EE 231 Lab 7

Sequential Circuits:
How Fast Are You?

In this lab you will design a sequential circuit to test your reaction speed. The basic idea is that you hit a switch as soon as you see an LED light up. The amount of time you took to react will be displayed on two 7-segment displays. Figure 1 shows the overall circuit that you will design and build. You will use a clock (*clk*) to drive a two-digit counter; the *w* signal will be a pulse that lights the LED. The LED will turn off as soon as you hit the switch, and the signal *Reset* can be used to reset the counters, the display, and get the circuit ready to start over.

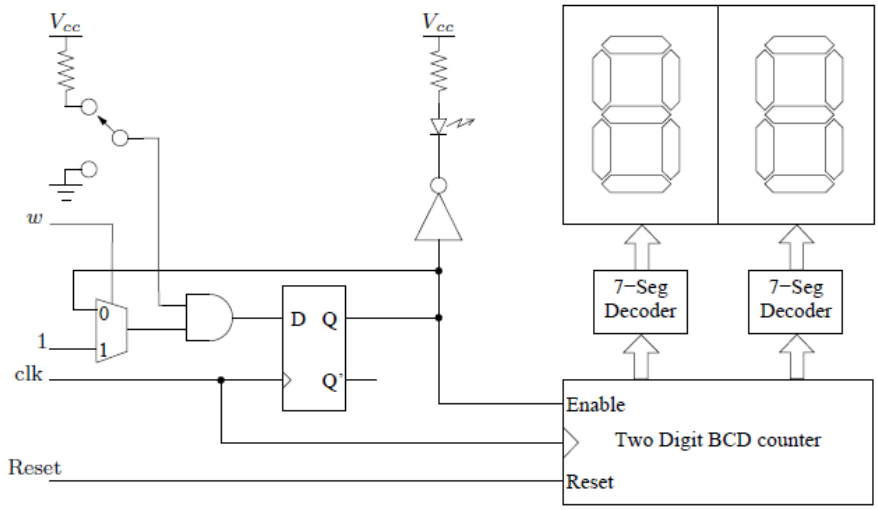

Figure 1: Circuit For Testing Your Reaction Speed

1. Prelab

1.1. We want a minimum resolution of measuring your reaction time to be 1/100 of a second, but your board has a 50MHz clock. How can you use a counter to generate a clock with frequency 100Hz

1.2. Now that you have a 100Hz clock, you can use that to count the time it takes you to press the switch after the LED goes on and display that on two 7-segment displays. To accomplish this you will need to write a code that will increment the least significant digit and every time it reaches the number 9 you reset it and increment the most significant digit. Write a Verilog code to implement the two digit BCD counter as sown in Figure 1.

1.3. Write a Verilog code to implement the entire circuit as shown in Figure 1. Use one of the LEDs on your board.

2. Lab

2.1. Simulate your Verilog program and make sure it works as planned.

2.2. Wire your double 7-segment display and test your counter code.

2.3. Wire two debounced switches. One will be used as the input to *w* and the other will be the one you depress once you see the LED light on.

2.4. Ask the instructor or TA to test the circuit and record how fast you are.

3. Supplementary Material

3.1. **Verilog – Sequential HDL**

So far the statement you have been using that are encapsulated by always were blocking type, i.e., they are executed sequentially. For this lab you will need to use a non-blocking procedural assignment. Non-blocking assignments are executed concurrently. To differentiate between the two lets look at the following two examples:

```
B = A

C = B + 1
```

and

```
B <= A

C <= B + 1
```

The first case is the blocking one, the statements are executed sequentially, therefore the first statement stores A into B, and then 1 is added to B and stored into C. At the end a value of A + 1 is stored into C. On the other hand, the second case is non-blocking. In this case the assignments are made using <= sign instead of = sign. Non-blocking statements are executed concurrently, so values of the right hand side are stored in temporary locations until the entire block is finished and then they will be assigned to the variables on the left. For this case C will contain the original value of B + 1 rather than A + 1 as in the first case.

Use blocking assignments when you must have sequential execution. If you are modeling edge-sensitive behavior use non-blocking assignments. In synchronous sequential circuits, changes occur based on transitions rather than levels. In this case we need to indicate whether the change should occur based on the rising edge or the

falling edge of the signal. This behavior is modeled using the two keywords *posedge* and *negedge* to represent rising and falling edge, respectively, for example:

always @(posedge clock, negedge reset)

will start executing on the rising edge of clock or on the falling edge of reset. If your circuit has an asynchronous reset, you may need an *if-else* statement to indicate whether you are resetting or triggering the circuit, in this case the very last statement of the non-blocking assignment needs to be the one related to the clock. For example for a D flip-flop with asynchronous reset you need to write it this way.

Program 1: An example of a D flip-flop with asynchronous reset

```
module D_flip_flop(
    output reg Q,
    input D, clock, reset
    );

    always @(posedge clock, negedge reset)
        if(~reset)
            Q <= 1'b0;
        else
            Q <= d;

endmodule
```