ELEC-4200
Digital System Design

FROM: Jacob Howard

TO: Prof. Ujjwal Guin & Yuxi Zhao

LAB SECTION: Thursdays @ 8:00 am (002)

DUE DATE: 4/22/21

Final Project Report

Introduction

The goal of this report is to explain my Final Project for Digital System Design (ELEC 4200). For our final project, we were able to choose and design a lab of our own with approval from Professor Guin. The project I chose to work on was a millisecond stopwatch. The stopwatch would be able to start and stop by a user's input and would also be able to be reset on a user's input.

Design

Initially, for my stopwatch design, I thought of what all my code would need for a stopwatch to work properly. The main elements that my code needed were a start and stop input, a reset input, a milliseconds output, and a seconds output. These were the core inputs and outputs I needed to make a stopwatch.

Next, I had to think of the code structure. I thought using a Finite State Machine (FSM) or an Algorithmic State Machine (ASM) would be the best and easiest way to structure the code. The code started with a “Reset State” that would check if the reset button has been pushed at all to reset the time. If after checking the Reset State, the code would go to the “Start State” to see if the start switch was on. If it was, the program would add “1” to the milliseconds each loop around until milliseconds hit “60”. Once milliseconds hit 60, the program would add “1” to the second's output and reset milliseconds to “0” and start the process over again until the start switch was turned off. If the start switch was off, the program would just display the last known values of milliseconds and seconds until the reset button was pressed.

Lastly, I had to implement a way to display the stopwatch time. In my initial design, I did this by using several 7-segment displays. My ultimate goal was to have four 7-segment displays showing the time; two displays being used to show the milliseconds ones and tens place and the other two displays to show the seconds ones and tens place. For testing, I chose to work with two 7-segment displays, to begin with; one showing the tens place of milliseconds and the other showing the one's place of seconds. I made two decoders to decode milliseconds and seconds for the displays. My initial design had the decoders and displays running simultaneously with the state machine code. This ensured the time to be displayed at any given state.

Testing

Once we completed our design code in Verilog, we were able to go into the lab and physically test our projects on the Nexys A7 boards and demo them to the TA. Sadly, my design had some issues during the physical testing. While my code was running on the board, the correct time was not displaying on the 7-segment displays. The start/stop switch was working correctly, as well as the reset button. I thought the problem was my design for the displays so I tried using LEDs to display the time in binary. Sadly, this solution did not seem to work either. There was some bug in my code that was not allowing me to display the correct values on the Nexys board. After spending some time debugging with the TA, we found that the simulation seemed to display the correct values for the time but the board was not displaying the correct values.

Results

So, through my testing, I found that the code does not completely work with physical testing in the Nexys Board at the moment. I'm sure with more time and help, I would be able to figure out all the issues and correctly display the stopwatch time on the board. Through simulation testing, I did find that the values are correctly counting: milliseconds counts to 60.

Once it hits 60, it will add "1" to seconds and reset to "0" and continue the counting process. You can see the successful simulations in *Figures 1 and 2*. *Figure 1* shows seconds going from 0 to 1 once milliseconds hits 60 and resets and *Figure 2* shows seconds holding "1" until milliseconds hits 60 again. Once milliseconds hit 60 again, it resets, and seconds gets "1" added to it and becomes decimal "2". The code I wrote for this Lab Final can be seen in the *Code 1* block below. Note that this code is slightly edited from the physical lab testing on the Nexys board, as this code was written to get the correct simulations. Some specific changes include commenting out the clock divider, decoders, and 7-segment displays. (*The code can be viewed easier on CodePile website at* https://www.codepile.net/pile/8Ol0NZOe)

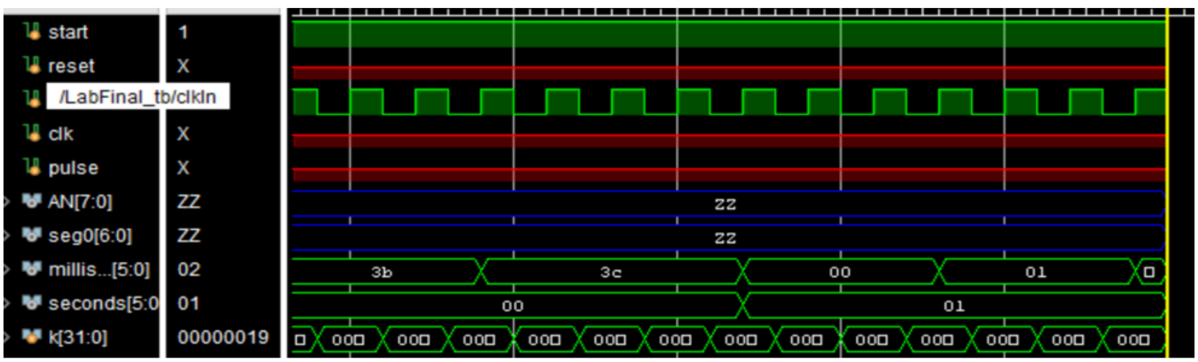

Figure 1

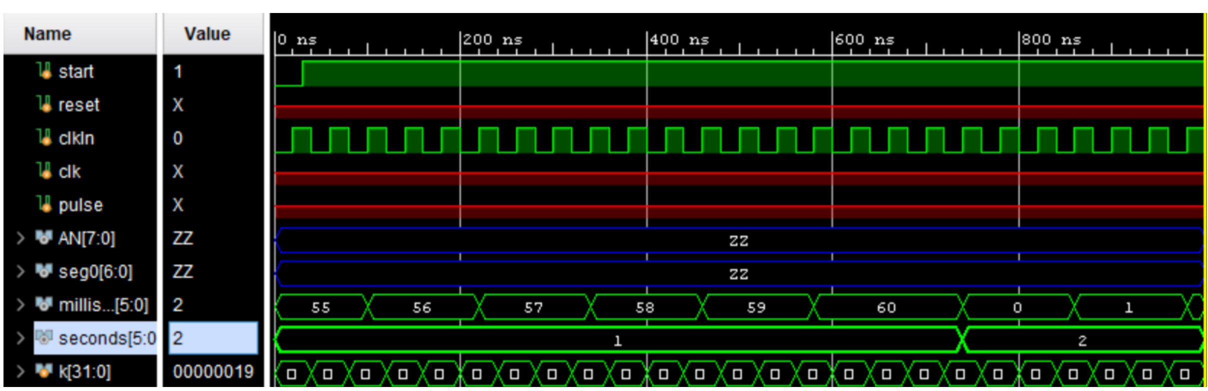

Figure 2

Conclusion

In conclusion, it seems that my design does perform as intended during the simulation. The problem seems to be through physical testing. I wish I would have had time to start on the project earlier and had access to a Nexys board to figure out the problem. I'm sure there are only some slight errors in my code that are causing physical testing to not completely work as intended. If I was able to figure out the problems, I would like to add up to at least four 7-segment displays to display the time. Another enhancement I would like to add is a start/stop button instead of a switch. This would make a more natural stopwatch. Overall, after working on my project, I found it to be more difficult than expected. I do believe with more time, I would be able to design a working stopwatch on the Nexys board. I may even come back to this project in the future to figure out what went wrong and apply better solutions.

```
module LabFinal_NoDisplay(
input start,
input reset,
input clkIn, //clock input 1
//input clk,

//output reg [11:0] timer, //full time value
output reg [7:0] AN, //Output for 7-seg display constraints
output reg [6:0] seg0, //7 segment display
output reg [5:0] milliseconds = 6'b000000,
output reg [5:0] seconds = 6'b000000,
output reg clockDiv_out
);
//(* DONT_TOUCH = "TRUE" *)
//reg [5:0] seconds, milliseconds;
reg[27:0] counter=28'd0;
parameter DIVISOR = 28'd5000000;
integer a;
wire clk;
wire lock;
//clk_wiz_1 inst2 (.clk_in1(clkIn), .clk_out1(clk), .locked(lock));
//reg [5:0] milliseconds, seconds; //placeholders for seconds and milliseconds
```

```
reg [2:0] state, nextstate;
parameter [2:0] Reset=0, Start=1, Milliseconds=2, Seconds=3, Delay=4; //states for start, stop,
and reset
//(* DONT_TOUCH = "TRUE" *) reg [5:0] milliseconds;

/*
//clock divider
always @(posedge clk)
begin
 counter <= counter + 28'd1;
 if(counter>=(DIVISOR-1))
  counter <= 28'd0;
 clockDiv_out <= (counter<DIVISOR/2)?1'b1:1'b0;
end
*/
//Main
always @ (posedge clkIn)
state <= nextstate;

always @(state or start or reset)
begin
    nextstate = 1'b0;

    case(state)
        Reset: if (reset) //Reset State
        begin
            milliseconds = 6'b000000; //resets milliseconds placeholder to 0
            seconds = 6'b000000; //resets seconds placeholder to 0
            //timer = 12'b000000000000; //resets the time to 00:00
            nextstate = Start;
        end
        else
        nextstate = Start;

        Start: if (start) //Start Case
            begin
            nextstate = Milliseconds;
            end
            else
            begin
            nextstate = Reset;
            end
        Milliseconds: if (milliseconds < 6'b111100)
                begin
               milliseconds = milliseconds + 1; //adding 1 millisecond
                //timer [5:0] = milliseconds; //setting first 6 bits of timer to milliseconds time
```

```
            nextstate = Reset; //Goes to decoder
            end
            else
            begin
            nextstate = Seconds;
            end
   Seconds: if (seconds < 6'b111100)
            begin
               milliseconds = 0; //resets milliseconds to zero
               seconds = seconds + 1; //adding to seconds if milli = 60

               nextstate = Reset; //Goes to decoder
            end
          else //may support more than 1 minute if coded here
          begin
          milliseconds = 6'b000000;
          seconds = 6'b000000;
          nextstate = Reset; //For now, if one minute hits, we stop
     end

      /*
     Decoder1:  //Decoder state. Decodes binary to 7-segment displays
        begin
        AN = 8'b11111110; //first 7-segment display
                    case(milliseconds)
                          //0-9
                       0   :seg0=7'b0000001;
                       1   :seg0=7'b1001111;
                       2   :seg0=7'b0010010;
                       3   :seg0=7'b0000110;
                       4   :seg0=7'b1001100;
                       5   :seg0=7'b0100100;
                       6   :seg0=7'b0100000;
                       7   :seg0=7'b0001111;
                       8   :seg0=7'b0000000;
                       9   :seg0=7'b0000100;
                       //10-19
                       10   :seg0=7'b0000001;
                       11   :seg0=7'b1001111;
                       12   :seg0=7'b0010010;
                       13   :seg0=7'b0000110;
                       14   :seg0=7'b1001100;
                       15   :seg0=7'b0100100;
                       16   :seg0=7'b0100000;
                       17   :seg0=7'b0001111;
                       18   :seg0=7'b0000000;
```

```
19   :seg0=7'b0000100;
//20-29
20   :seg0=7'b0000001;
21   :seg0=7'b1001111;
22   :seg0=7'b0010010;
23   :seg0=7'b0000110;
24   :seg0=7'b1001100;
25   :seg0=7'b0100100;
26   :seg0=7'b0100000;
27   :seg0=7'b0001111;
28   :seg0=7'b0000000;
29   :seg0=7'b0000100;
//30-39
30   :seg0=7'b0000001;
31   :seg0=7'b1001111;
32   :seg0=7'b0010010;
33   :seg0=7'b0000110;
34   :seg0=7'b1001100;
35   :seg0=7'b0100100;
36   :seg0=7'b0100000;
37   :seg0=7'b0001111;
38   :seg0=7'b0000000;
39   :seg0=7'b0000100;
//40-49
40   :seg0=7'b0000001;
41   :seg0=7'b1001111;
42   :seg0=7'b0010010;
43   :seg0=7'b0000110;
44   :seg0=7'b1001100;
45   :seg0=7'b0100100;
46   :seg0=7'b0100000;
47   :seg0=7'b0001111;
48   :seg0=7'b0000000;
49   :seg0=7'b0000100;
//50-59
50   :seg0=7'b0000001;
51   :seg0=7'b1001111;
52   :seg0=7'b0010010;
53   :seg0=7'b0000110;
54   :seg0=7'b1001100;
55   :seg0=7'b0100100;
56   :seg0=7'b0100000;
57   :seg0=7'b0001111;
58   :seg0=7'b0000000;
59   :seg0=7'b0000100;
//default
```

```
                default: seg0=7'bx;
            endcase

    nextstate = Decoder2; //Must loop back to start state
    end

    Decoder2:
    begin
    AN = 8'b11111101;
            case(seconds)
                0   :seg0=7'b0000001;
                1   :seg0=7'b0000001;
                2   :seg0=7'b0000001;
                3   :seg0=7'b0000001;
                4   :seg0=7'b0000001;
                5   :seg0=7'b0000001;
                6   :seg0=7'b0000001;
                7   :seg0=7'b0000001;
                8   :seg0=7'b0000001;
                9   :seg0=7'b0000001;
                //10-19
                10   :seg0=7'b1001111;
                11   :seg0=7'b1001111;
                12   :seg0=7'b1001111;
                13   :seg0=7'b1001111;
                14   :seg0=7'b1001111;
                15   :seg0=7'b1001111;
                16   :seg0=7'b1001111;
                17   :seg0=7'b1001111;
                18   :seg0=7'b1001111;
                19   :seg0=7'b1001111;
                //20-29
                20   :seg0=7'b0010010;
                21   :seg0=7'b0010010;
                22   :seg0=7'b0010010;
                23   :seg0=7'b0010010;
                24   :seg0=7'b0010010;
                25   :seg0=7'b0010010;
                26   :seg0=7'b0010010;
                27   :seg0=7'b0010010;
                28   :seg0=7'b0010010;
                29   :seg0=7'b0010010;
                //30-39
                30   :seg0=7'b0000110;
                31   :seg0=7'b0000110;
                32   :seg0=7'b0000110;
```

```
                    33  :seg0=7'b0000110;
                    34  :seg0=7'b0000110;
                    35  :seg0=7'b0000110;
                    36  :seg0=7'b0000110;
                    37  :seg0=7'b0000110;
                    38  :seg0=7'b0000110;
                    39  :seg0=7'b0000110;
                    //40-49
                    40  :seg0=7'b1001100;
                    41  :seg0=7'b1001100;
                    42  :seg0=7'b1001100;
                    43  :seg0=7'b1001100;
                    44  :seg0=7'b1001100;
                    45  :seg0=7'b1001100;
                    46  :seg0=7'b1001100;
                    47  :seg0=7'b1001100;
                    48  :seg0=7'b1001100;
                    49  :seg0=7'b1001100;
                    //50-59
                    50  :seg0=7'b0100100;
                    51  :seg0=7'b0100100;
                    52  :seg0=7'b0100100;
                    53  :seg0=7'b0100100;
                    54  :seg0=7'b0100100;
                    55  :seg0=7'b0100100;
                    56  :seg0=7'b0100100;
                    57  :seg0=7'b0100100;
                    58  :seg0=7'b0100100;
                    59  :seg0=7'b0100100;
                    default: seg0=7'bx;
                  endcase
                  nextstate = Reset;
          end

      Delay: if ( a<100)
      begin
        a = a+1;
        nextstate = Delay;
      end
      else
      begin
      a = 0;
      nextstate = Decoder1;
      end

*/
```

```
  endcase
end

/* Decoder
always @(milliseconds)
      begin
         if (clkIn)
         begin
         /* Millisecond Decoder (1's place) *
            AN = 8'b11111110; //first 7-segment display
            case(milliseconds)
                  //0-9
               0  :seg0=7'b0000001;
               1  :seg0=7'b1001111;
               2  :seg0=7'b0010010;
               3  :seg0=7'b0000110;
               4  :seg0=7'b1001100;
               5  :seg0=7'b0100100;
               6  :seg0=7'b0100000;
               7  :seg0=7'b0001111;
               8  :seg0=7'b0000000;
               9  :seg0=7'b0000100;
               //10-19
               10  :seg0=7'b0000001;
               11  :seg0=7'b1001111;
               12  :seg0=7'b0010010;
               13  :seg0=7'b0000110;
               14  :seg0=7'b1001100;
               15  :seg0=7'b0100100;
               16  :seg0=7'b0100000;
               17  :seg0=7'b0001111;
               18  :seg0=7'b0000000;
               19  :seg0=7'b0000100;
               //20-29
               20  :seg0=7'b0000001;
               21  :seg0=7'b1001111;
               22  :seg0=7'b0010010;
               23  :seg0=7'b0000110;
               24  :seg0=7'b1001100;
               25  :seg0=7'b0100100;
               26  :seg0=7'b0100000;
               27  :seg0=7'b0001111;
               28  :seg0=7'b0000000;
               29  :seg0=7'b0000100;
               //30-39
               30  :seg0=7'b0000001;
```

```
            31   :seg0=7'b1001111;
            32   :seg0=7'b0010010;
            33   :seg0=7'b0000110;
            34   :seg0=7'b1001100;
            35   :seg0=7'b0100100;
            36   :seg0=7'b0100000;
            37   :seg0=7'b0001111;
            38   :seg0=7'b0000000;
            39   :seg0=7'b0000100;
            //40-49
            40   :seg0=7'b0000001;
            41   :seg0=7'b1001111;
            42   :seg0=7'b0010010;
            43   :seg0=7'b0000110;
            44   :seg0=7'b1001100;
            45   :seg0=7'b0100100;
            46   :seg0=7'b0100000;
            47   :seg0=7'b0001111;
            48   :seg0=7'b0000000;
            49   :seg0=7'b0000100;
            //50-59
            50   :seg0=7'b0000001;
            51   :seg0=7'b1001111;
            52   :seg0=7'b0010010;
            53   :seg0=7'b0000110;
            54   :seg0=7'b1001100;
            55   :seg0=7'b0100100;
            56   :seg0=7'b0100000;
            57   :seg0=7'b0001111;
            58   :seg0=7'b0000000;
            59   :seg0=7'b0000100;
            //default
            default: seg0=7'bx;
        endcase
    end
else
    begin
        /* Millisecond Decoder (10's place) *
        AN = 8'b11111101;
        case(seconds)
            0   :seg0=7'b0000001;
            1   :seg0=7'b0000001;
            2   :seg0=7'b0000001;
            3   :seg0=7'b0000001;
            4   :seg0=7'b0000001;
            5   :seg0=7'b0000001;
```

```
6   :seg0=7'b0000001;
7   :seg0=7'b0000001;
8   :seg0=7'b0000001;
9   :seg0=7'b0000001;
//10-19
10   :seg0=7'b1001111;
11   :seg0=7'b1001111;
12   :seg0=7'b1001111;
13   :seg0=7'b1001111;
14   :seg0=7'b1001111;
15   :seg0=7'b1001111;
16   :seg0=7'b1001111;
17   :seg0=7'b1001111;
18   :seg0=7'b1001111;
19   :seg0=7'b1001111;
//20-29
20   :seg0=7'b0010010;
21   :seg0=7'b0010010;
22   :seg0=7'b0010010;
23   :seg0=7'b0010010;
24   :seg0=7'b0010010;
25   :seg0=7'b0010010;
26   :seg0=7'b0010010;
27   :seg0=7'b0010010;
28   :seg0=7'b0010010;
29   :seg0=7'b0010010;
//30-39
30   :seg0=7'b0000110;
31   :seg0=7'b0000110;
32   :seg0=7'b0000110;
33   :seg0=7'b0000110;
34   :seg0=7'b0000110;
35   :seg0=7'b0000110;
36   :seg0=7'b0000110;
37   :seg0=7'b0000110;
38   :seg0=7'b0000110;
39   :seg0=7'b0000110;
//40-49
40   :seg0=7'b1001100;
41   :seg0=7'b1001100;
42   :seg0=7'b1001100;
43   :seg0=7'b1001100;
44   :seg0=7'b1001100;
45   :seg0=7'b1001100;
46   :seg0=7'b1001100;
47   :seg0=7'b1001100;
```

```
                48   :seg0=7'b1001100;
                49   :seg0=7'b1001100;
                //50-59
                50   :seg0=7'b0100100;
                51   :seg0=7'b0100100;
                52   :seg0=7'b0100100;
                53   :seg0=7'b0100100;
                54   :seg0=7'b0100100;
                55   :seg0=7'b0100100;
                56   :seg0=7'b0100100;
                57   :seg0=7'b0100100;
                58   :seg0=7'b0100100;
                59   :seg0=7'b0100100;
                default: seg0=7'bx;
            endcase
        end
    end
*/
endmodule
```

Code 1