EC311 Introduction to Logic Design Spring 2017
Lab 4: Seven-Segment Display
Clock Dividers and Display Drivers

Goals

- To get acquainted with the data encoding/decoding (binary-to-7Segment).
- To get familiar with clock dividers.

1. Overview

In this lab you will design a 7-segment display driver. The driver will then be used to display the results of the counter from the previous lab on the FPGA board's 7-segment display.

The design will have two modes, one of which would be selected via a control switch:

1. Automatic counting: the counter should increment automatically with a step of 1 second.
2. Manual counting: the counter should increment on a push of a button.

Both possible triggers for the count up will be fed into a multiplexer, controlled by the above switch.

Your design will have multiple clock domains:

- Slow clock (~1Hz): to use as an increment trigger.
- Fast clock (~1kHz): to iterate through different 7-Segment displays.
- Normal clock (100MHz): controls everything else (debouncer, clock divider, etc).

Tasks

1. Binary to 7-Segment Decoder

A 7-Segment Display is composed of 7 LEDs (marked CA, CB, …, CG on your board). The four 7-segment displays on the Nexys-3 board are common anode, meaning that the LEDs will light up when you set the input to 0 and turn off when you set the input to 1 (active low inputs). The 4 digits on the display are controlled by the corresponding active low AN line (AN0, …, AN3). Only one AN line should be set to 0 at a time, such that only one of the four 7-segment displays is ON at any given time.

For more detail on the 7-Segment Display, refer to pages 18 and 19 of http://digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf.

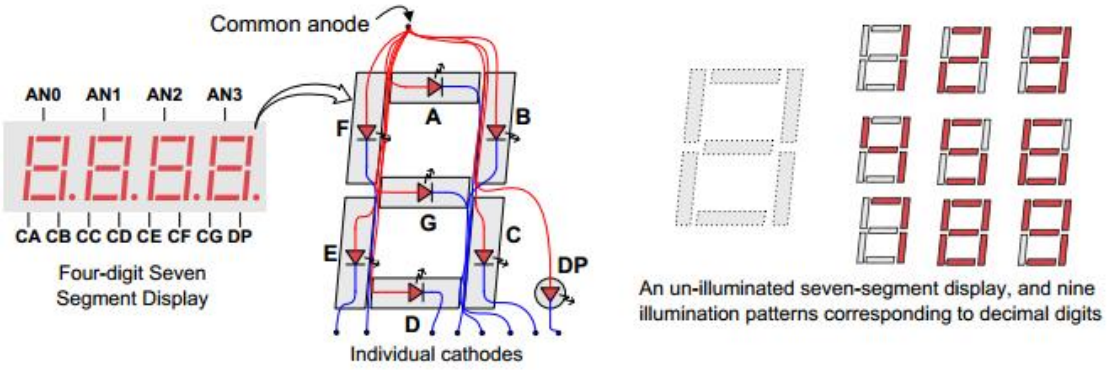

Figure 1: 7-Segment Display

Figure 1 above shows the connectivity for the 7-segment display. In this lab, you will be displaying the digits in HEX. For example, to display the number 1, segments B and C should be turned ON, i.e set to zero, and all the other segments should be turned OFF, i.e. set to one.

Implement a binary to 7-segment display decoder. The input is a 4-bit binary number (representing one hexadecimal digit), and the output is the seven segments of the display A-G in Figure 1. Your Verilog code can include a case statement with one-to-one mapping of input to output. An example how to encode digit F is shown below:

```
module binary_to_7segment(bin, seven);
        input [3:0] bin;
        output reg [6:0] seven;     // Assume that MSB is G and LSB is A

        always @(bin)
        begin
                case (bin)
                        4'b0000:    // Some code here
                        4'b0001:    // and here
                        // …….
                        4'b1110:    // and here
                        default:    seven = 7'b0001110
                                    // This will show F
                                    // Remember: 0 means "light up"
                endcase
        end
endmodule
```

The port numbers of all the seven segments can be found on the FPGA itself. For example, CA: <T17> (meaning segment A is at port T17). Therefore, the signal seven[0] is to be assigned to port T17.

2. 16-bit Counter

We would like to use the four 7-segment displays to display a 4-digit hexadecimal number, which is 16-bit binary number. That is, the 4 least significant bits will be displayed on the right-most 7-segment display, and the 7 most significant bits will be displayed on the left-most 7-segment display. The number to be displayed will be the output of the counter from the previous lab. Modify your counter (and debouncer) so that it will now output a 16-bit number.

3. Driving Four Displays

The binary to 7-segment display decoder from task 1 can drive one 7-segment display at a time, while we would like to display a different digit on each of the four 7-segment displays. We will use the decoder to drive each of the four displays in a round-robin fashion. To do so, we will make the displays flash each digit very fast (~1kHz) – so fast, that the user would not notice that the digits are not being displayed concurrently.

In order to do this, you need to apply the codes 4'b1110, 4'b1101, 4'b1011, and 4'b0111 to the AN line. This will make sure that only one digit (one 7-segment display) is ON at a time. Make sure to synchronize your digit value output with the display alternation. A sample code is given below:

```
module seven_alternate(clk, reset, big_bin, small_bin, AN);
        input clk;              // 1kHz clk
        input reset;
        input [15:0] big_bin;
        output reg [3:0] small_bin;
        output reg [3:0] AN;

        reg [1:0] count;        // we need to iterate through the displays

        always @(posedge clk or posedge reset)
        begin
                if (reset)
                begin
                        AN = 0;
                        small_bin = 0;
                        count = 0;
                end
                else
                begin
                        count = count + 1'b1;
                        case (count)
```

```
                                    2'b00: begin
                                           AN = 4'b1110;
                                           small_bin = big_bin[3:0]
                                           end
                                    2'b01: begin
                                           AN = 4'b1101;
                                           small_bin = big_bin[7:4]
                                           end
                                    2'b10: // your code here
                                    default: // your code here
                    endcase
            end
endmodule
```

Note that the output of the seven_alternate should be converted to 7-segment format before it could be displayed.

4. Clock Divider

A clock divider receives a clock signal as input, and using an internal counter divides the clock by some number in order to achieve the desired frequency. If the content of internal counter is greater than some number (you will need to figure out what this number is), then the output should be 1, and otherwise the output should be 0. A sample clock divider (divide by 16) is shown below:

```
module clk_div16(in_clk, out_clk);
        input in_clk;
        output out_clk;

        reg [3:0] count;

        always @(posedge in_clk)
        begin
                count <= count + 1'b1;
                if (count >= 4'b1000)
                        out_clk <= 1;
                else
                        out_clk <= 0;
        end

endmodule
```

5. Final Design

Figure 2 below shows a block diagram of the final design. Note that your design should also

have a global reset input, which is input to some of the design modules (omitted from the figure for clarity).

Note that the slow clock is required for testing your design on the board, since the fast clock is too fast for the human eye to see the changes in the counter. For the purpose of simulation you should only use a fast clock (so no clock divider). Otherwise, the simulation time will be too long.

Deliverables

- Demo your simulation waveforms and final design uploaded to the Xilinx Nexys-3 FPGA board.
- Submit your Verilog code on blackboard. It is recommended to tar or zip all your code together before uploading it.

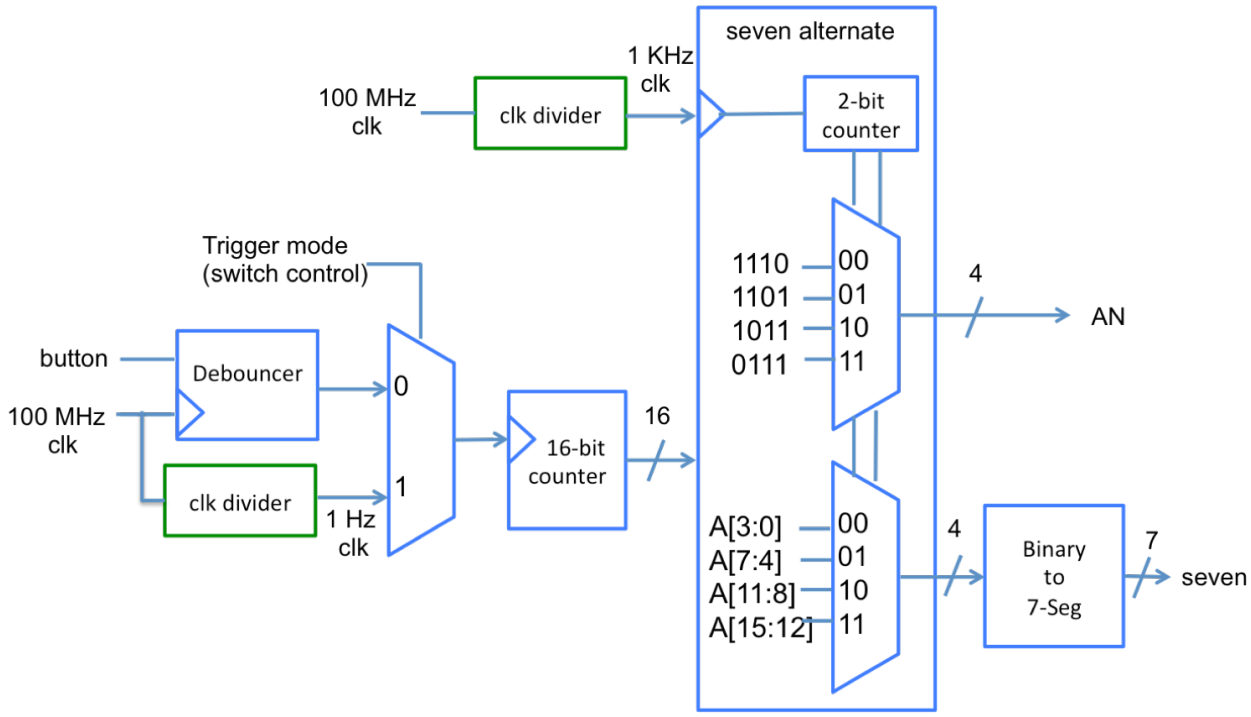

Figure 2: Block diagram for the overall design