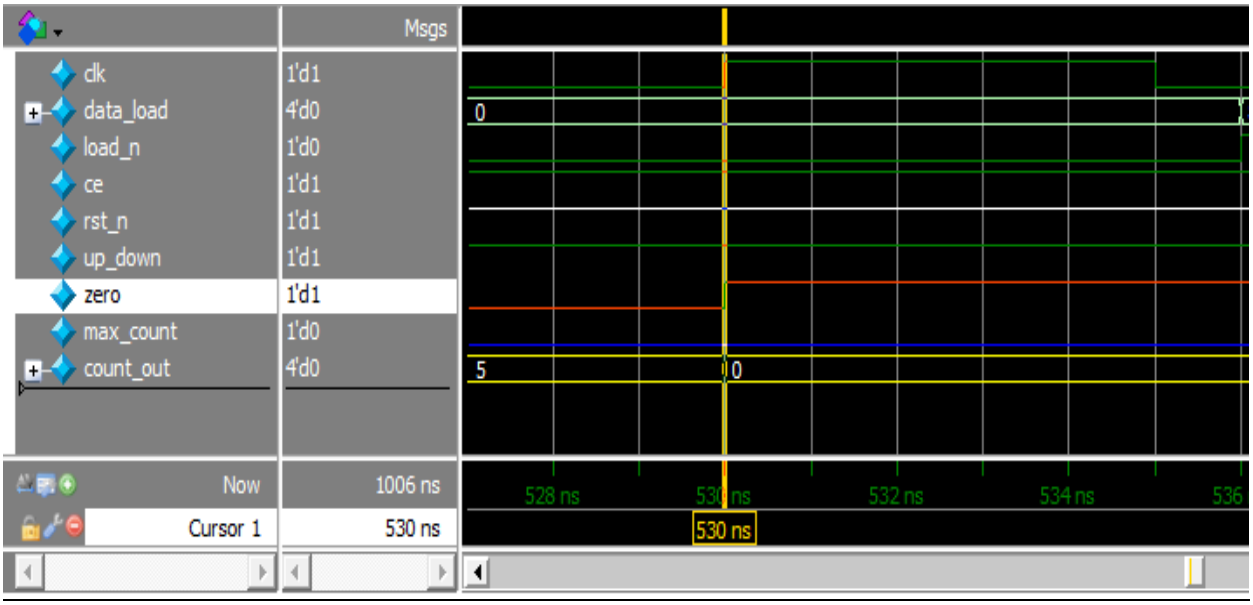
1-check on reset



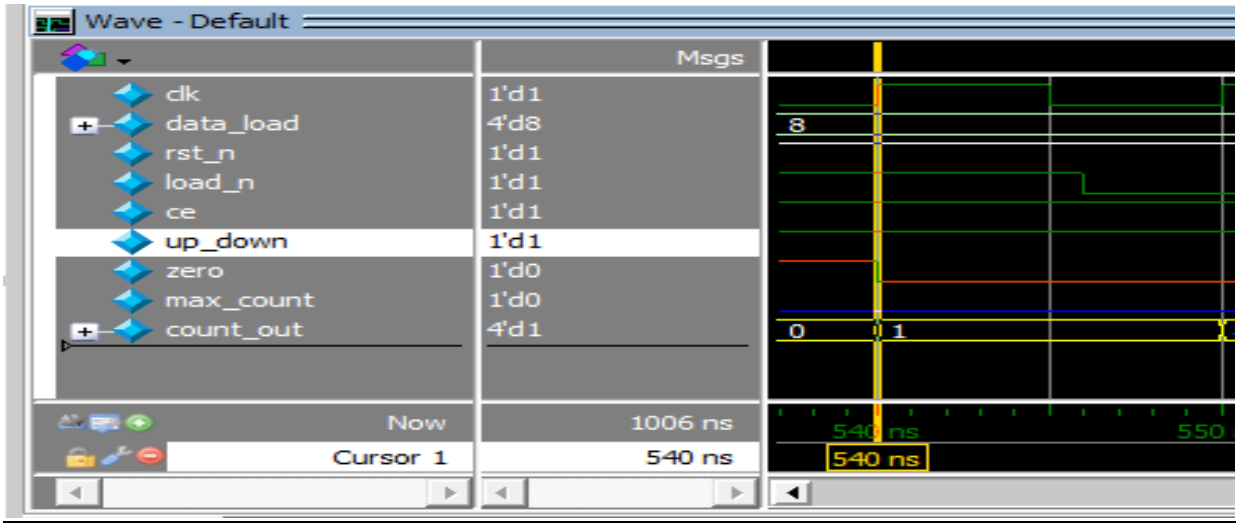
2-check on load with 0 and max count

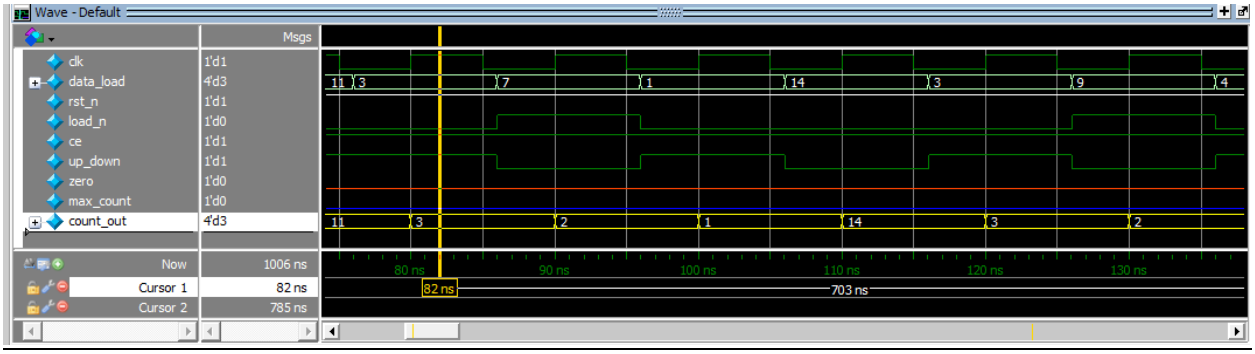


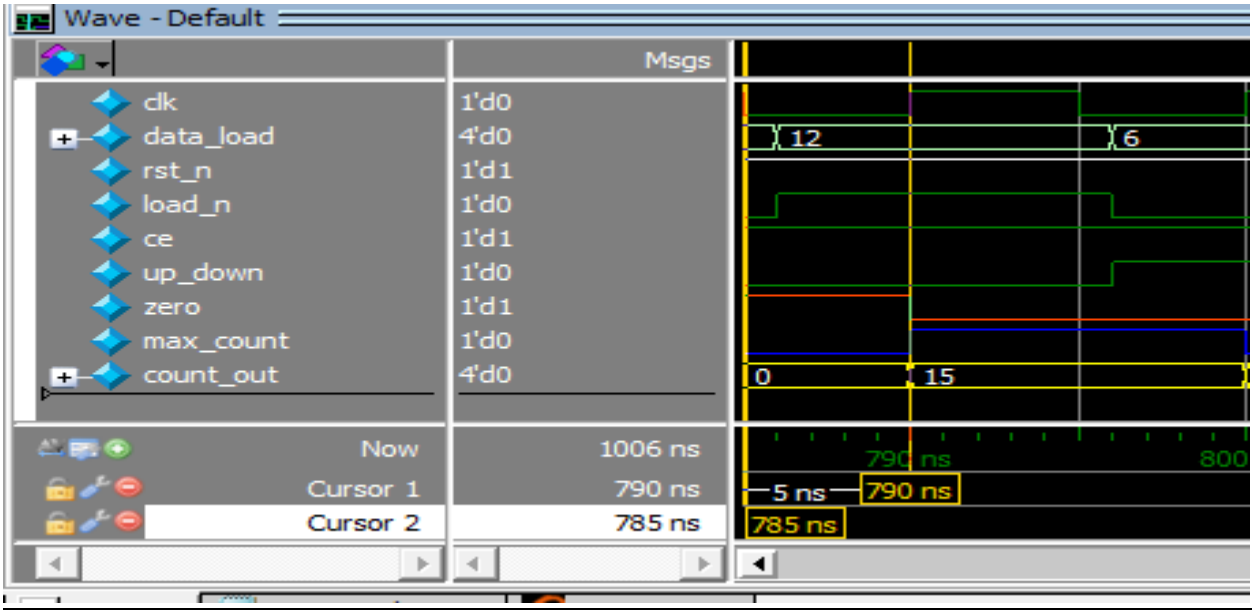
3-check on load with 15 and zero output



4-count up and down







ALSU

Design

Changes are made as comments on code

```
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input  clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input  [2:0] opcode;
input signed [2:0] A, B;
output reg signed[15:0] leds;
output reg signed[5:0] out;

reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg;
reg signed [2:0] A_reg, B_reg; //change to signed
reg signed[5:0] out_next;      //change to signed
wire invalid_red_op, invalid_opcode, invalid;

//Invalid handling
assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
assign invalid = invalid_red_op | invalid_opcode;
```

```
always @(posedge clk or posedge rst) begin
  if(rst) begin
     cin_reg <= 0;
     red_op_B_reg <= 0;
     red_op_A_reg <= 0;
     bypass_B_reg <= 0;
     bypass_A_reg <= 0;
     direction_reg <= 0;
     serial_in_reg <= 0;
     opcode_reg <= 0;
     A_reg <= 0;
     B_reg <= 0;
    out_next<=0; // we need a ff to keep value of output to be Zero
  end else begin
     cin_reg <= cin;
     red_op_B_reg <= red_op_B;
     red_op_A_reg <= red_op_A;
     bypass_B_reg <= bypass_B;
     bypass_A_reg <= bypass_A;
     direction_reg <= direction;
     serial_in_reg <= serial_in;
     opcode_reg <= opcode;
     A_reg <= A;
     B_reg <= B;
    out_next<=out; // we need a ff to keep value of output until serial in reg get its new value
  end
end
```

```
50  //leds output blinking
51  always @(posedge clk or posedge rst) begin
52    if(rst) begin
53      leds <= 0;
54    end else begin
55      if (invalid)
56        leds <= ~leds;
57      else
58        leds <= 0;
59    end
60  end
```

```
63  always @(posedge clk or posedge rst) begin
64
65    if(rst) begin
66      out <= 0;
67    end
68    else begin
69      if (bypass_A_reg && bypass_B_reg)
70        out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
71      else if (bypass_A_reg)
72        out <= A_reg;
73      else if (bypass_B_reg)
74        out <= B_reg;
75      else if (invalid) // cahnge the priority of invalid bits after bypass_reg
76          out <= 0;
77      else begin
78          case (opcode)
79            3'h0: begin //change Opcode to OR not AND
80              if (red_op_A_reg && red_op_B_reg)
81                out = (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
82              else if (red_op_A_reg)
83                out <= |A_reg;
84              else if (red_op_B_reg)
85                out <= |B_reg;
86              else
87                out <= A_reg | B_reg;
88            end
```

```
77      else begin
78          case (opcode)
79            3'h0: begin //change Opcode to OR not AND
80              if (red_op_A_reg && red_op_B_reg)
81                out = (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
82              else if (red_op_A_reg)
83                out <= |A_reg;
84              else if (red_op_B_reg)
85                out <= |B_reg;
86              else
87                out <= A_reg | B_reg;
88            end
89            3'h1: begin // change opcode to XOR not OR
90              if (red_op_A_reg && red_op_B_reg)
91                out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
92              else if (red_op_A_reg)
93                out <= ^A_reg;
94              else if (red_op_B_reg)
95                out <= ^B_reg;
96              else
97                out <= A_reg ^ B_reg;
98            end
99            3'h2: begin //here we add condition to check full adder if ON or OFF
100             if(FULL_ADDER == "ON")
101               out <= A_reg + B_reg+cin_reg;
102             else if(FULL_ADDER == "OFF")
103               out <= A_reg + B_reg;
104
105           end
106           3'h3: out <= A_reg * B_reg;
107           3'h4: begin
108
109             if (direction_reg)
110               out <= {out_next[4:0], serial_in_reg};
111             else
112               out <= {serial_in_reg, out_next[5:1]};
113           end
```

```
114             3'h5: begin
115               if (direction_reg)
116                 out <= {out_next[4:0], out_next[5]};
117               else
118                 out <= {out_next[0], out_next[5:1]};
119             end
120           endcase
121       end
122     end
123 end
124
125 endmodule
```

Testbench

```
package pack_alsu;
typedef enum  { OR=0,XOR,ADD,MULT,SHIFT,ROTATE,INVALID6,INVALID7 } Opcode_e;
parameter MAXPOS=7,MAXNEG=-8,ZERO=0;
class transaction;
    rand bit  clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    rand bit [2:0] opcode;
    rand bit signed [2:0] A, B;

    constraint trans {
        rst dist {1:=5 , 0:=95};
        if (opcode ==ADD || opcode== MULT){
            A dist {MAXPOS :=30 ,MAXNEG:=30,ZERO:=20,[MAXNEG+1:-1]:=5,[1:MAXPOS-1]:=5};
        }
        if ((opcode ==OR || opcode== XOR ) && red_op_A==1'b1){
            A dist {1:=20,2:=20,4:=20,3:=10,5:=10,6:=10,7:=10,0:=5};
            B==3'b000;
        }
        if ((opcode ==OR || opcode== XOR ) && red_op_B==1'b1){
            B dist {1:=20,2:=20,4:=20,3:=10,5:=10,6:=10,7:=10,0:=5};
            A==3'b000;
        }
        opcode dist {[0:5]:=30,6:=5,7:=5};
        bypass_A dist {0:=90,1:=10};
        bypass_B dist {0:=90,1:=10};
        if(opcode ==3'b000 ||opcode ==3'b001 ){
        red_op_A && red_op_B dist {1:=70,0:=30};
        }
    }
endclass
endpackage
```

```
import pack_alsu::*;

module ALSU_tb ();
parameter INPUT_PRIORITY = "B";
parameter FULL_ADDER = "ON";

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

Opcode_e kind;
int errors =0,correct=0;
bit invalid;
bit signed [5:0] last_out=0;

ALSU #(.INPUT_PRIORITY(INPUT_PRIORITY),.FULL_ADDER(FULL_ADDER)) tb (.*);


always #10 clk=!clk;
transaction tr=new();

```

```
always #10 clk=!clk;
transaction tr=new();


initial begin
    rst=1'b1;
    test();
    repeat(10000) begin
        assert(tr.randomize());
        opcode=tr.opcode;A=tr.A;B=tr.B;
        rst=tr.rst;cin=tr.cin; red_op_A=tr.red_op_A; red_op_B=tr.red_op_B; bypass_A=tr.bypass_A;
        bypass_B=tr.bypass_B; direction=tr.direction; serial_in=tr.serial_in ;
        test();
    end

     {rst,red_op_A, red_op_B, bypass_A, bypass_B}=5'b00000;
       opcode=3'h2;
       cin=1;
       A=3'b001;B=3'b001;
       test();
    $display("number of correct =%0d ,error=%0d",correct,errors);
$finish;

end
```

```
task  test();


    repeat (2) @(negedge clk);

     invalid=(tr.opcode==INVALID6 || tr.opcode == INVALID7)
          || ((red_op_A==1'b1 | red_op_B==1'b1) && (opcode!= OR | opcode!=XOR));

    // check on led
    if(invalid && leds!=leds)begin
        $display("@%0t you have problem in led toggle",$time); errors++;
    end
    else if(!invalid && leds!=0)begin
        $display("@%0t you have problem in led zero",$time);   errors++;
    end

    // check output -out
    if(rst)begin //check on reset
        if(rst && out !=0 && leds !=0)begin
            $display("@%0t you have problem in reset",$time); errors++;
        end
        else correct++;

    end
    else begin
```

```
    else begin
        // check on priority case on bypass
        if(bypass_A && bypass_B)begin
            if (bypass_A ==1 && bypass_B ==1 && out != A && INPUT_PRIORITY== "A") begin
                $display("@%0t you have problem priority A",$time); errors++;
            end
            else if (bypass_A ==1 && bypass_B ==1 && out != B && INPUT_PRIORITY== "B") begin
                $display("@%0t you have problem priority B",$time); errors++;
            end
            else correct++;
        end
        // check on pypass operations
        else if(bypass_A | bypass_B)begin //check on bypass
            if(bypass_A ==1 && out != A)begin
                $display("@%0t you have problem bypass A",$time); errors++;
            end
            else if(bypass_B ==1 && out != B)begin
                $display("@%0t you have problem bypass B",$time); errors++;
            end
            else correct++;
        end
        //check on invalid condition output
        else if(invalid) begin
            if (invalid ==1 && out!= 0 && leds!= leds)begin
                $display("@%0t you have problem in invalid ",$time); errors++;
            end
            else correct++;
        end
        else begin
```

```
        else begin
            //here we check on OP code
            case (opcode)
            OR:begin// check on priority first
                if(red_op_A==1 && red_op_B==1 && out != A && INPUT_PRIORITY== "A")begin
                    $display("@%0t you have problem priority A",$time); errors++;
                end
                else if(red_op_A==1 && red_op_B==1 && out != B && INPUT_PRIORITY== "B")begin
                    $display("@%0t you have problem priority B",$time); errors++;
                end
                else if(red_op_A==1 && out != (|A))begin
                    $display("@%0t you have problem in OR _red_A ",$time); errors++;
                end
                else if(red_op_B==1 && out != (|B))begin
                    $display("@%0t you have problem in OR _red_B ",$time); errors++;
                end
                else if(red_op_A==0 && red_op_B==0 && out != (A|B))begin
                    $display("@%0t you have problem in OR A|B ",$time); errors++;
                end
                else correct++;
            end
            XOR:begin
                if(red_op_A==1 && red_op_B==1 && out != A && INPUT_PRIORITY== "A")begin
                    $display("@%0t you have problem priority A",$time); errors++;
                end
                else if(red_op_A==1 && red_op_B==1 && out != B && INPUT_PRIORITY== "B")begin
                    $display("@%0t you have problem priority B",$time); errors++;
                end
                else if(red_op_A==1 && out != (^A))begin
                    $display("@%0t you have problem in XOR _red_A ",$time); errors++;
                end
                else if(red_op_B==1 && out != (^B))begin
                    $display("@%0t you have problem in XOR _red_B ",$time); errors++;
                end
                else if(red_op_A==0 && red_op_B==0 && out != (A^B))begin
                    $display("@%0t you have problem in XOR A^B ",$time); errors++;
                end
```

```
166         ADD:begin
167             if(FULL_ADDER == "ON" && out!= A+B+cin)begin
168                 $display("@%0t you have problem in ADDER with full adder ",$time); errors++;
169             end
170             else if(FULL_ADDER == "OFF" && out!= A+B)begin
171                 $display("@%0t you have problem in ADDER  ",$time); errors++;
172             end
173             else correct++;
174         end
175         MULT:begin
176             if(out!= A*B)begin
177                 $display("@%0t you have problem ADDER with full adder ",$time); errors++;
178             end
179             else correct++;
180         end
181         SHIFT:begin
182             if(direction==1 && out != {last_out[4:0],serial_in})begin
183                 $display("@%0t you have problem in shift left ",$time); errors++;
184             end
185             else if(direction==0 && out != {serial_in,last_out[5:1]})begin
186                 $display("@%0t you have problem in shift right ",$time); errors++;
187             end
188             else correct++;
189         end
190         ROTATE:begin
191             if(direction==1 && out != {last_out[4:0],last_out[5]})begin
192                 $display("@%0t you have problem in rotate left ",$time); errors++;
193             end
194             else if(direction==0 && out != {last_out[0],last_out[5:1]})begin
195                 $display("@%0t you have problem in rotate right ",$time); errors++;
196             end
197             else correct++;
198         end
```

Do file

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

vcover report ALSU_tb.ucdb -details -all -output coverage_report.txt
```

Count

```
# number of correct =10002 ,error=0
# ** Note: $finish    : E:/study/kareem wassem/ASSIGMENTS/2/ALSU/ALSU_tb.sv(71)
#    Time: 400080 ns  Iteration: 1  Instance: /ALSU_tb
```

Coverage report

```
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           50        50         0     100.0

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

Statement Coverage for file ALSU.v --

    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, rst, cin, 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 signed[15:0] leds;
    8                                           output reg signed[5:0] out;
    9
    10                                          reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
    11                                          reg [2:0] opcode_reg;
    12                                          reg signed [2:0] A_reg, B_reg; //change to signed
    13                                          reg signed[5:0] out_next;      //change to signed
    14                                          wire invalid_red_op, invalid_opcode, invalid;
    15
    16                                          //Invalid handling
    17              1                    959    assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
    18              1                    861    assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
    19              1                    542    assign invalid = invalid_red_op | invalid_opcode;
    20
    21                                          //Registering input signals
    22              1                   2001    always @(posedge clk or posedge rst) begin
    23                                            if(rst) begin
    24              1                    115        cin_reg <= 0;
    25              1                    115        red_op_B_reg <= 0;
    26              1                    115        red_op_A_reg <= 0;
    27              1                    115        bypass_B_reg <= 0;
    28              1                    115        bypass_A_reg <= 0;
    29              1                    115        direction_reg <= 0;
    30              1                    115        serial_in_reg <= 0;
    31              1                    115        opcode_reg <= 0;
    32              1                    115        A_reg <= 0;
    33              1                    115        B_reg <= 0;
    34              1                    115       out_next<=0; // we need a ff to keep value of output to be Zero
    35                                            end else begin
    36              1                   1886        cin_reg <= cin;
    37              1                   1886        red_op_B_reg <= red_op_B;
    38              1                   1886        red_op_A_reg <= red_op_A;
    39              1                   1886        bypass_B_reg <= bypass_B;
    40              1                   1886        bypass_A_reg <= bypass_A;
    41              1                   1886        direction_reg <= direction;
    42              1                   1886        serial_in_reg <= serial_in;
    43              1                   1886        opcode_reg <= opcode;
    44              1                   1886        A_reg <= A;
    45              1                   1886        B_reg <= B;
    46              1                   1886       out_next<=out; // we need a ff to keep value of output until serial in reg get its new value
    47                                            end
    48                                          end
    49
    50                                          //leds output blinking
    51              1                   2061    always @(posedge clk or posedge rst) begin
    52                                            if(rst) begin
    53              1                    173        leds <= 0;
    54                                            end else begin
    55                                                if (invalid)
    56              1                   1073            leds <= ~leds;
    57                                                else
    58              1                    815            leds <= 0;
    59                                            end
    60                                          end
    61
    62                                          //ALSU output processing
    63              1                   1946    always @(posedge clk or posedge rst) begin
    64
    65                                            if(rst) begin
    66              1                    115        out <= 0;
    67                                            end
    68                                            else begin
    69                                              if (bypass_A_reg && bypass_B_reg)
    70              1                     18            out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    71                                              else if (bypass_A_reg)
```

```
72              1                       151           out <= A_reg;
73                                                  else if (bypass_B_reg)
74              1                       160           out <= B_reg;
75                                                  else if (invalid) // cahnge the priority of invalid bits after bypass_reg
76              1                       846             out <= 0;
77                                                  else begin
78                                                      case (opcode)
79                                                        3'h0: begin //change Opcode to OR not AND
80                                                          if (red_op_A_reg && red_op_B_reg)
81              1                        16                   out = (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
82                                                          else if (red_op_A_reg)
83              1                        23                   out <= |A_reg;
84                                                          else if (red_op_B_reg)
85              1                         8                   out <= |B_reg;
86                                                          else
87              1                        80                   out <= A_reg | B_reg;
88                                                        end
89                                                        3'h1: begin // change opcode to XOR not OR
90                                                          if (red_op_A_reg && red_op_B_reg)
91              1                        14                   out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
92                                                          else if (red_op_A_reg)
93              1                        21                   out <= ^A_reg;
94                                                          else if (red_op_B_reg)
95              1                        10                   out <= ^B_reg;
96                                                          else
97              1                        85                   out <= A_reg ^ B_reg;
98                                                        end
99                                                        3'h2: begin //here we add condition to check full adder if ON or OFF
100                                                         if(FULL_ADDER == "ON")
101             1                       115                   out <= A_reg + B_reg+cin_reg;
102                                                         else if(FULL_ADDER == "OFF")
103                                                           out <= A_reg + B_reg;
104
105                                                       end
106             1                       104               3'h3: out <= A_reg * B_reg;
107                                                       3'h4: begin
108
109                                                         if (direction_reg)
110             1                        40                   out <= {out_next[4:0], serial_in_reg};
111                                                         else
112             1                        52                   out <= {serial_in_reg, out_next[5:1]};
113                                                       end
114                                                       3'h5: begin
115                                                         if (direction_reg)
116             1                        30                   out <= {out_next[4:0], out_next[5]};
117                                                         else
118             1                        40                   out <= {out_next[0], out_next[5:1]};
119                                                       end
120                                                     endcase
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        32        32         0     100.0

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

Branch Coverage for file ALSU.v --

------------------------------------IF Branch------------------------------------
    23                                      2001     Count coming in to IF
    23              1                        115       if(rst) begin
    35              1                       1886       end else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    52                                      2061     Count coming in to IF
    52              1                        173       if(rst) begin
    54              1                       1888       end else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    55                                      1888     Count coming in to IF
    55              1                       1073           if (invalid)
    57              1                        815           else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    65                                      1946     Count coming in to IF
    65              1                        115       if(rst) begin
    68              1                       1831       else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    69                                      1831     Count coming in to IF
    69              1                          18         if (bypass_A_reg && bypass_B_reg)
    71              1                         151         else if (bypass_A_reg)
    73              1                         160         else if (bypass_B_reg)
    75              1                         846         else if (invalid) // cahnge the priority of invalid bits after bypass_reg
    77              1                         656         else begin
Branch totals: 5 hits of 5 branches = 100.0%

----------------------------------CASE Branch----------------------------------
    78                                       656     Count coming in to CASE
    79              1                        127             3'h0: begin //change Opcode to OR not AND
    89              1                        130             3'h1: begin // change opcode to XOR not OR
    99              1                        115             3'h2: begin //here we add condition to check full adder if ON or OFF
    106             1                        104             3'h3: out <= A_reg * B_reg;
    107             1                         92             3'h4: begin
    114             1                         70             3'h5: begin
```

```
                                        18     All False Count
Branch totals: 7 hits of 7 branches = 100.0%

------------------------------------IF Branch------------------------------------
    80                                  127     Count coming in to IF
    80              1                    16                 if (red_op_A_reg && red_op_B_reg)
    82              1                    23                 else if (red_op_A_reg)
    84              1                     8                 else if (red_op_B_reg)
    86              1                    80                 else
Branch totals: 4 hits of 4 branches = 100.0%

------------------------------------IF Branch------------------------------------
    90                                  130     Count coming in to IF
    90              1                    14                 if (red_op_A_reg && red_op_B_reg)
    92              1                    21                 else if (red_op_A_reg)
    94              1                    10                 else if (red_op_B_reg)
    96              1                    85                 else
Branch totals: 4 hits of 4 branches = 100.0%

------------------------------------IF Branch------------------------------------
    109                                  92     Count coming in to IF
    109             1                    40                 if (direction_reg)
    111             1                    52                 else
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    115                                  70     Count coming in to IF
    115             1                    30                 if (direction_reg)
    117             1                    40                 else
Branch totals: 2 hits of 2 branches = 100.0%


Condition Coverage:
    Enabled Coverage            Active   Covered    Misses % Covered
    ----------------            ------   -------    ------ ---------
    FEC Condition Terms              6         6         0     100.0
```

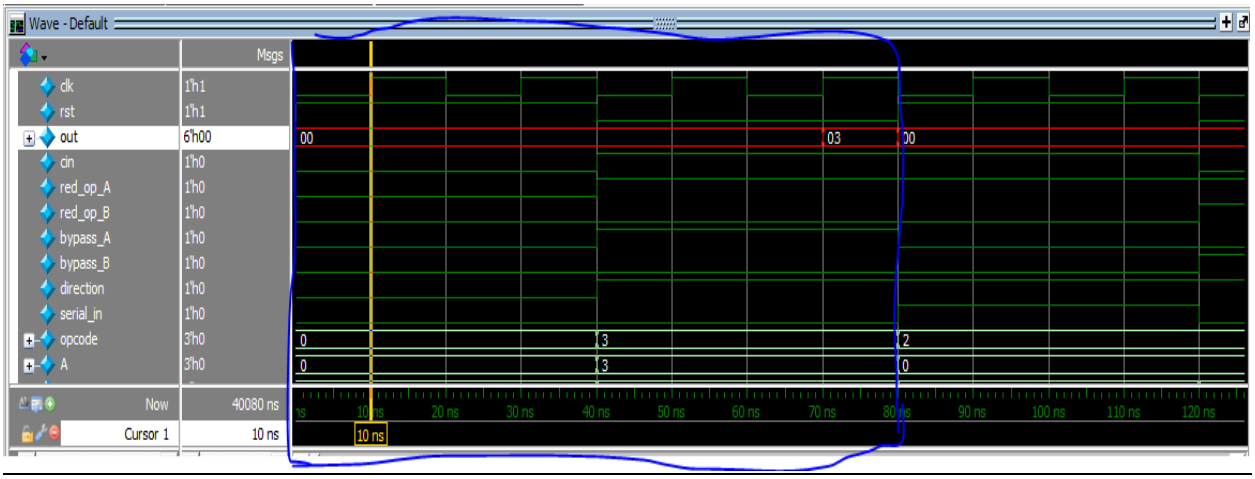
```
==============================Condition Details================================

Condition Coverage for file ALSU.v --

----------------Focused Condition View-------------------
Line       69 Item    1  (bypass_A_reg && bypass_B_reg)
Condition totals: 2 of 2 input terms covered = 100.0%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  bypass_A_reg         Y
  bypass_B_reg         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  bypass_A_reg_0        -
  Row   2:          1  bypass_A_reg_1        bypass_B_reg
  Row   3:          1  bypass_B_reg_0        bypass_A_reg
  Row   4:          1  bypass_B_reg_1        bypass_A_reg

----------------Focused Condition View-------------------
Line       80 Item    1  (red_op_A_reg && red_op_B_reg)
Condition totals: 2 of 2 input terms covered = 100.0%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  red_op_A_reg         Y
  red_op_B_reg         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  red_op_A_reg_0        -
  Row   2:          1  red_op_A_reg_1        red_op_B_reg
  Row   3:          1  red_op_B_reg_0        red_op_A_reg
  Row   4:          1  red_op_B_reg_1        red_op_A_reg

----------------Focused Condition View-------------------
Line       90 Item    1  (red_op_A_reg && red_op_B_reg)
Condition totals: 2 of 2 input terms covered = 100.0%

    Input Term   Covered  Reason for no coverage   Hint
   -----------  --------  ----------------------  --------------
  red_op_A_reg         Y
  red_op_B_reg         Y
```

```
Toggle Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Toggle Bins                    130       130         0     100.0

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

Toggle Coverage for File ALSU.v --

       Line                                  Node      1H->0L      0L->1H                          "Coverage"
-------------------------------------------------------------------------------------------------------------
          4                             serial_in           1           1                             100.00
          4                                   rst           1           1                             100.00
          4                                red_op_B           1           1                             100.00
          4                                red_op_A           1           1                             100.00
          4                             direction             1           1                             100.00
          4                                   clk           1           1                             100.00
          4                                   cin           1           1                             100.00
          4                                bypass_B           1           1                             100.00
          4                                bypass_A           1           1                             100.00
          5                             opcode[2]             1           1                             100.00
          5                             opcode[1]             1           1                             100.00
          5                             opcode[0]             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
          6                                  A[2]             1           1                             100.00
          6                                  A[1]             1           1                             100.00
          6                                  A[0]             1           1                             100.00
          7                               leds[9]             1           1                             100.00
          7                               leds[8]             1           1                             100.00
          7                               leds[7]             1           1                             100.00
          7                               leds[6]             1           1                             100.00
          7                               leds[5]             1           1                             100.00
          7                               leds[4]             1           1                             100.00
          7                               leds[3]             1           1                             100.00
          7                               leds[2]             1           1                             100.00
          7                               leds[1]             1           1                             100.00
          7                              leds[15]             1           1                             100.00
          7                              leds[14]             1           1                             100.00
          7                              leds[13]             1           1                             100.00
          7                              leds[12]             1           1                             100.00
          7                              leds[11]             1           1                             100.00
          7                              leds[10]             1           1                             100.00
          7                               leds[0]             1           1                             100.00
          8                                out[5]             1           1                             100.00
          8                                out[4]             1           1                             100.00
          8                                out[3]             1           1                             100.00
          8                                out[2]             1           1                             100.00
```

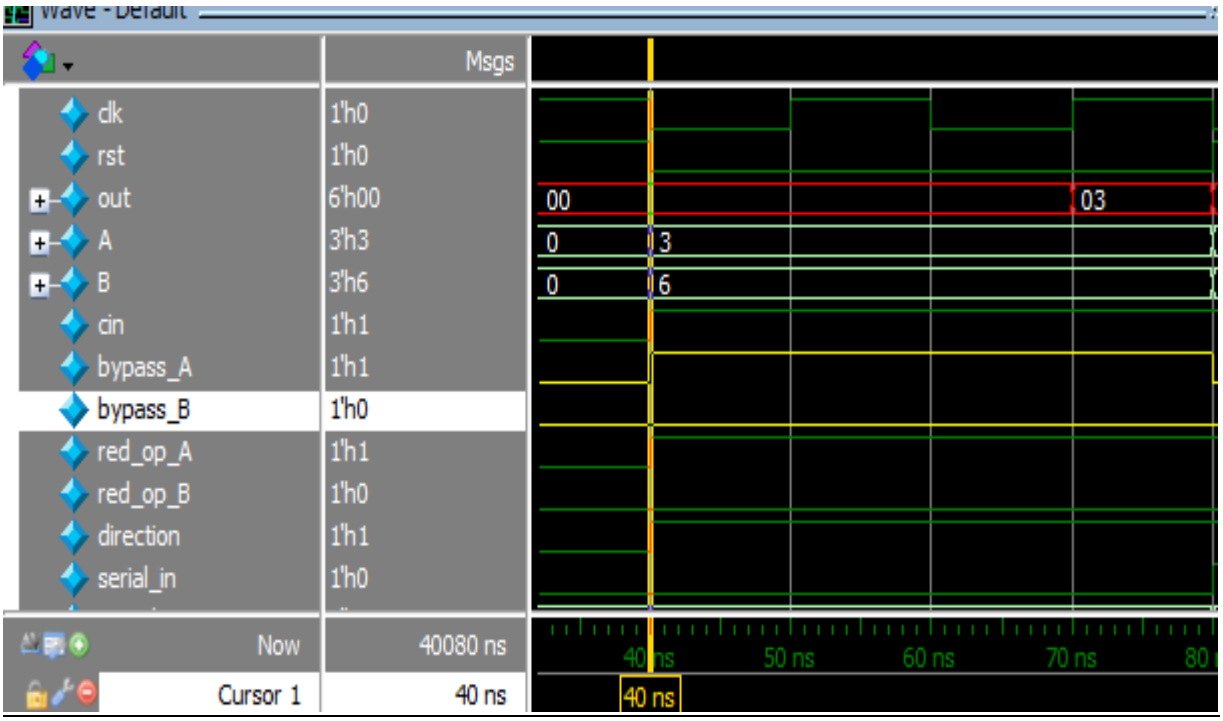
Verification plan

	Label	Description	Stimulus Generation	Functionality Check
2	ALSU_1	we assert reset on start so OUT should be low and led should be low	Directed at the start of the simulation then it randomized under cosntraint to be of high 95 % from time	A checker in the testbench to make sure the output is correct by test function
3	ALSU_2	when the byPass_A is asserted OUT take value or reg A igonre Opcode and byPass_B is asserted OUT take value or reg B if Both high so out take input with HIGH priority	Randomized in class under constraint that make bypassA and bypass B is Low 90 % of time	A checker in the testbench to make sure the output is functionally correct by test function
4	ALSU_3	when Opcode= 6 or 7 its invalid and if red_op1 or red_op2 is high and OPcode is not or ,xor so its invalid case then Output is low	Randomized in class	A checker in the testbench to make sure the output is functionally correct by test function
5	ALSU_4	when opcode =OR so output is =|A when red_opA is high same output =|B if this red_op_b high if both high so output check priority and if both low output =A|B	Randomized in class under consraints that Opcode is valid most of simulation and also if OR operation so redA and RedB is high together most of simulation and also at least input if RedA is high only A should have at least 1 bit =1	A checker in the testbench to make sure the output is functionally correct by test function
6	ALSU_5	when opcode =XOR so output is =^A when red_opA is high same output =^B if this red_op_b high if both high so output check priority and if both low output =A^B	Randomized in class under consraints if xOR operation so redA and RedB is high together most of simulation and also at least input if RedA is high only A should have at least 1 bit =1 and same for B	A checker in the testbench to make sure the output is functionally correct by test function
7	ALSU_6	when opcode =ADD and full adder on so output =A+B+cin if full adder off out=A+B	Randomized in class	A checker in the testbench to make sure the output is functionally correct by test function
8	ALSU_7	when opcode =mult so out=A*B	Randomized in class	A checker in the testbench to make sure the output is functionally correct by test function
9	ALSU_8	when opcode =SHIFT and depend on Direction output will be left or right and serial in	Randomized in class	A checker in the testbench to make sure the output is functionally correct by test function
10	ALSU_9	when opcode =ROTATE and depend on Direction output will be rotate left or right	Randomized in class	A checker in the testbench to make sure the output is functionally correct by test function

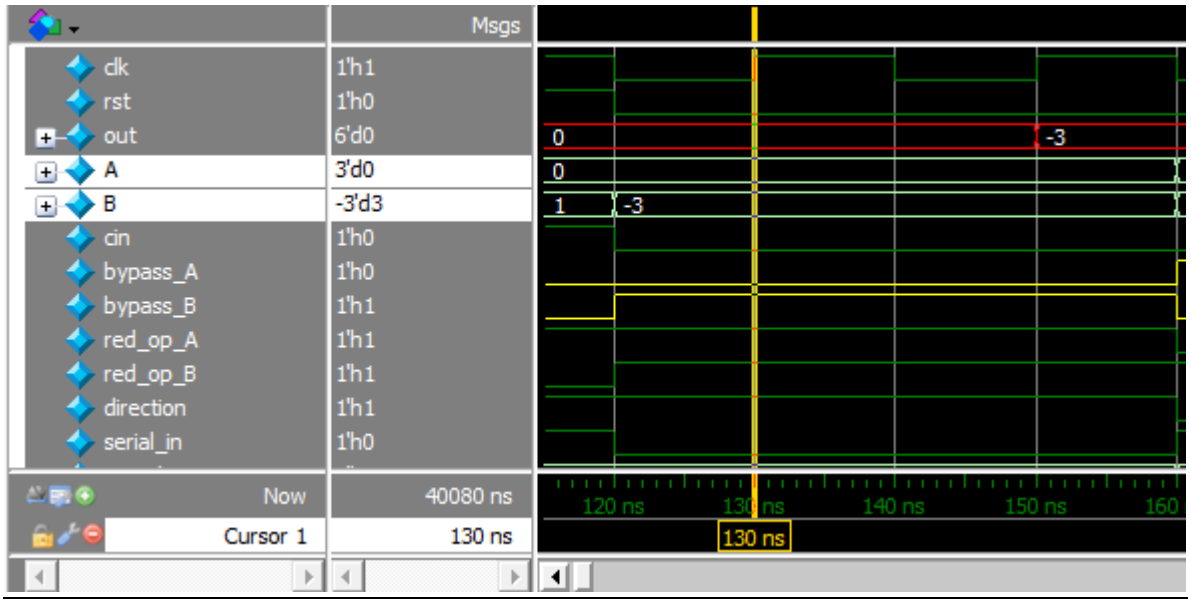
Simulation



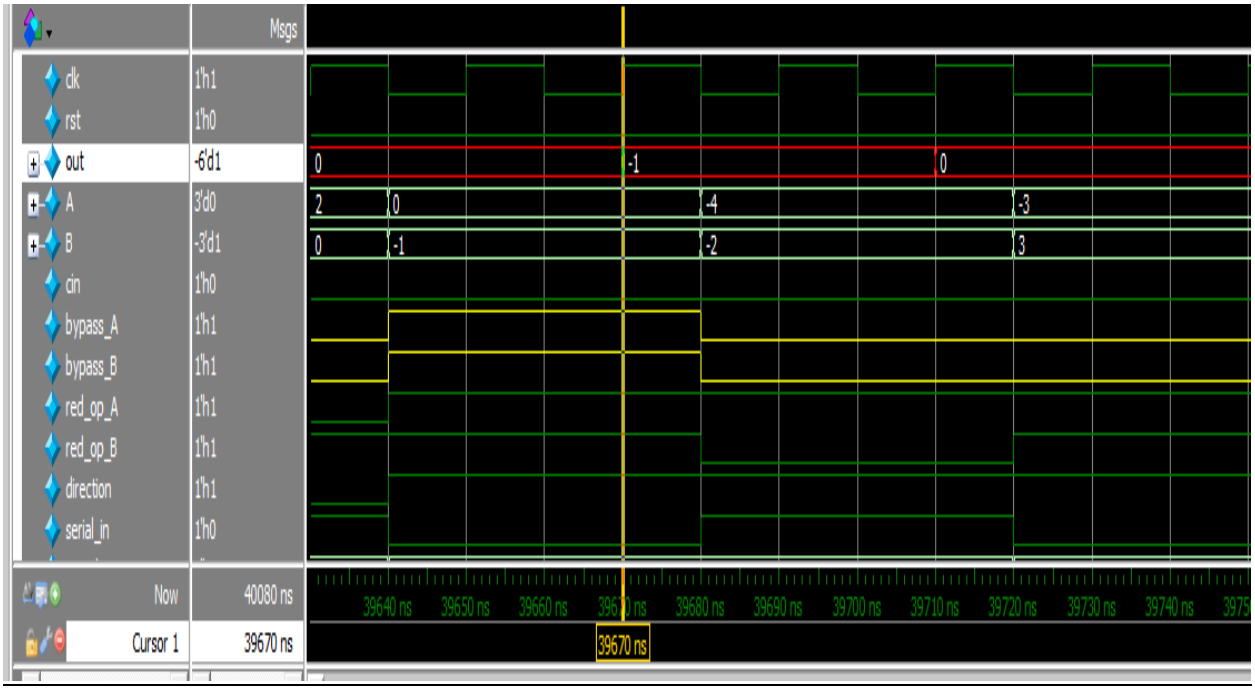
bypassA_check



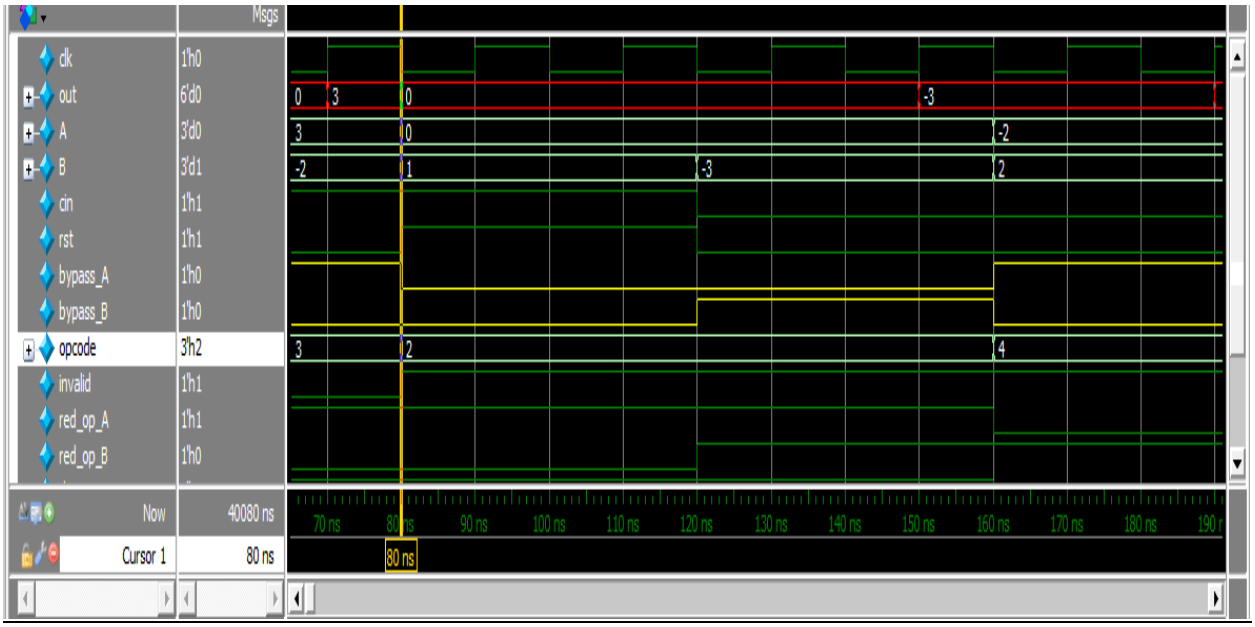
Bypass B check



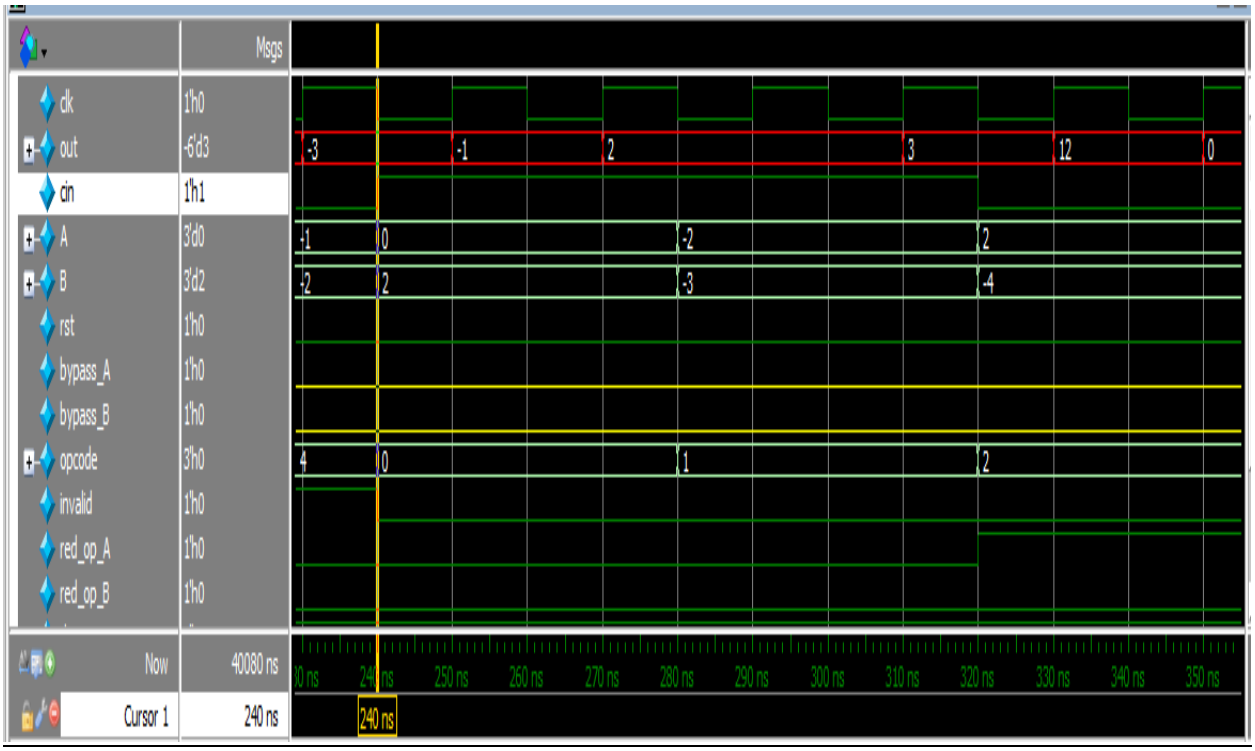
Bypass A and B check priority on B



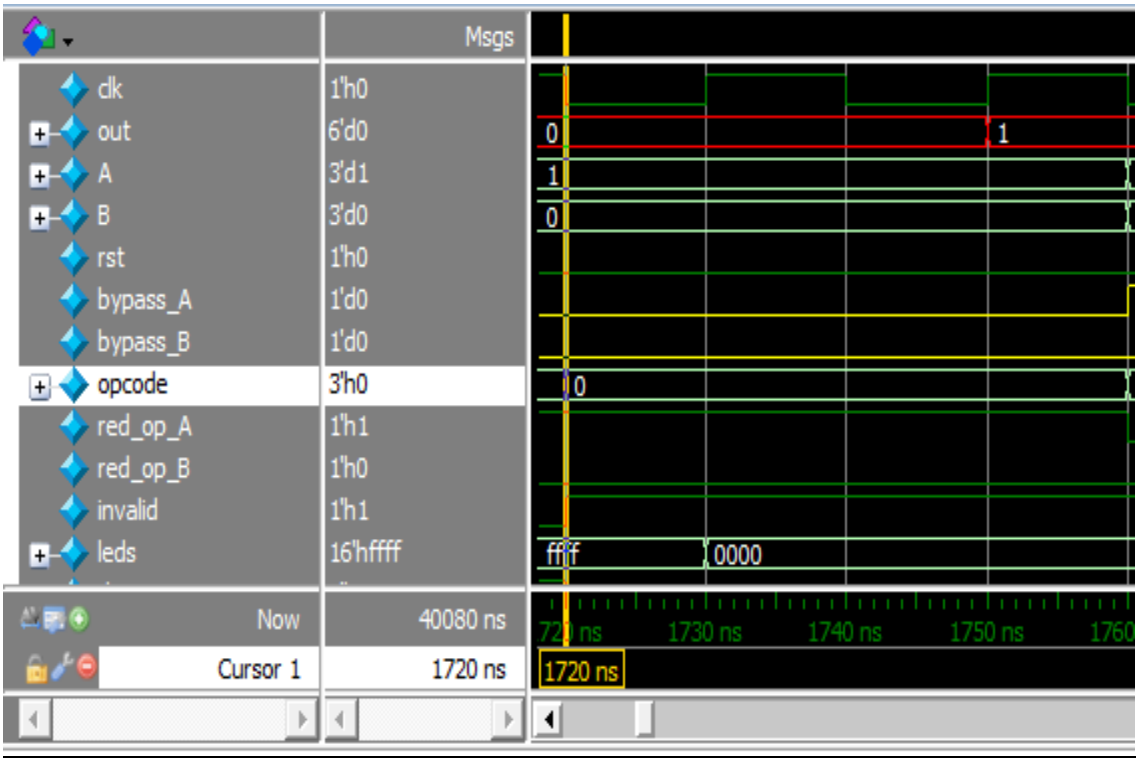
Invalid case check



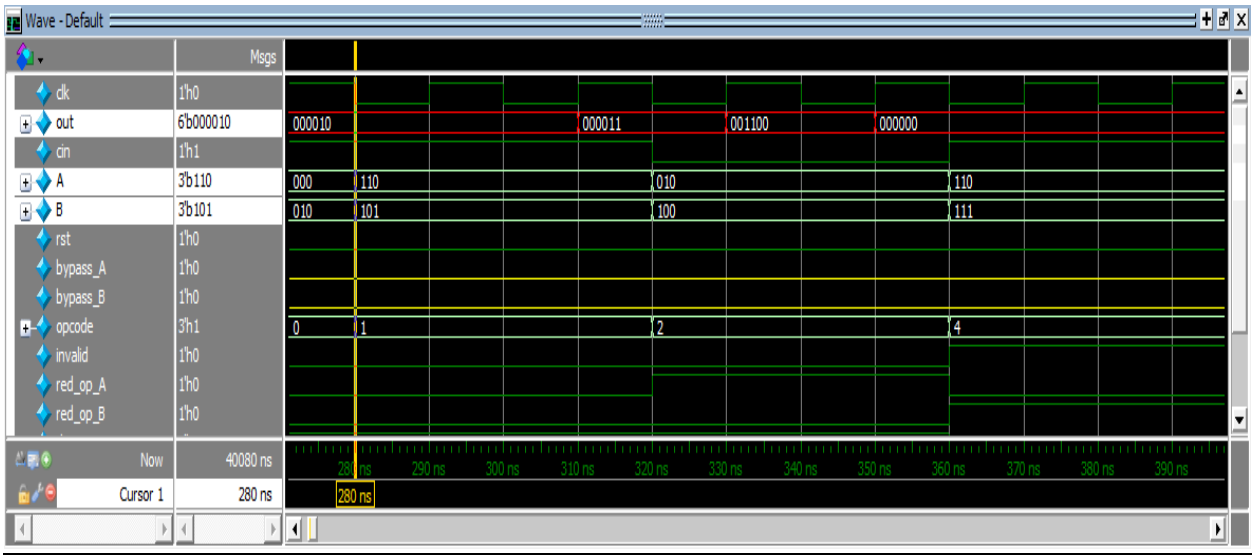
OR check



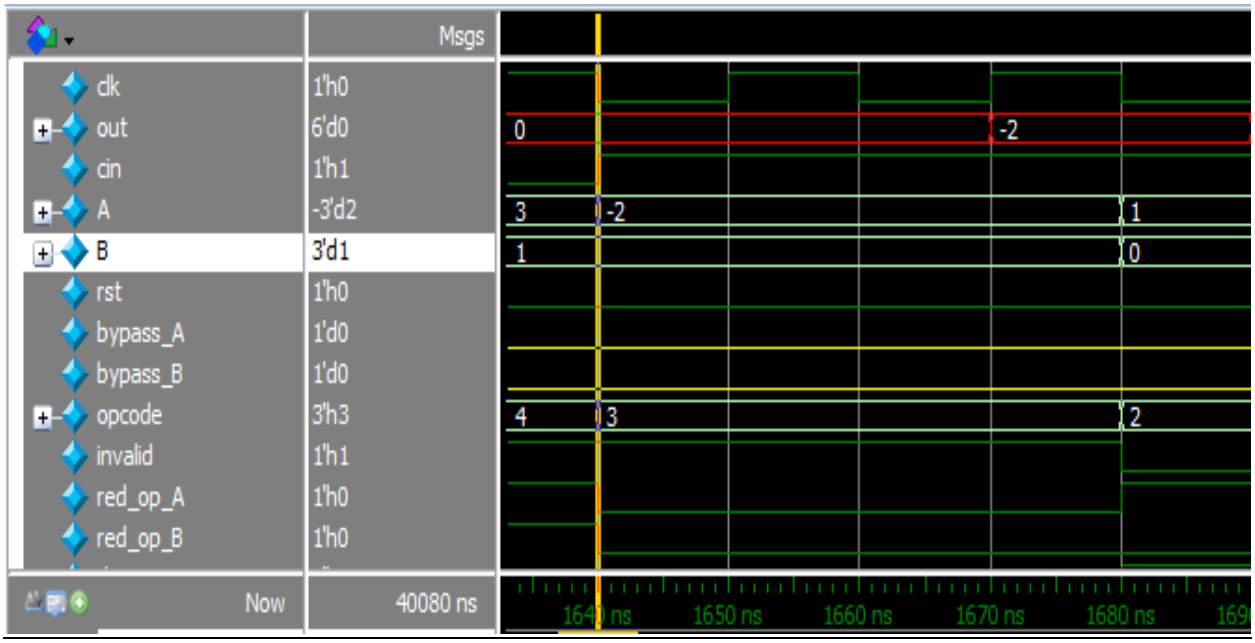
OR ON A only



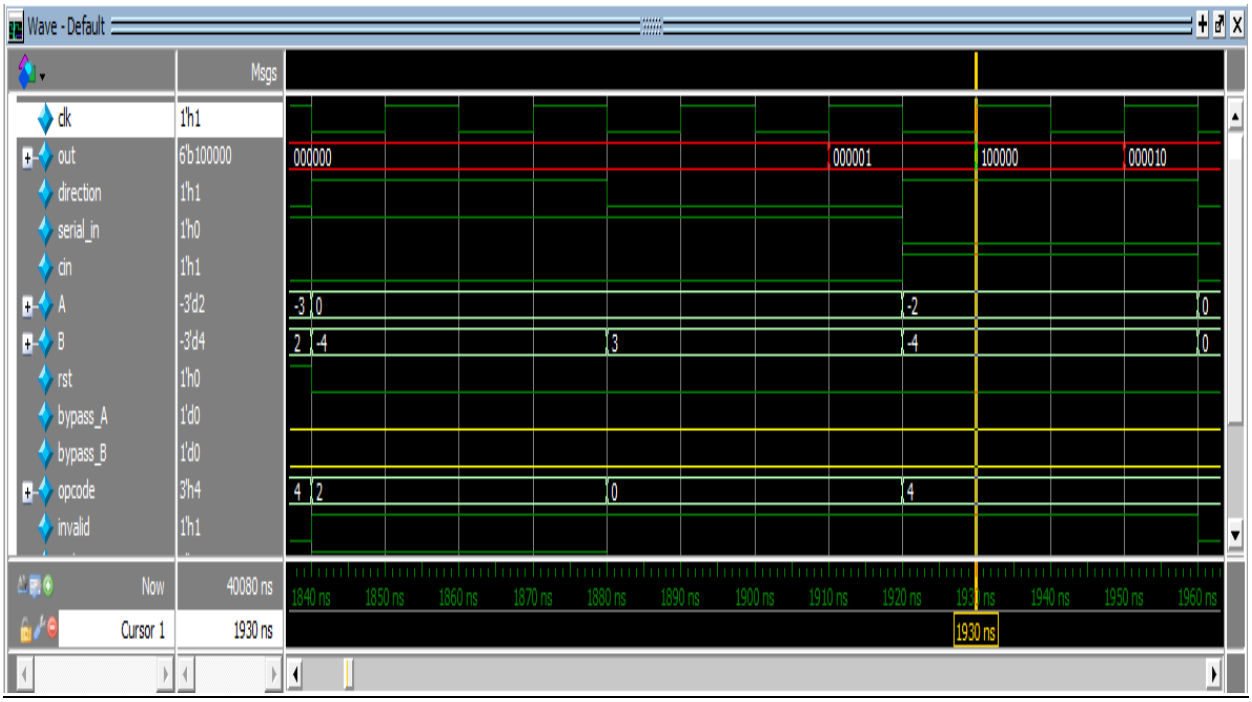
XOR



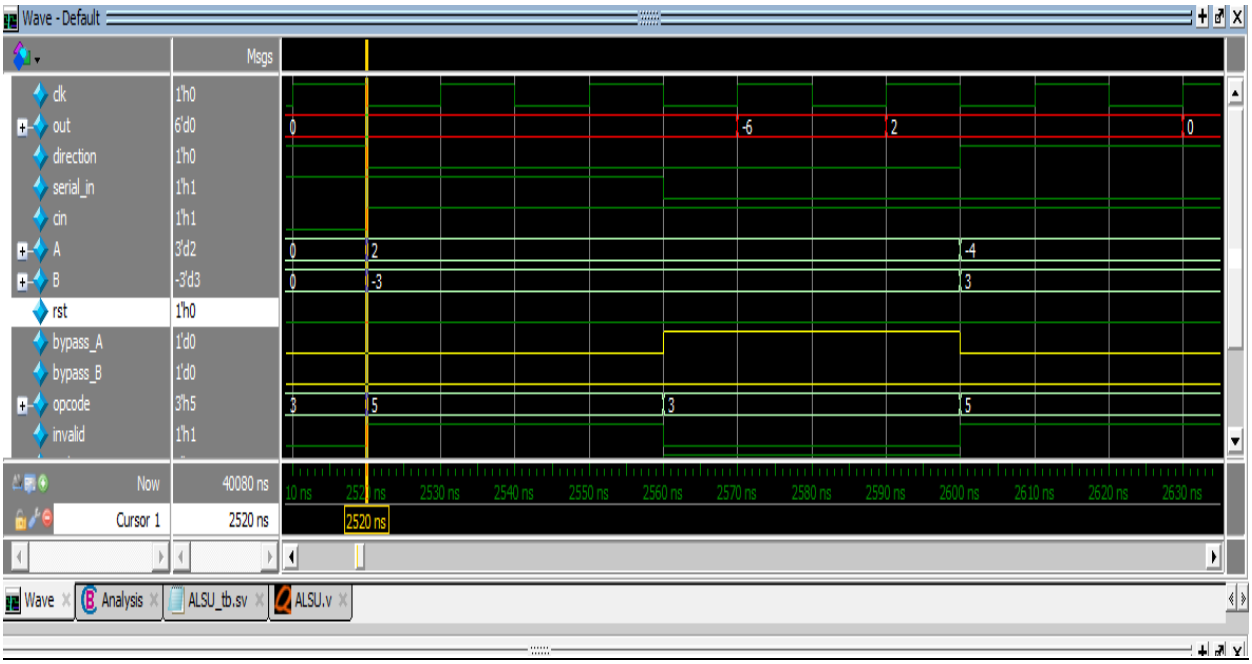
Mult



Shift_left



Rotate left


Wave - Default
Msgs
clk
out
direction
serial_in
cin
A
B
rst
bypass_A
bypass_B
opcode
invalid
1'h0
6'd0
1'h0
1'h1
1'h1
3'd2
-3'd3
1'h0
1'd0
1'd0
3'h5
1'h1
0
-6
2
0
0
2
-4
0
-3
3
3
5
3
5
Now
40080 ns
Cursor 1
2520 ns
2520 ns
2530 ns
2540 ns
2550 ns
2560 ns
2570 ns
2580 ns
2590 ns
2600 ns
2610 ns
2620 ns
2630 ns
Wave
Analysis
ALSU_tb.sv
ALSU.v