MAZARITA_TEAM

Alaa_Salah

Mohanad_Mohamed

Contents

1) Design Code: 2
2) Snippets: 11
3) Synthesis : 18
4) XDC : 20
5) Sources Files: 25

1)Design Code:

→ALSU:

```
module ALSU_(
                input wire [2:0] A,
                input wire [2:0] B,
                input wire [2:0] opcode,
                input wire cin,
                input wire serial_in,
                input wire direction,
                input wire red_op_A,
                input wire red_op_B,
                input wire byPass_A,
                input wire byPass_B,
                input wire clk,
                input wire rst,

                output reg valid,
                output reg [5:0] out,
                output reg [15:0] leds
            );
//parameters
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
//internal signals
wire [15:0] led_blink;

always @(posedge clk or posedge rst) begin
    //valid <=1;
    if (rst) begin
        // reset
        out <= 'b0;
        leds <= 'b0;
        valid <= 1;
    end
    else if(byPass_A && ~byPass_B) begin
        out <= A;
        valid <= 1;
    end
    else if(byPass_B && ~byPass_A) begin
        out <= B;
```

```
        valid <= 1;
    end
    else if(byPass_A && byPass_B) begin
        out <= INPUT_PRIORITY;
        valid <= 1;
    end
    else begin

        case (opcode)
            3'b000 : begin
                valid <= 1;                 // AND
                if(red_op_A && ~red_op_B)
                    out <= &A;
                else if(~red_op_A && red_op_B)
                    out <= &B;
                else if(red_op_A && red_op_B)
                    out <= &INPUT_PRIORITY;
                else
                    out <= A & B;
            end

            3'b001 : begin
                valid <= 1;                     // XOR
                if(red_op_A && ~red_op_B)
                    out <= ^A;
                else if(~red_op_A && red_op_B)
                    out <= ^B;
                else if(red_op_A && red_op_B)
                    out <= ^INPUT_PRIORITY;
                else
                    out <= A ^ B;
            end

            3'b010 : begin                      // Addition
                if(red_op_A || red_op_B) begin
                    out <= 0;
                    leds <= led_blink;
                    valid <=0;
                end
                else begin
                    valid <= 1;
                if(FULL_ADDER == "ON")
                    out <= A + B + cin;
                else
                    out <= A + B;
```

```
            end
        end
        3'b011 : begin                      // Multiplication
            if(red_op_A || red_op_B) begin
                out <= 0;
                leds <= led_blink;
                valid <=0;
            end
            else
                valid <= 1;
                out <= A * B;
        end
        3'b100 : begin                      // Shift output
            if(red_op_A || red_op_B) begin
                out <= 0;
                leds <= led_blink;
                valid <=0;
            end
            else begin
                valid <= 1;
                if(direction)
                    out <= {out[4:0], serial_in};
                else
                    out <= {serial_in, out[5:1]};
            end
        end
        3'b101 : begin                      // Rotation
            if(red_op_A || red_op_B) begin
                out <= 0;
                leds <= led_blink;
                valid <=0;
            end
            else begin
                valid <= 1;
                if(direction)
                    out <= {out[4:0], out[5]};
                else
                    out <= {out[0], out[5:1]};
            end
        end
        3'b110 || 3'b111 : begin            // Invalid opcode
            out <= 0;
            leds <= led_blink;
            valid <=0;
        end
```

```
            default :   begin                   // to be continued
                out <= 0;
                leds <= 0;
                valid <= 1;
                end
    endcase
    end
end
blinking blink(clk,led_blink);
endmodule
```

→Blinking :

```
module blinking(clk,led );//input reset,
input wire clk;
reg [27:0] counter;
output reg [15:0] led;
reg clkout;

initial begin
    counter = 0;
    clkout = 0;
    led = 0;
end
always @(posedge clk) begin
    if (counter == 0) begin
        counter <= 149999999;//actual value used to be shown on board blinking
        //counter <= 10; to be shown in simulation
        clkout <= ~clkout;
    end else begin
        counter <= counter - 1;
    end
end

always @(posedge clkout) begin
            led <= ~led;
    end
endmodule
```

→Seven_Segment Display

```
module ALSU_Display(
                    input clock_100Mhz, // 100 Mhz clock source on Basys 3 FPGA
                    input reset,
                    input valid,
                    input wire[5:0] out_ALU,
                    output reg [3:0] Anode_Activate,
                    output reg [6:0] LED_out
                    );

    reg [3:0] LED_BCD;
   reg [19:0] refresh_counter; // 20-bit for creating 10.5ms refresh period or
380Hz refresh rate
   //reg [3:0] refresh_counter;       // the first 2 MSB bits for creating 4 LED-
activating signals with 2.6ms digit period
    wire [1:0] LED_activating_counter;
    //output reg [3:0] Anode_Activate, // anode signals of the 7-segment LED
display
    //output reg [6:0] LED_out// cathode patterns of the 7-segment LED display
    //=================================================================
    always @(posedge clock_100Mhz or posedge reset)
    begin
        if(reset==1)
            refresh_counter <= 0;
        else
            refresh_counter <= refresh_counter + 1;
    end
    assign LED_activating_counter = refresh_counter[19:18];
    // anode activating signals for 4 LEDs, digit period of 2.6ms
    // decoder to generate anode signals
    always @(*)
    begin
        case(LED_activating_counter) //anode active high so invert all bits
        2'b00: begin
            Anode_Activate = 4'b1110;
            // activate LED4 and Deactivate LED2, LED3, LED1
            if(~valid)
                LED_BCD =  4'b1011;
            else LED_BCD = out_ALU[3:0];
              end
        2'b01: begin
            Anode_Activate = 4'b1101;
            // activate LED3 and Deactivate LED1, LED2, LED4
            if(~valid)
                LED_BCD = 4'b0000;
            else LED_BCD = out_ALU[5:4];
```

```
              end
        2'b10: begin
            Anode_Activate = 4'b1011;
            // activate LED2 and Deactivate LED3, LED1, LED4
            if(~valid)
                LED_BCD =  4'b0100;
            else LED_BCD = 4'bxxxx;
                end
        2'b11: begin
            Anode_Activate = 4'b0111;
            // activate LED1 and Deactivate LED2, LED3, LED4
            if(~valid)
                LED_BCD =  4'b1110;
            else LED_BCD = 4'bxxxx;
               end
        endcase
    end
    // Cathode patterns of the 7-segment LED display
    always @(*)
    begin
        case(LED_BCD) // cathode active low
        4'b0000: LED_out = 7'b0000001; // "0"
        4'b0001: LED_out = 7'b1001111; // "1"
        4'b0010: LED_out = 7'b0010010; // "2"
        4'b0011: LED_out = 7'b0000110; // "3"
        4'b0100: LED_out = 7'b1001100; // "4"
        4'b0101: LED_out = 7'b0100100; // "5"
        4'b0110: LED_out = 7'b0100000; // "6"
        4'b0111: LED_out = 7'b0001111; // "7"
        4'b1000: LED_out = 7'b0000000; // "8"
        4'b1001: LED_out = 7'b0000100; // "9"

        4'b1010: LED_out = 7'b0001000; // "10--> A"
        4'b1011: LED_out = 7'b1100000; // "11--> b"
        4'b1100: LED_out = 7'b0110001; // "12--> C"
        4'b1101: LED_out = 7'b1000010; // "13--> d"
        4'b1110: LED_out = 7'b0110000; // "14--> E"
        4'b1111: LED_out = 7'b0111000; // "15--> F"
        default: LED_out = 7'b1111110; // "- activate common G only"
        endcase
    end

endmodule
```

→D f/f

```
module D_Asynch(d,clk,rst,q);
input wire rst,clk;
input wire [2:0] d;
output reg [2:0] q;
//output wire [2:0] qbar;

always @(posedge clk or posedge rst) begin
    if (rst)
        q<=0;
    else begin
        q <= d;
    end
end
//assign qbar = ~q;
endmodule
```

→TopModule

```
module ALSU_TO_SEVEN_SEGMENT(
                            input wire [2:0] A,
                            input wire [2:0] B,
                            input wire [2:0] opcode,
                            input wire cin,
                            input wire serial_in,
                            input wire direction,
                            input wire red_op_A,
                            input wire red_op_B,
                            input wire byPass_A,
                            input wire byPass_B,
                            input wire clk,
                            input wire rst,

                            output wire[3:0] anode,
                            output wire[6:0] cathode,
                            output wire[15:0] leds
                                );
//internal wires
wire [5:0] out_ALSU;
wire valid;
```

```
wire [2:0] A_registered,B_registered;
D_Asynch A_(A,clk,rst,A_registered);
D_Asynch B_(B,clk,rst,B_registered);
// instance for ALSU block
ALSU_
ALSU(A_registered,B_registered,opcode,cin,serial_in,direction,red_op_A,red_op_B,b
yPass_A,byPass_B,clk,rst,valid,out_ALSU,leds);

//instance for 7 segment display
ALSU_Display ALSU_display_on_7segment(clk,rst,valid,out_ALSU,anode,cathode);

endmodule
```

→Test Bench

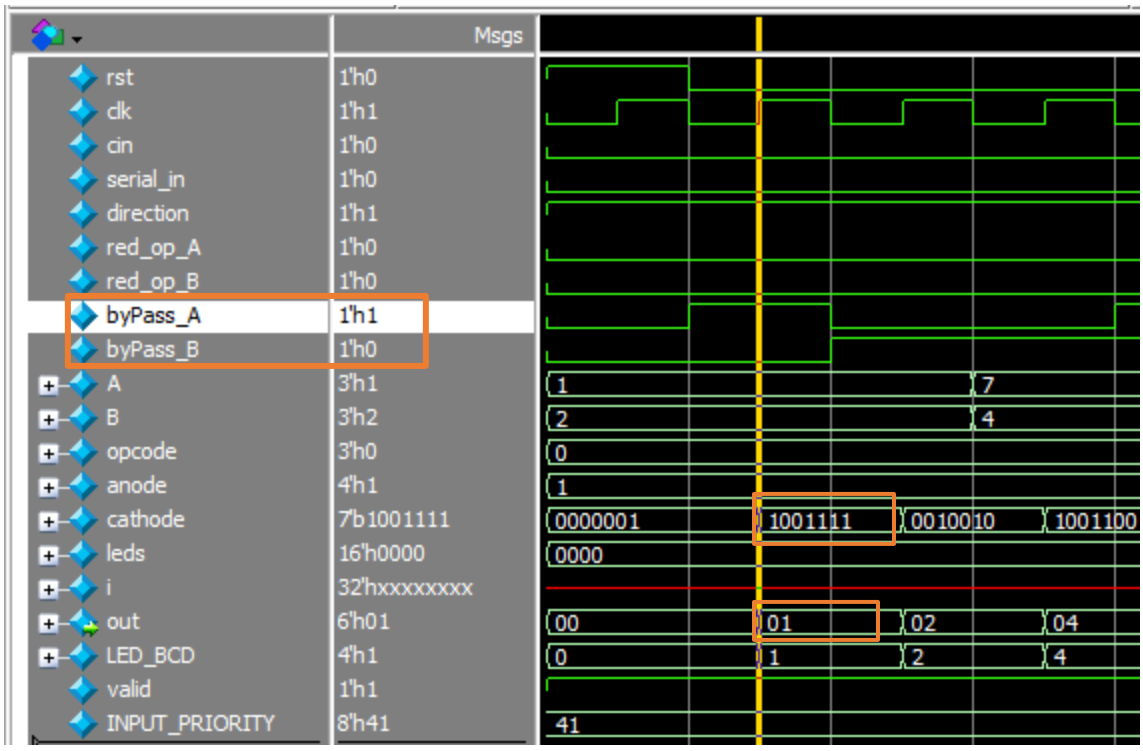
```
module ALSU_TO_SEVEN_SEGMENT_tb();
reg rst,clk,cin,serial_in,direction,red_op_A,red_op_B,byPass_A,byPass_B;
reg [2:0] A,B,opcode;
wire [3:0] anode ;
wire [6:0] cathode;
wire [15:0] leds;
//wire valid;
initial begin
    clk = 0;
    forever #1 clk = ~clk;
end
integer i ;
initial begin
    rst = 1;
    cin = 0;
    serial_in = 0;
    direction = 1;// for left
    red_op_B = 0;
    red_op_A = 0;
    byPass_A = 0;
    byPass_B = 0;
    A = 'b001;
    B = 'b010;
    opcode = 000; // for AND operation
    #2;
    // for checking bypass A
    byPass_A = 1;
    //byPass_B = 0;
    rst = 0;
```

```
    #4;
    // for checking bypass B
    byPass_A = 0;
    byPass_B = 1;
    #2;
    //for checking input priority
    A = 'b111;
    B = 'b100;
    #4;
    byPass_A = 1;
    #2;
    byPass_A = 0;
    byPass_B = 0;
    rst = 1;
    #2;
    //then go through opcode case
    rst = 0;
    //#2;
    // for reduction A
    red_op_A = 1;
    red_op_B = 0;
    #2;
    //for reduction B
    red_op_A = 0;
    red_op_B = 1;
    #2;
    // checking input priority to reduction operation opcode = 000 (AND)
    red_op_A = 1;
    #2;
    // bitwise AND --> out = A & B
    red_op_A = 0;
    red_op_B = 0;
    #2;
    // the same operation at opcode = 001 XOR(^) so i'll write only one case
    // bit wise XOR
    opcode = 'b001;
    #2;
    //addition
    A = 'b010;
    B = 'b011;
    opcode = 'b010;
    #2;
    opcode = 'b011;//multplication
    #2;
    // check invalid case when not logic opcode but use red-op-a
```

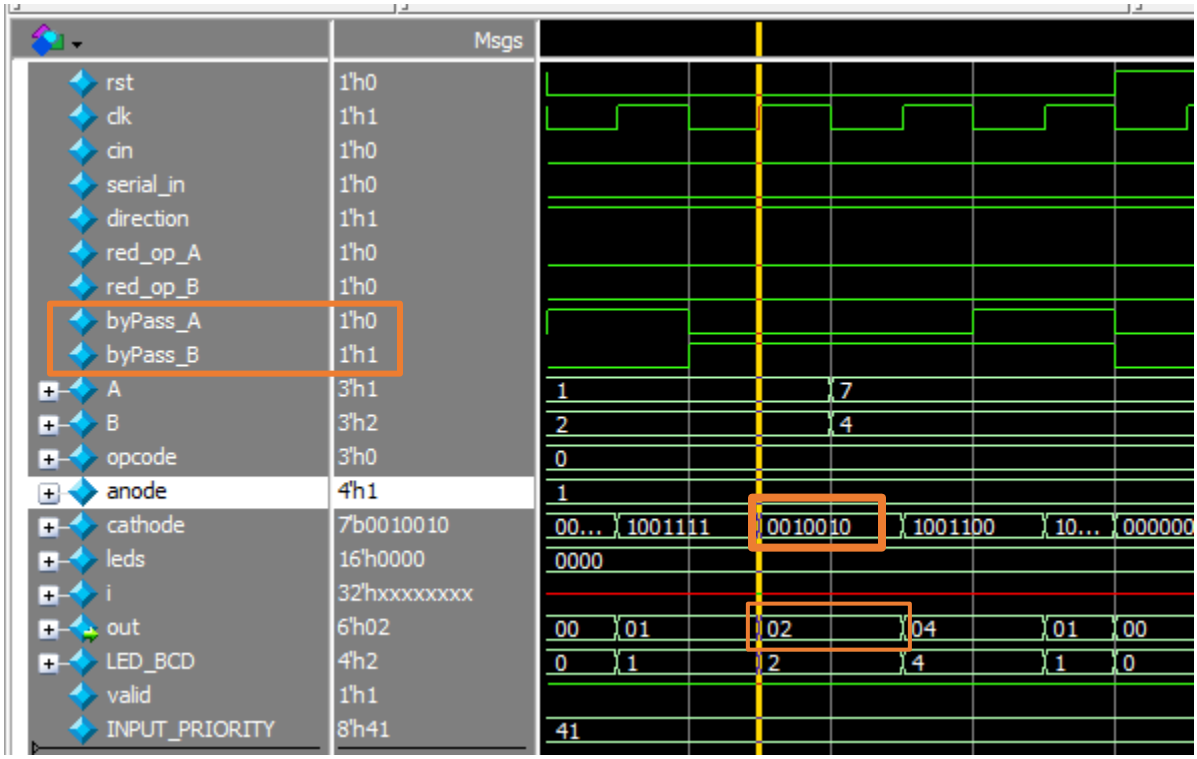
```
    red_op_A = 1;
    #20;// blinking working well
    rst = 1;
    red_op_A = 0;
    red_op_B = 0;
    #2;
    rst = 0;
    opcode = 'b100;// left operation for each clk posedge then put long delay and
random for serial_in
    for(i = 0;i<10;i = i+1) begin
        @(negedge clk)
        serial_in = $random;
    end
    #2;
    direction = 0;
    for(i = 0;i<10;i = i+1) begin
        @(negedge clk)
        serial_in = $random;
    end
    #2;
    opcode = 'b101;
    #2;
    direction = 1;
    #10;
    direction = 0;
    #10;
    $stop;
end
ALSU_TO_SEVEN_SEGMENT
DUT(A,B,opcode,cin,serial_in,direction,red_op_A,red_op_B,byPass_A,byPass_B,clk,rs
t,anode,cathode,leds);
endmodule
```

2) Snippets:

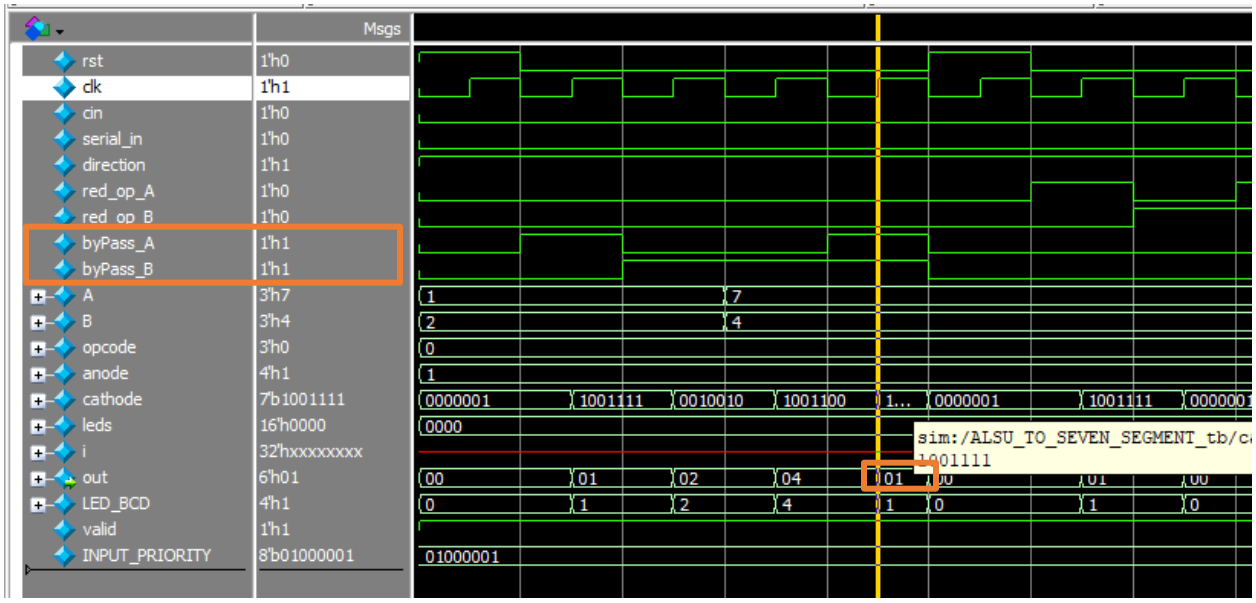
1. At start reset is high for 2 time unit as delay so output is 0,then deactivated reset and now bypass A is high so out is equal to A which is shown in the following figure also Cathode is activated by B , C led only to display one.



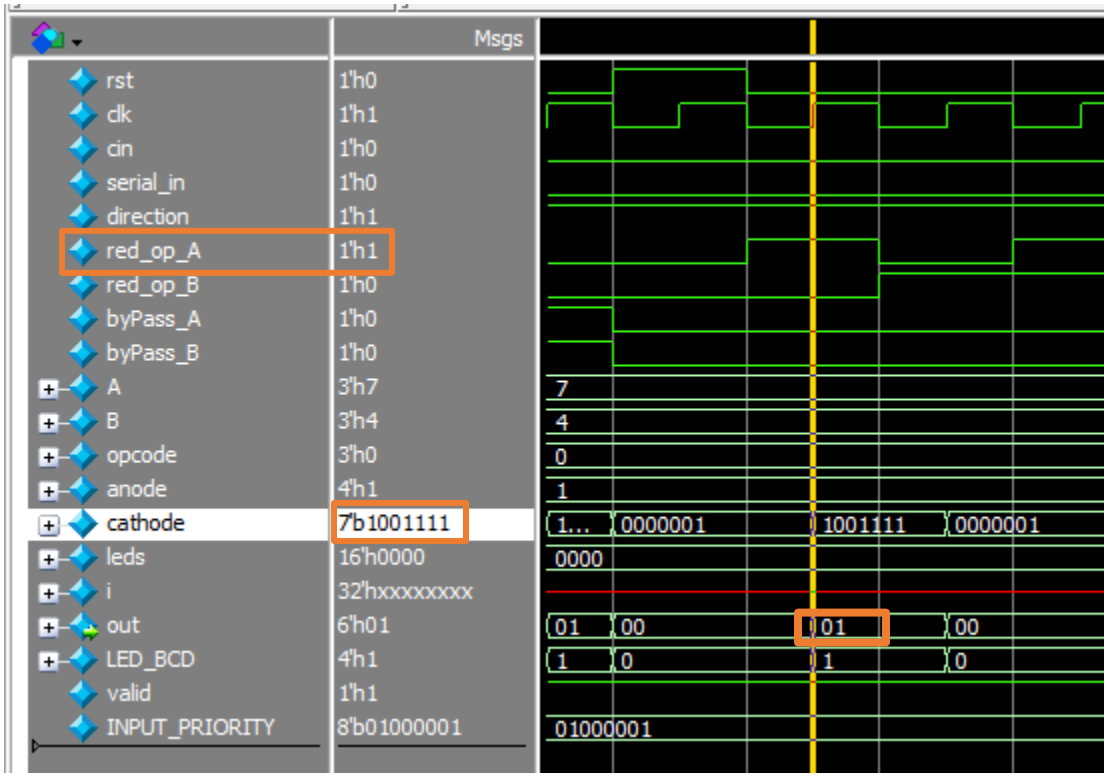
2. Here ByPass B is active then out = B which is 2 and cathode is activated by abdeg leds to display 2.



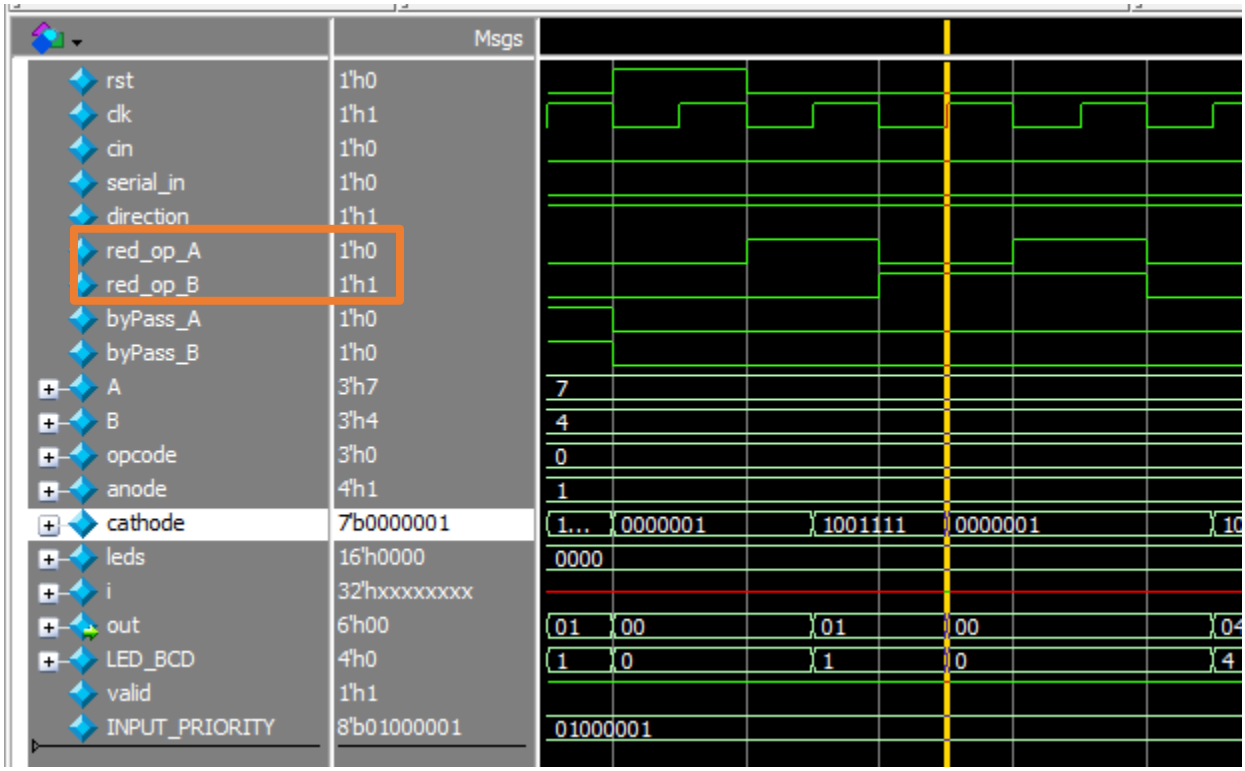
3. Here both bypass a,b are high then take the value of INPUTPRIORITY parameter only least six bits which is one.



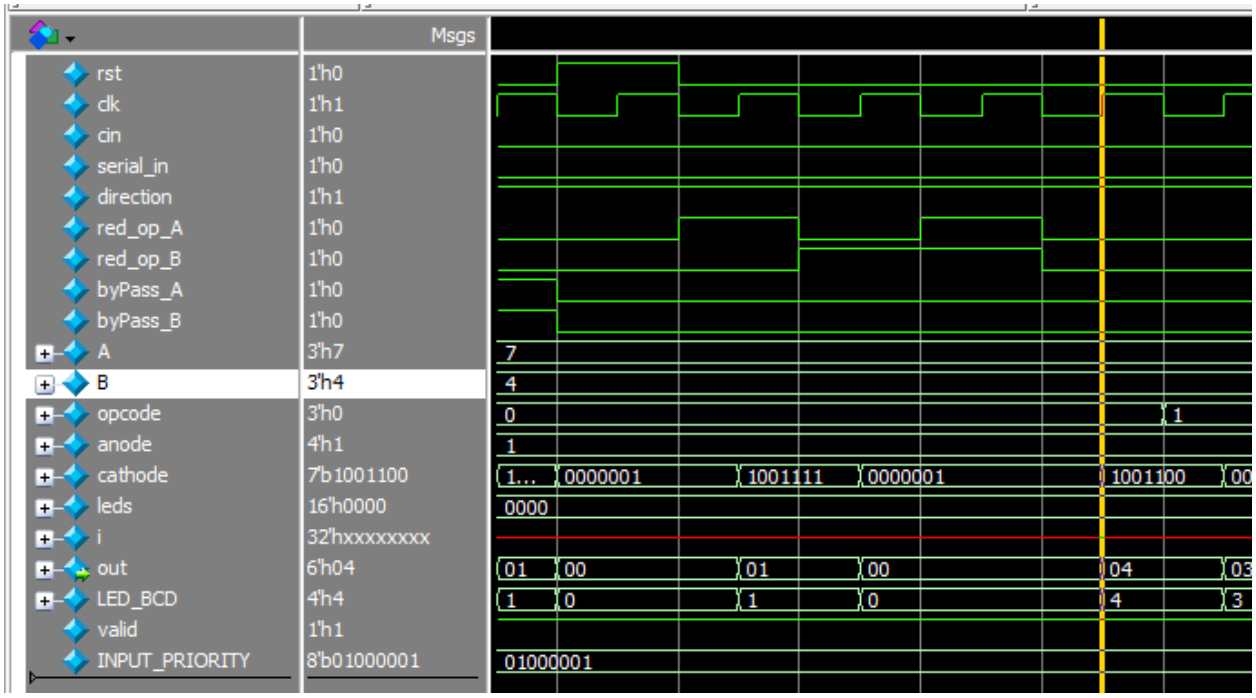
4. Reduction operation for A.



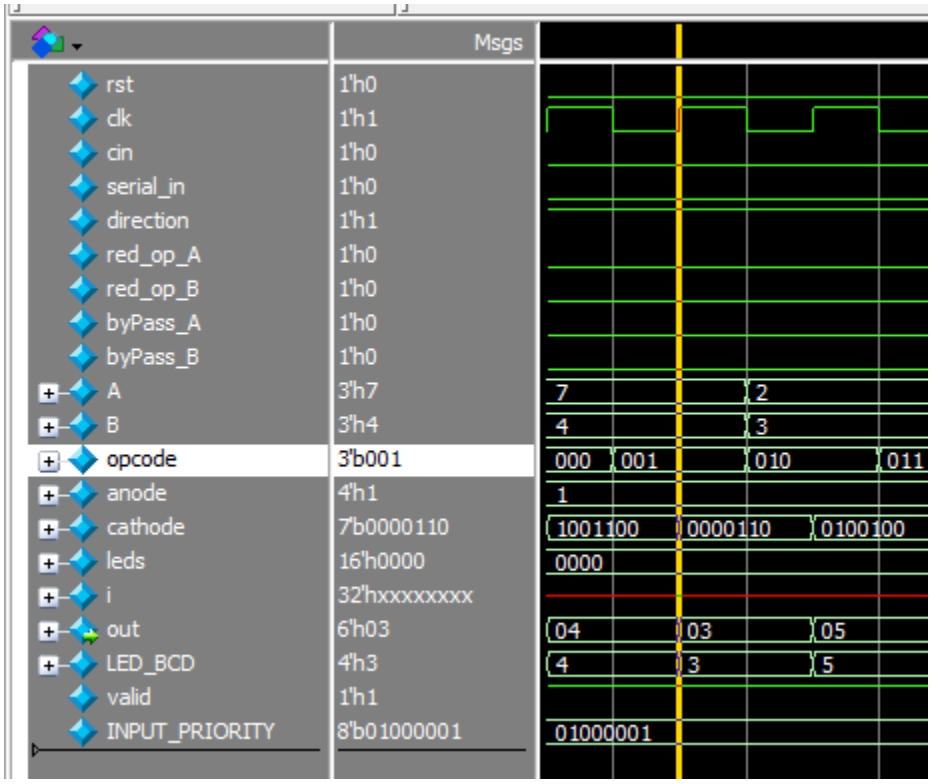
5. Reduction operation for B [4 in decimal]



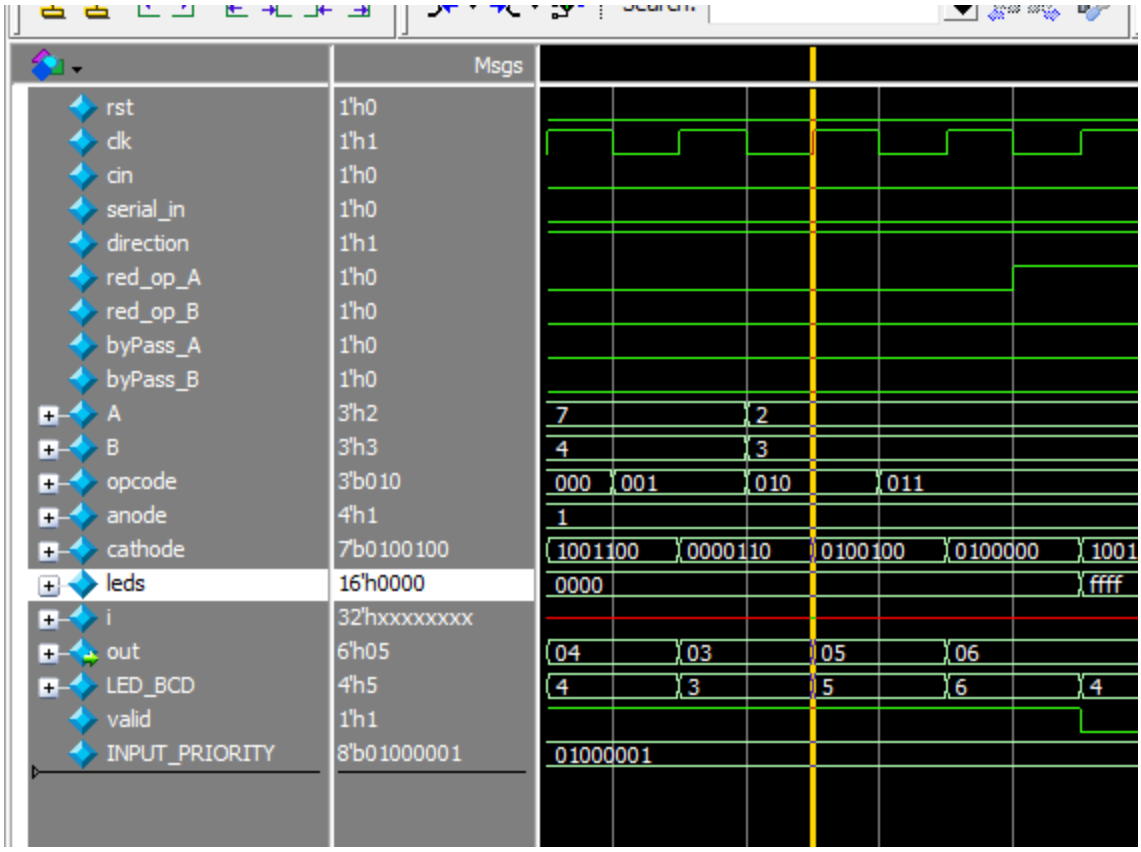
6. And operation between A (7) & B (4) = 4



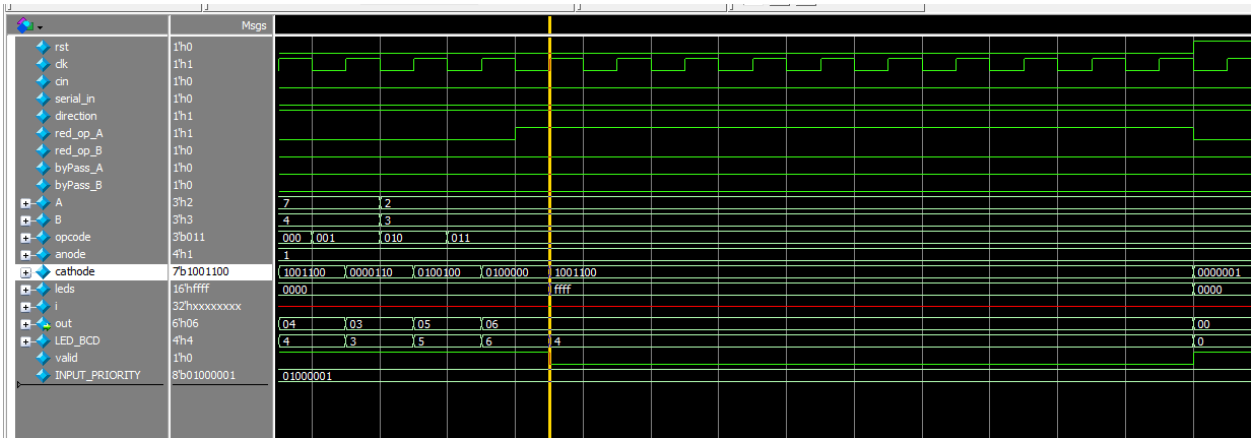
7. Opcode = 001 Xor operation between A ^ B



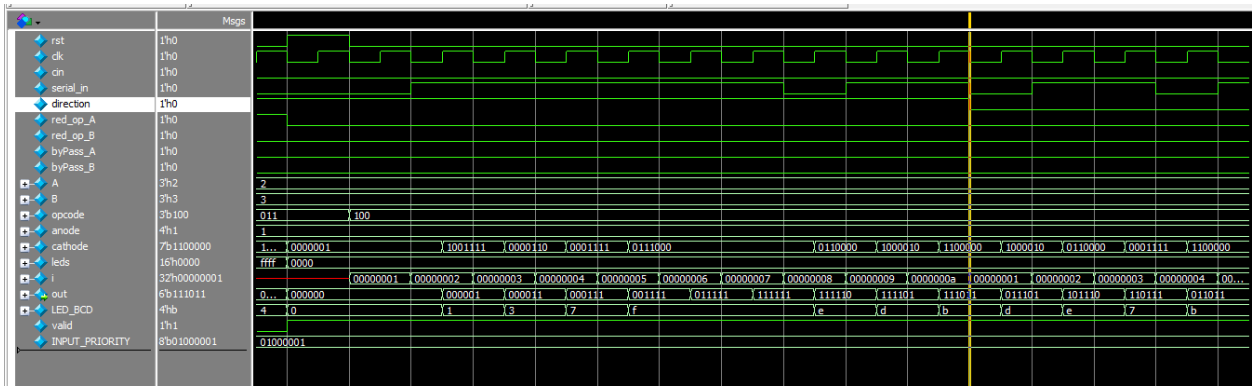
8. Addition and multiplication for A , B



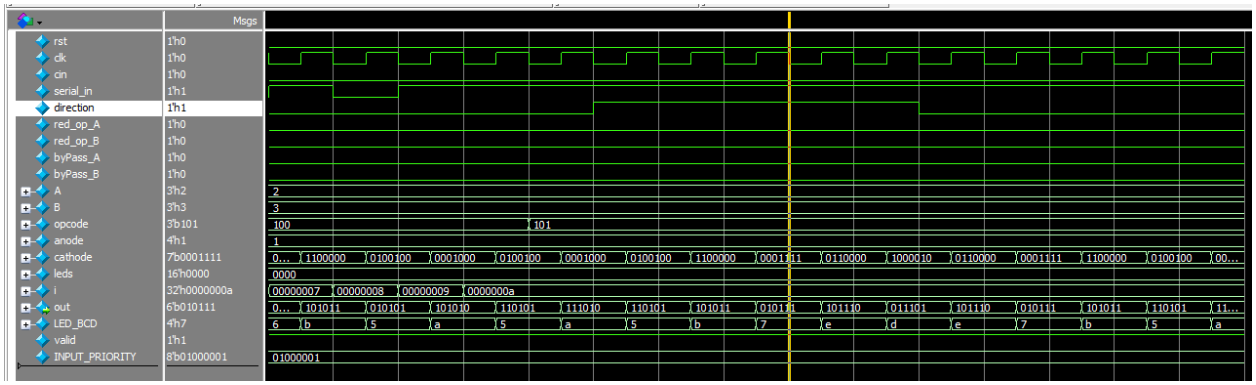
9. Invalid case opcode not logic op but red_op_A is high Then leds are blinking



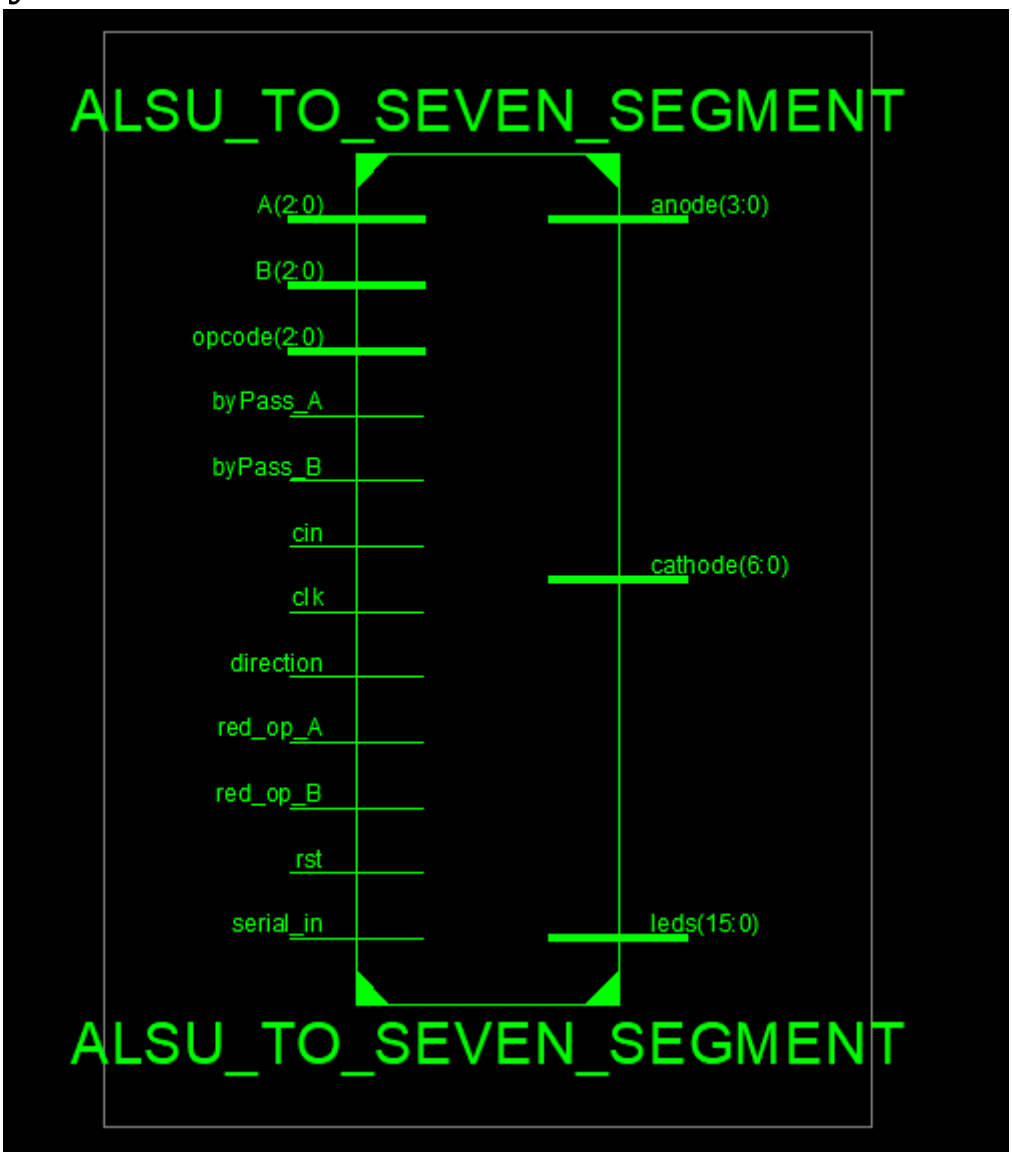
10. Shift operation before vertical yellow line is left shift and after that is shift right

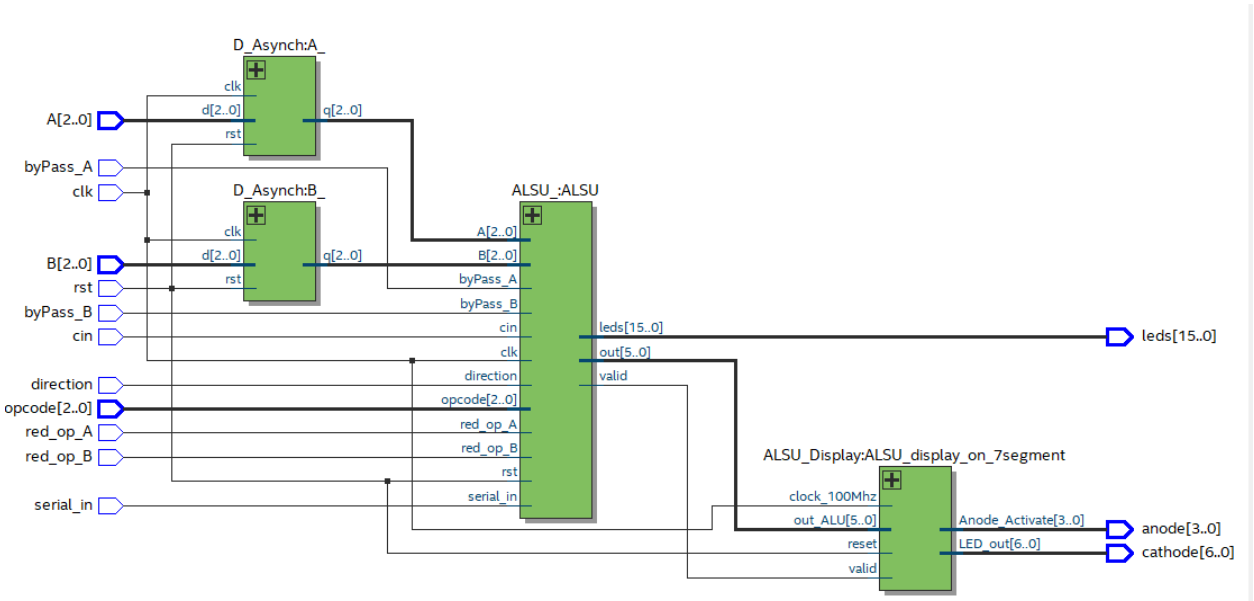


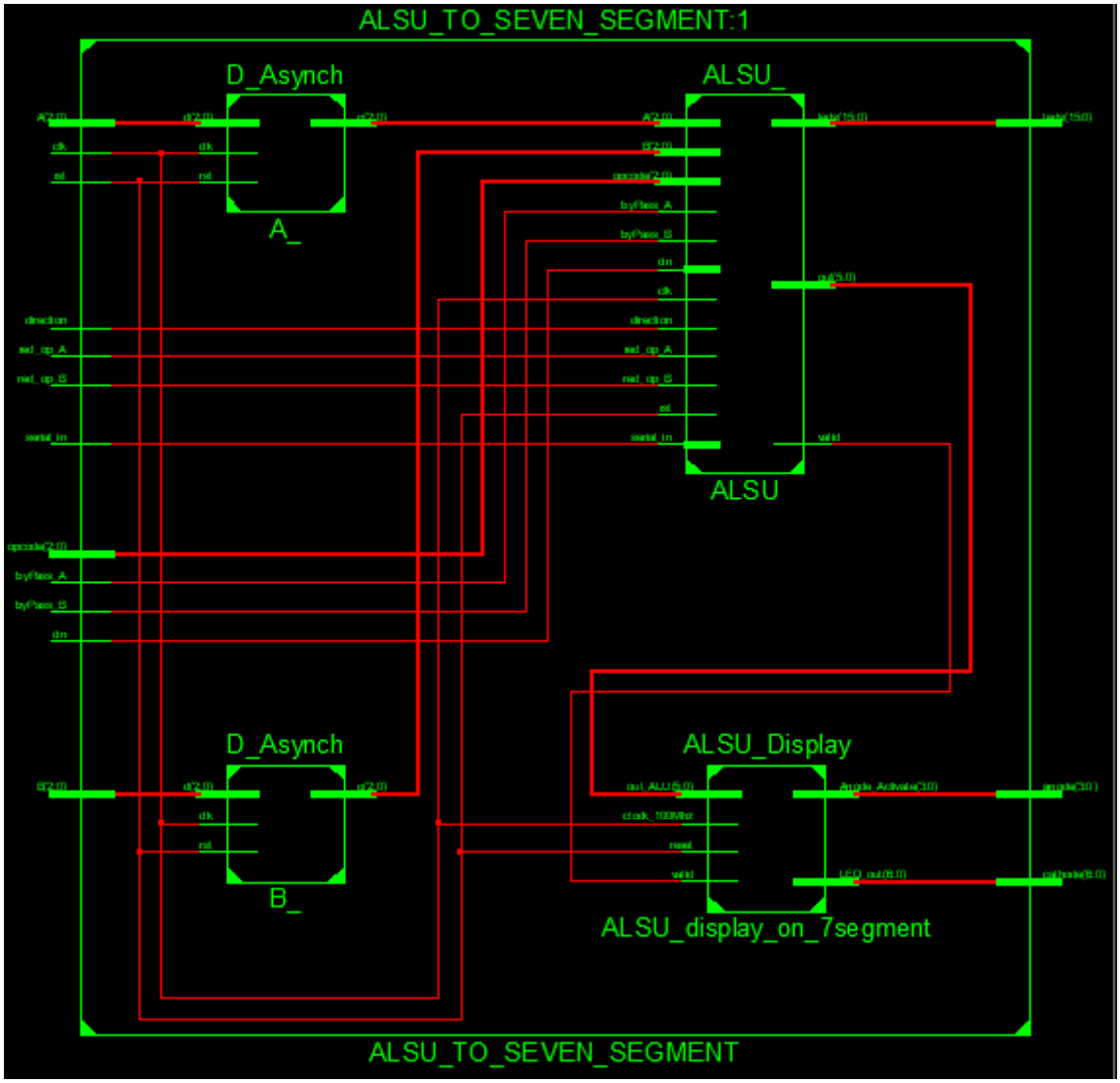
11. Finally with opcode 101 there is rotate operation where direction is high then rotate to left otherwise rotate to the right

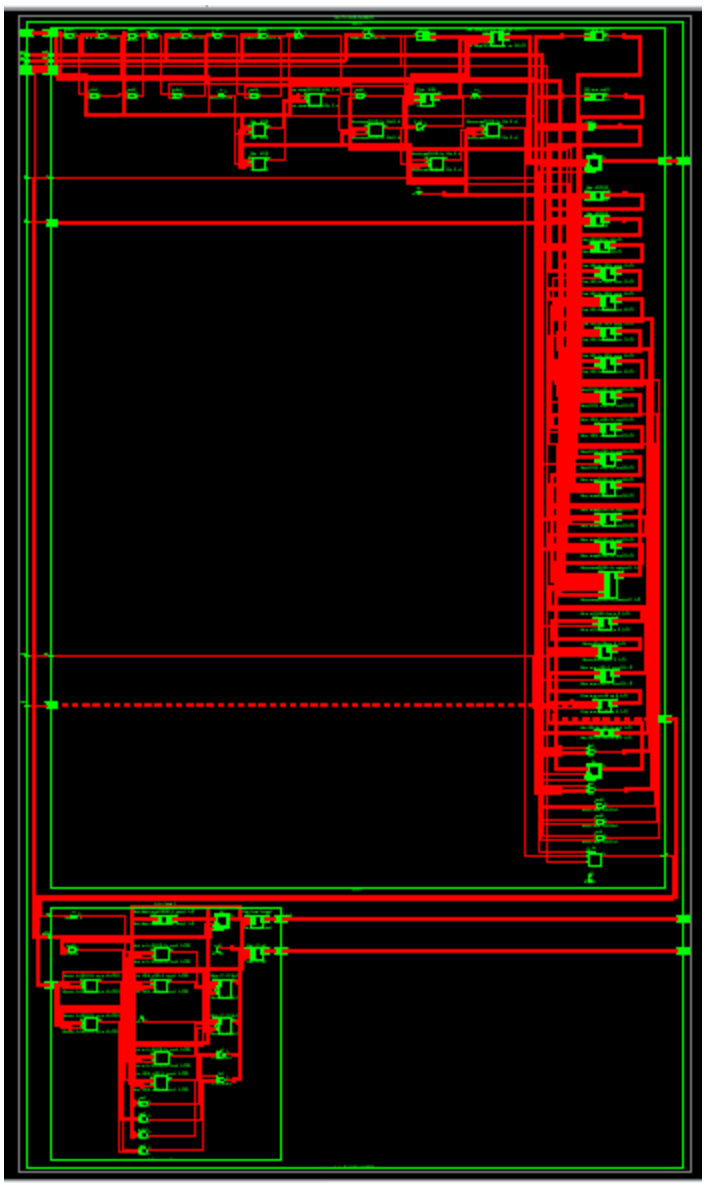


3) Synthesis :


ALSU_TO_SEVEN_SEGMENT
A(2:0)
B(2:0)
opcode(2:0)
byPass_A
byPass_B
cin
clk
direction
red_op_A
red_op_B
rst
serial_in
anode(3:0)
cathode(6:0)
leds(15:0)
ALSU_TO_SEVEN_SEGMENT


D_Asynch:A_
clk
d[2..0]
rst
q[2..0]
A[2..0]
byPass_A
clk
D_Asynch:B_
clk
d[2..0]
rst
q[2..0]
B[2..0]
rst
byPass_B
cin
direction
opcode[2..0]
red_op_A
red_op_B
serial_in
ALSU_:ALSU
A[2..0]
B[2..0]
byPass_A
byPass_B
cin
clk
direction
opcode[2..0]
red_op_A
red_op_B
rst
serial_in
leds[15..0]
out[5..0]
valid
leds[15..0]
ALSU_Display:ALSU_display_on_7segment
clock_100Mhz
out_ALU[5..0]
reset
valid
Anode_Activate[3..0]
LED_out[6..0]
anode[3..0]
cathode[6..0]


ALSU_TO_SEVEN_SEGMENT:1
D_Asynch
A_
ALSU_
ALSU
D_Asynch
B_
ALSU_Display
ALSU_display_on_7segment
ALSU_TO_SEVEN_SEGMENT



---

4) XDC :

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top
level signal names in the project

## Clock signal
```

```
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
clk]


## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports
opcode[0]]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports
opcode[1]]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports
opcode[2]]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports A[0]]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports A[1]]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports A[2]]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports B[0]]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports B[1]]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports B[2]]
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports cin]
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports red_op_A]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports red_op_B]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports byPass_A]
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports byPass_B]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports
direction]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports
serial_in]


## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports leds[0]]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports leds[1]]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports leds[2]]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports leds[3]]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports leds[4]]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports leds[5]]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports leds[6]]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports leds[7]]
set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports leds[8]]
set_property -dict { PACKAGE_PIN V3    IOSTANDARD LVCMOS33 } [get_ports leds[9]]
set_property -dict { PACKAGE_PIN W3    IOSTANDARD LVCMOS33 } [get_ports leds[10]]
set_property -dict { PACKAGE_PIN U3    IOSTANDARD LVCMOS33 } [get_ports leds[11]]
set_property -dict { PACKAGE_PIN P3    IOSTANDARD LVCMOS33 } [get_ports leds[12]]
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports leds[13]]
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports leds[14]]
```

```
set_property -dict { PACKAGE_PIN L1    IOSTANDARD LVCMOS33 } [get_ports leds[15]]


##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports
cathode[6]]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports
cathode[5]]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports
cathode[4]]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports
cathode[3]]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports
cathode[2]]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports
cathode[1]]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports
cathode[0]]

#set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports dp]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports anode[0]]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports anode[1]]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports anode[2]]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports anode[3]]


##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]


##Pmod Header JA
#set_property -dict { PACKAGE_PIN J1   IOSTANDARD LVCMOS33 } [get_ports
{JA[0]}];#Sch name = JA1
#set_property -dict { PACKAGE_PIN L2   IOSTANDARD LVCMOS33 } [get_ports
{JA[1]}];#Sch name = JA2
#set_property -dict { PACKAGE_PIN J2   IOSTANDARD LVCMOS33 } [get_ports
{JA[2]}];#Sch name = JA3
#set_property -dict { PACKAGE_PIN G2   IOSTANDARD LVCMOS33 } [get_ports
{JA[3]}];#Sch name = JA4
```

```
#set_property -dict { PACKAGE_PIN H1   IOSTANDARD LVCMOS33 } [get_ports
{JA[4]}];#Sch name = JA7
#set_property -dict { PACKAGE_PIN K2   IOSTANDARD LVCMOS33 } [get_ports
{JA[5]}];#Sch name = JA8
#set_property -dict { PACKAGE_PIN H2   IOSTANDARD LVCMOS33 } [get_ports
{JA[6]}];#Sch name = JA9
#set_property -dict { PACKAGE_PIN G3   IOSTANDARD LVCMOS33 } [get_ports
{JA[7]}];#Sch name = JA10

##Pmod Header JB
#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports
{JB[0]}];#Sch name = JB1
#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports
{JB[1]}];#Sch name = JB2
#set_property -dict { PACKAGE_PIN B15   IOSTANDARD LVCMOS33 } [get_ports
{JB[2]}];#Sch name = JB3
#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports
{JB[3]}];#Sch name = JB4
#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports
{JB[4]}];#Sch name = JB7
#set_property -dict { PACKAGE_PIN A17   IOSTANDARD LVCMOS33 } [get_ports
{JB[5]}];#Sch name = JB8
#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports
{JB[6]}];#Sch name = JB9
#set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports
{JB[7]}];#Sch name = JB10

##Pmod Header JC
#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports
{JC[0]}];#Sch name = JC1
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports
{JC[1]}];#Sch name = JC2
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports
{JC[2]}];#Sch name = JC3
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports
{JC[3]}];#Sch name = JC4
#set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports
{JC[4]}];#Sch name = JC7
#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports
{JC[5]}];#Sch name = JC8
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports
{JC[6]}];#Sch name = JC9
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports
{JC[7]}];#Sch name = JC10
```

```
##Pmod Header JXADC
#set_property -dict { PACKAGE_PIN J3   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[0]}];#Sch name = XA1_P
#set_property -dict { PACKAGE_PIN L3   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[1]}];#Sch name = XA2_P
#set_property -dict { PACKAGE_PIN M2   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[2]}];#Sch name = XA3_P
#set_property -dict { PACKAGE_PIN N2   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[3]}];#Sch name = XA4_P
#set_property -dict { PACKAGE_PIN K3   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[4]}];#Sch name = XA1_N
#set_property -dict { PACKAGE_PIN M3   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[5]}];#Sch name = XA2_N
#set_property -dict { PACKAGE_PIN M1   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[6]}];#Sch name = XA3_N
#set_property -dict { PACKAGE_PIN N1   IOSTANDARD LVCMOS33 } [get_ports
{JXADC[7]}];#Sch name = XA4_N


##VGA Connector
#set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[0]}]
#set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[1]}]
#set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[2]}]
#set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports
{vgaRed[3]}]
#set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[0]}]
#set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[1]}]
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[2]}]
#set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports
{vgaBlue[3]}]
#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[0]}]
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[1]}]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[2]}]
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports
{vgaGreen[3]}]
#set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports Hsync]
```

```
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports Vsync]


##USB-RS232 Interface
#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports RsRx]
#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports RsTx]


##USB HID (PS/2)
#set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33   PULLUP true }
[get_ports PS2Clk]
#set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33   PULLUP true }
[get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using
the
##STARTUPE2 primitive.
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports
{QspiDB[0]}]
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports
{QspiDB[1]}]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports
{QspiDB[2]}]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports
{QspiDB[3]}]
#set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports QspiCSn]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
```

5) Sources Files:

https://drive.google.com/drive/folders/1yqP9wNd3dxJazNdYnQVOibtiw5X8h21X?usp=sharing

Thanks