Lab 9

Top module

```
module top(
    input clk,
    input clr,
    input go,
    input [3:0] x,
    input [3:0] y,
    output [7:0] cathode,
    output [3:0] anode
    );

    wire [3:0] gcd;
    wire clk_190,clk_25;
    wire [3:0] thousands,hundreds,tens,ones;
    wire xld,yld,gld,eqflg,ltflg,xmsel,ymsel;
    wire button,reset,bt_go;

    frq_div #(20) fun1(clk,clk_190);
    clk_wiz fun2(clk_25,clk);

    debounce fun3(clk_190,reset,clr);
    debounce fun4(clk_190,bt_go,go);

    pulse fun5(clk_25,button,bt_go);
```

```
    controlpath
fun6(reset,clk_25,button,eqflg,ltflg,ymsel,xmsel,xld,yld,gld);
    datapath
fun7(clk_25,reset,xmsel,ymsel,xld,yld,gld,x,y,gcd,eqflg,ltflg);

    bin2bcd fun8(gcd,thousands,hundreds,tens,ones);
    sevenseg
fun9(clk,reset,ones,tens,hundreds,thousands,cathode,anode);

endmodule
```

Frequency divider

```
module frq_div(
    input clk,
    output div_clk
    );

parameter n=25;

reg [n-1:0] q=0;

always @ (posedge clk)
begin

q<=q+1;

end
```

```
assign div_clk=q[n-1];

endmodule
```

Debounce

```
module debounce(
    input clk,
    output deb_out,
    input deb_in
    );

    reg D1,D2,D3;
    always@(posedge clk)
    begin
        D1<=deb_in;
        D2<=D1;
        D3<=D2; // inversion is done to generate a pulse
    end

    assign deb_out=D1&&D2&&D3;

endmodule
```

Pulse

```
module pulse(
    input clk,
    output deb_out,
    input deb_in
    );

    reg D1,D2,D3;
    always@(posedge clk)
    begin
        D1<=deb_in;
        D2<=D1;
        D3<=~D2; // inversion is done to generate a pulse
    end

    assign deb_out=D1&&D2&&D3;

endmodule
```

Control Path

```
module controlpath(
    input clr,
    input clk,
    input go,
```

```
    input eqflg,
    input ltflg,
    output reg ymsel=0,
    output reg xmsel=0,
    output reg xld=0,
    output reg yld=0,
    output reg gld=0
    );

    parameter [2:0]
start=3'd0,inp=3'd1,test1=3'd2,done=3'd3,test2=3'd4,update1=3'd
5,update2=3'd6;
    reg [2:0] ps=start;
    reg [2:0] ns=start;

    always @ (*)
    begin
        case(ps)

            start:  begin
                    if(go)
                        ns<=inp;
                    else
                        ns<=start;

                   xmsel<=0;
                   ymsel<=0;
                   xld<=0;
                   yld<=0;
```

```
        gld<=0;

      end
inp:   begin
         ns<=test1;

        xmsel<=1;
        ymsel<=1;
        xld<=1;
        yld<=1;

       end

test1:  begin

        if(eqflg)
           ns<=done;
        else
           ns<=test2;

       end

test2:  begin

        if(ltflg)
           ns<=update1;
        else
           ns<=update2;
```

```
        end

update1: begin

        ns<=test1;

        ymsel<=0;
        yld<=1;

        end

update2: begin
        ns<=test1;

        xmsel<=0;
        xld<=1;

        end

done:  begin

      if(clr==1'b1)
          ns<=start;
      else
          ns<=done;

      gld<=1;

      end
```

```
            default: ns<= start;

            endcase
    end

    always @ (posedge clk or posedge clr)
    begin
        if(clr)
            ps<=start;
        else
            ps<=ns;
    end

endmodule
```

Data Path

```
module datapath(
    input clk,
    input clr,
    input xmsel,
    input ymsel,
    input xld,
    input yld,
    input gld,
    input [3:0] xin,
    input [3:0] yin,
```

```
output reg [3:0] gcd,
output reg eqflg,
output reg ltflg
);

reg [3:0] x;
reg [3:0] y;
reg [3:0] x1;
reg [3:0] y1;
wire [3:0] xmy;
wire [3:0] ymx;

assign xmy=x-y;
assign ymx=y-x;

always @ (*)
begin
    if(x==y)
        eqflg<=1'b1;
    else
        eqflg<=1'b0;
end

always @ (*)
begin
    if(x<y)
        ltflg<=1'b1;
    else
        ltflg<=1'b0;
```

```
end

always @ (posedge clk or posedge clr)
begin
    if(clr==1'b1)
        gcd<=1'b0;
    else
        if(gld)
            gcd<=x;
end

always @ (*)
begin
    if(xmsel==1'b1)
        x1<=xin;
    else
        x1<=xmy;
end

always@(*)
begin
    if(ymsel==1)
        y1<=yin;
    else
        y1<=ymx;
end

always@(posedge clk or posedge clr)
begin
```

```
        if(xld==1)
            x=x1;
        else if(clr==1)
            x<=0;
    end

    always@(posedge clk or posedge clr)
    begin
        if(yld)
            y=y1;
        else if(clr==1)
            y<=0;
    end

endmodule
```

Bin to BCD

```
module bin2bcd(number, thousands, hundreds, tens, ones);
  // I/O Signal Definitions
  input  [3:0] number;
  output reg [3:0] thousands;
  output reg [3:0] hundreds;
  output reg [3:0] tens;
  output reg [3:0] ones;

  // Internal variable for storing bits
  reg [19:0] shift;
```

```
  integer i;

  always @(number)
  begin
    // Clear previous number and store new number in shift
register
    shift[19:8] = 0;
    shift[7:0] = number;

    // Loop eight times
    for (i=0; i<8; i=i+1) begin
      if (shift[11:8] >= 5)
        shift[11:8] = shift[11:8] + 3;

      if (shift[15:12] >= 5)
        shift[15:12] = shift[15:12] + 3;

      if (shift[19:16] >= 5)
        shift[19:16] = shift[19:16] + 3;

      // Shift entire register left once
      shift = shift << 1;
    end

    // Push decimal numbers to output
    hundreds = shift[19:16];
    tens     = shift[15:12];
    ones     = shift[11:8];
    thousands= 4'b0000
```

```
   end
endmodule
```

Seven Segment

```
`timescale 1ns / 1ps
module sevenseg(
    input clk,
    input clr,              // clear pin used to reset the LED display
    input [3:0] ones,
    input [3:0] tens,
    input [3:0] hundreds,
    input [3:0] thousands,
    output [7:0] cathode,
    output [3:0] anode
    );
     reg [6:0]sseg_temp;
     reg [3:0]an_temp = 4'b1110 ;
     reg [17:0] count; //the 18 bit counter which allows us to
multiplex at 1000Hz

     always @ (posedge clk)
      begin
        count <= count + 1;
      end
```

```
        //code for display multiple digits (do not initialize an_temp in
line 41 and comment out lines 44 to 60)
    always @ (*)
     begin
      case(count[17:16]) //using only the 2 MSB's of the counter

       2'b00 :  //When the 2 MSB's are 00 enable the fourth display
        begin
         case(ones)
              4'd0 : sseg_temp = 7'b0000001; //to display 0
              4'd1 : sseg_temp = 7'b1001111; //to display 1
              4'd2 : sseg_temp = 7'b0010010; //to display 2
              4'd3 : sseg_temp = 7'b0000110; //to display 3
              4'd4 : sseg_temp = 7'b1001100; //to display 4
              4'd5 : sseg_temp = 7'b0100100; //to display 5
              4'd6 : sseg_temp = 7'b0100000; //to display 6
              4'd7 : sseg_temp = 7'b0001111; //to display 7
              4'd8 : sseg_temp = 7'b0000000; //to display 8
              4'd9 : sseg_temp = 7'b0000100; //to display 9
              default : sseg_temp = 7'b1111110; //dash
          endcase

         //sseg = ones;
         an_temp = 4'b1110;
        end

       2'b01:  //When the 2 MSB's are 01 enable the third display
        begin
           case(tens)
```

```
        4'd0 : sseg_temp = 7'b0000001; //to display 0
        4'd1 : sseg_temp = 7'b1001111; //to display 1
        4'd2 : sseg_temp = 7'b0010010; //to display 2
        4'd3 : sseg_temp = 7'b0000110; //to display 3
        4'd4 : sseg_temp = 7'b1001100; //to display 4
        4'd5 : sseg_temp = 7'b0100100; //to display 5
        4'd6 : sseg_temp = 7'b0100000; //to display 6
        4'd7 : sseg_temp = 7'b0001111; //to display 7
        4'd8 : sseg_temp = 7'b0000000; //to display 8
        4'd9 : sseg_temp = 7'b0000100; //to display 9
        default : sseg_temp = 7'b1111110; //dash
      endcase

  //sseg = tens;
  an_temp = 4'b1101;
 end

 2'b10:  //When the 2 MSB's are 10 enable the second display
  begin
   case(hundreds)
        4'd0 : sseg_temp = 7'b0000001; //to display 0
        4'd1 : sseg_temp = 7'b1001111; //to display 1
        4'd2 : sseg_temp = 7'b0010010; //to display 2
        4'd3 : sseg_temp = 7'b0000110; //to display 3
        4'd4 : sseg_temp = 7'b1001100; //to display 4
        4'd5 : sseg_temp = 7'b0100100; //to display 5
        4'd6 : sseg_temp = 7'b0100000; //to display 6
        4'd7 : sseg_temp = 7'b0001111; //to display 7
        4'd8 : sseg_temp = 7'b0000000; //to display 8
```

```
        4'd9 : sseg_temp = 7'b0000100; //to display 9
        default : sseg_temp = 7'b1111110; //dash
     endcase

    //sseg = hundreds;
    an_temp = 4'b1011;
   end

  2'b11:  //When the 2 MSB's are 11 enable the first display
   begin
    case(thousands)
        4'd0 : sseg_temp = 7'b0000001; //to display 0
        4'd1 : sseg_temp = 7'b1001111; //to display 1
        4'd2 : sseg_temp = 7'b0010010; //to display 2
        4'd3 : sseg_temp = 7'b0000110; //to display 3
        4'd4 : sseg_temp = 7'b1001100; //to display 4
        4'd5 : sseg_temp = 7'b0100100; //to display 5
        4'd6 : sseg_temp = 7'b0100000; //to display 6
        4'd7 : sseg_temp = 7'b0001111; //to display 7
        4'd8 : sseg_temp = 7'b0000000; //to display 8
        4'd9 : sseg_temp = 7'b0000100; //to display 9
        default : sseg_temp = 7'b1111110; //dash
    endcase

    //sseg = thousands;
    an_temp = 4'b0111;
   end
 endcase
end
```

```
    assign anode = an_temp;
    assign cathode = {sseg_temp, 1'b1};
 Endmodule
```

XDC File

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {x[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {x[0]}]
set_property PACKAGE_PIN V16 [get_ports {x[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {x[1]}]
set_property PACKAGE_PIN W16 [get_ports {x[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {x[2]}]
set_property PACKAGE_PIN W17 [get_ports {x[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {x[3]}]
set_property PACKAGE_PIN W15 [get_ports {y[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {y[0]}]
```

```
set_property PACKAGE_PIN V15 [get_ports {y[1]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {y[1]}]
set_property PACKAGE_PIN W14 [get_ports {y[2]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {y[2]}]
set_property PACKAGE_PIN W13 [get_ports {y[3]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {y[3]}]

#7 segment display
    set_property PACKAGE_PIN W7 [get_ports {cathode[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[7]}]
    set_property PACKAGE_PIN W6 [get_ports {cathode[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[6]}]
    set_property PACKAGE_PIN U8 [get_ports {cathode[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[5]}]
    set_property PACKAGE_PIN V8 [get_ports {cathode[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[4]}]
    set_property PACKAGE_PIN U5 [get_ports {cathode[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[3]}]
    set_property PACKAGE_PIN V5 [get_ports {cathode[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[2]}]
```

```
   set_property PACKAGE_PIN U7 [get_ports {cathode[1]}]
       set_property IOSTANDARD LVCMOS33 [get_ports
{cathode[1]}]
   set_property PACKAGE_PIN V7 [get_ports cathode[0]]
       set_property IOSTANDARD LVCMOS33 [get_ports
cathode[0]]

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

##Buttons
set_property PACKAGE_PIN U18 [get_ports clr]

       set_property IOSTANDARD LVCMOS33 [get_ports clr]
set_property PACKAGE_PIN T18 [get_ports go]

       set_property IOSTANDARD LVCMOS33 [get_ports go]
```

Output

x=2
y=4

gcd=2