University of Engineering and Technology, Peshawar

Department of Computer Systems Engineering.

Course Lab: CSE-308 Digital System Design

Zubair Khan	**19** PWCSE **1797**
Section	A
Batch	21 (Spring_2022)
Submitted to	Engr. Madiha Sher

LAB 06 Title

4-Digit BCD Counter on Multiplexed Seven Segment Display

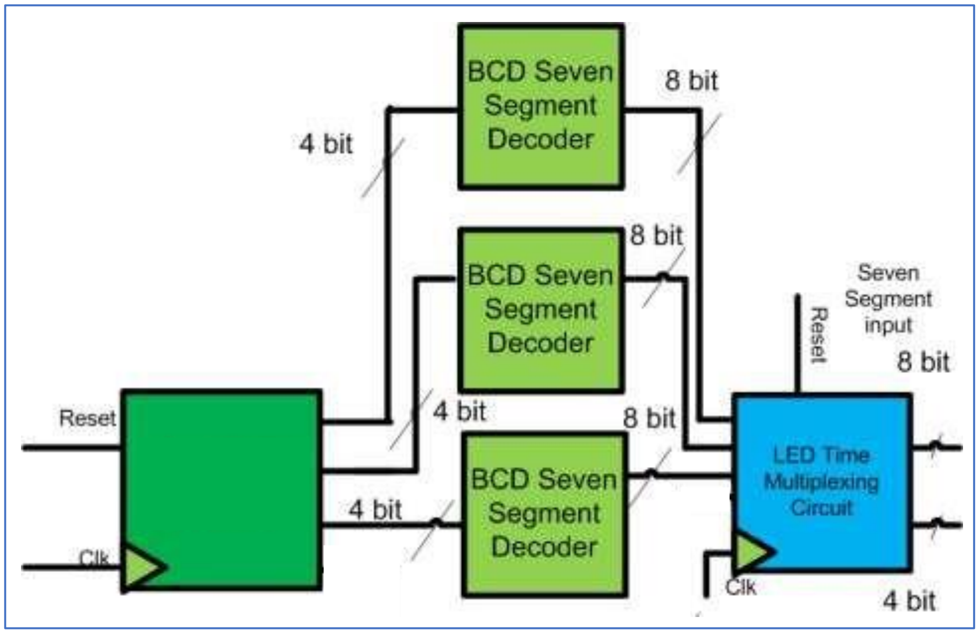
Objectives(s):

This lab will enable students to:

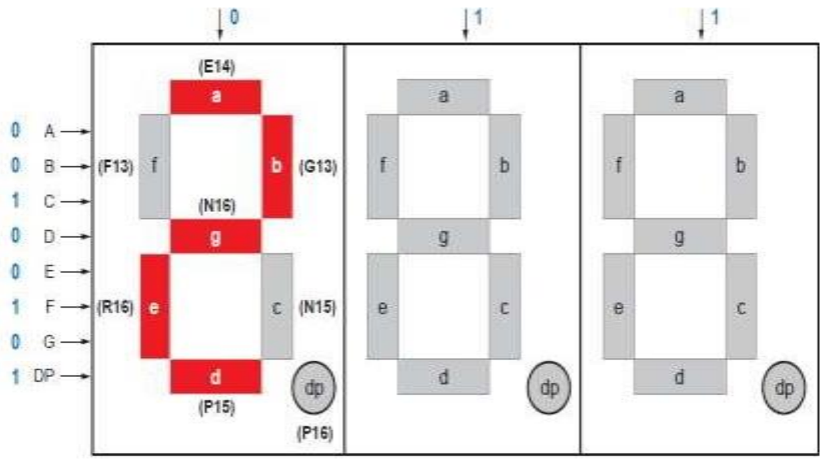
- Learn top down and bottom up design methodologies
- Learn to use time multiplexed 4-digit Seven Segment display

Theory

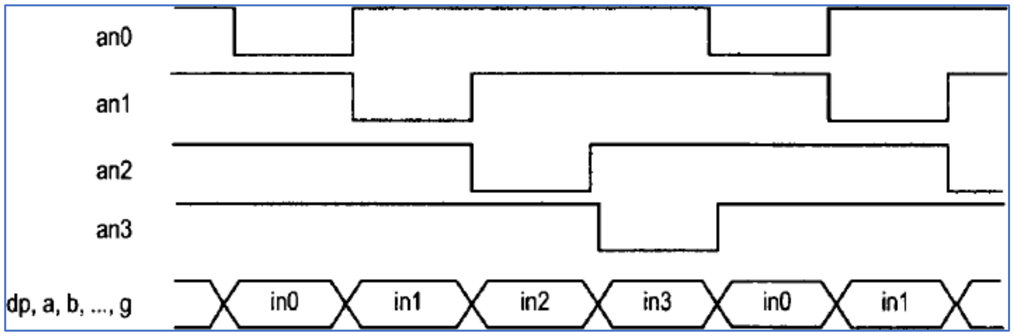
Block Diagram: The S6 board has four seven-segment LED displays, each containing seven bars and one small round dot. To reduce the use of FPGA's I/0 pins, the S6 board uses a time-multiplexing sharing scheme. In this scheme, the three displays have their individual enable signals but share eight common signals to light the segments. All signals are active low (i.e., enabled when a signal is 0). The schematic of displaying a "2" on the leftmost LED is shown in Figure.


BCD Seven Segment Decoder
BCD Seven Segment Decoder
BCD Seven Segment Decoder
4 bit
4 bit
4 bit
8 bit
8 bit
8 bit
Reset
Clk
Reset
Seven Segment input
8 bit
LED Time Multiplexing Circuit
Clk
4 bit

Note that the enable signal (i.e., an) is "011". This configuration clearly can enable only one display at a time. We can time-multiplex the four LED patterns by enabling the four displays in turn, as shown in the simplified timing diagram. If the refreshing rate of the enable signal is fast enough, the human eye cannot distinguish the on and off intervals of the LEDs and perceives that all four displays are lit simultaneously. This scheme reduces the number of I/O pins from 32 to 11 (i.e., eight LED segments plus three enable signals) but requires a time-multiplexing circuit.



3 Digit BCD Counter and timing diagram of LED multiplexing



LED time Multiplexing

The refresh rate of the enable signal has to be fast enough to fool our eyes but should be slow enough so that the LEDs can be turned on and off completely. The rate around the range 1000 Hz should work properly. In our design, we use an 18-bit binary counter for this purpose. The two MSBs are decoded to generate the enable signal and are used as the selection signal for multiplexing

LAB TASK

Implement a BCD counter that runs from 000 to 999 and shows each BCD digit on the seven-segment display.

Explanation:

In this LAB we have don two tasks 6a and 6b

- **6a:** Implementation of BCD counter that runs from 0 to 9 and shows each BCD digit on the seven-segment display
- **6b:** Implementation of BCD counter that runs from 000 to 999 and shows each BCD digit on the seven-segment display

6a: Implementation of BCD counter that runs from 0 to 9 and shows each BCD digit on the seven-segment display

Source Code

For [*. v] file (Verilog)

```
module counter(
    input clk, rst,
    output [2:0]en,
    output [6:0] counter7seg
    );
    reg [3:0] counter;
    assign en = 3'b011;

    seven_seg seg(counter, counter7seg);
    clk_divider Div_Clock(clk, clk1, rst);

    always @ (posedge clk1)
    begin
        if(rst)
            counter = 0;
        else
            begin
                counter = counter + 1'b1;
                if(counter == 10)
                    counter = 0;
            end
    end
endmodule


module seven_seg(
    input [3:0] in,
    output [6:0] out
);
    assign out = (in == 0) ? 7'd0:
                 (in == 1) ? 7'b1111001:
                 (in == 2) ? 7'b0100100:
                 (in == 3) ? 7'b0110000:
                 (in == 4) ? 7'b0011001:
                 (in == 5) ? 7'b0010010:
                 (in == 6) ? 7'b0000010:
                  (in == 7) ? 7'b1111000:
                  (in == 8) ? 7'b0000000:
                  (in == 9) ? 7'b0011000: 7'b1111111;
endmodule

module clk_divider(clk, clk1, rst);
    integer cnt;
    input clk, rst;
    output reg clk1;
    always @(posedge clk)
        begin
            if(rst)
                begin
                    cnt = 0;
                    clk1 = 0;
                end
            else
                begin
                    cnt = cnt + 1;
                    if(cnt == 10000000)
                        begin
                            clk1 = ~clk1;
                            cnt = 0;
                        end
                end
        end
endmodule
```

6b: Implementation of BCD counter that runs from 000 to 999 and shows each BCD digit on the seven-segment display

Source Code

For [*.v] file (verilog)

```
module counter(
    input clk, rst,
    output [2:0]en,
    output [6:0] counter7seg
    );

    reg [3:0] U,T,H;

    seven_seg seg_7(U,T,H, en, clk, rst, counter7seg);
    clk_divider Div_Clock(clk, clk1, rst);

    always @ (posedge clk1)
    if(rst)
    begin
            U = 0;
            T = 0;
            H = 0;
   end
    else
    begin
        U = U + 1'b1;
        if(U == 10)
        begin
            U = 0;
            T = T + 1'b1;
            if(T == 10)
            begin
                T = 0;
                H = H + 1'b1;
                if(H==10)
                    H = 0;
            end
        end
  end

endmodule
```

```
module seven_seg(U,T,H, en, clk, rst, out);
input [3:0]U,T,H;
output reg [2:0]en;
input clk,rst;
output [6:0]out;
reg[3:0]in;
    assign out = (in == 0) ? 7'd0:
                 (in == 1) ? 7'b1111001:
                 (in == 2) ? 7'b0100100:
                 (in == 3) ? 7'b0110000:
                 (in == 4) ? 7'b0011001:
                 (in == 5) ? 7'b0010010:
                 (in == 6) ? 7'b0000010:
                 (in == 7) ? 7'b1111000:
                 (in == 8) ? 7'b0000000:
                 (in == 9) ? 7'b0011000: 7'b1111111;

  always @(posedge clk)
  begin
      if(rst)
          in = U;
      else begin
          if(en == 3'b110) begin
              en = 3'b101;
              in = T;
          end
          else begin
              if(en == 3'b101)begin
                  en = 3'b 011;
                  in = H;
              end
          end
      end
  end
endmodule


module clk_divider(clk, clk1, rst);
    integer cnt;
    input clk, rst;
    output reg clk1;
    always @(posedge clk)
        begin
            if(rst)
                begin
                    cnt = 0;
```

```verilog
                    clk1 = 0;
                end
            else
                begin
                    cnt = cnt + 1;
                    if(cnt == 100000000)
                        begin
                            clk1 = ~clk1;
                            cnt = 0;
                        end
                end
        end
endmodule
```

For [*.ucf] file (verilog)

```
NET "clk" LOC = V10;
NET "rst" LOC = F17;
NET "en[2]" LOC = B3;
NET "en[1]" LOC = A2;
NET "en[0]" LOC = B2;
NET "counter7seg[6]" LOC = A3;
NET "counter7seg[5]" LOC = B4;
NET "counter7seg[4]" LOC = A4;
NET "counter7seg[3]" LOC = C4;
NET "counter7seg[2]" LOC = C5;
NET "counter7seg[1]" LOC = D6;
NET "counter7seg[0]" LOC = A5;
NET "counter7seg[6]" SLEW = FAST;
NET "counter7seg[5]" SLEW = FAST;
NET "counter7seg[4]" SLEW = FAST;
NET "counter7seg[3]" SLEW = FAST;
NET "counter7seg[2]" SLEW = FAST;
NET "counter7seg[1]" SLEW = FAST;
NET "counter7seg[0]" SLEW = FAST;
# PlanAhead Generated IO constraints

NET "en[2]" SLEW = FAST;
NET "en[1]" SLEW = FAST;
NET "en[0]" SLEW = FAST;
```