EE 231 Lab 7

Sequential Circuits:
How Fast Are You?

1.Lab

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

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

1.3.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.

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

2.Supplementary Material

2.1.Verilog – Sequential HDL

So far the statements 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, let’s look at the following two examples:

```
    B = A
    C = B + 1
and
    B <= A
    C <= B + 1
```

The first case is the blocking one, where 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 <= instead of a = 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 will 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)

This 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 a D flip-flop with asynchronous reset, example code follows for how to write this.

```
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
```

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