CSM152A - Lab 3 Report

Group Members:
Grant Roberts, 405-180-937
Ryan Brennan, 605-347-597
Yu Gao, 804-957-843

Section 5
TA: Kuo
Date Demoed: 2/19/2020

1) Introduction:

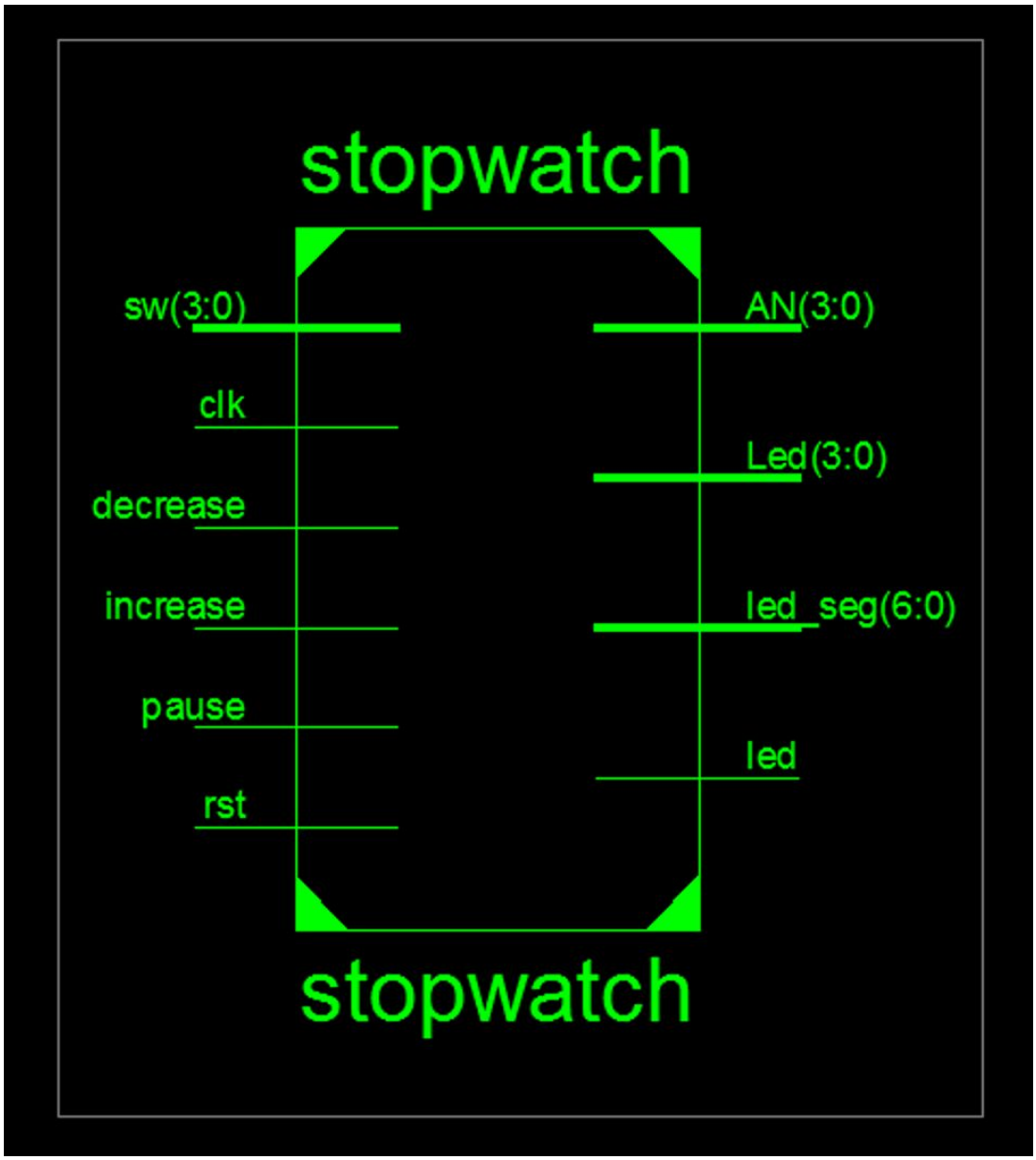
This lab builds on each of our prior labs, expanding the scope of our project to encompass the full design process for the Nexys 3 Spartan-6 FPGA board. Our task was to design and implement a stopwatch using the four seven-segment displays and two switches which included the functionality to count up or down as well as a mode to adjust the time on the clock. This adjustment mode worked by having an adjustment state that was set by a switch on the board. If adjustment was low, then the clock behaved normally. If it was high then we entered a state that is dependent on several other switches. The first is adj:b which if low incremented the clock by 2 at a rate of 2Hz, if high then we could increment or decrement the clock using the buttons on the board. The last was the count down switch, which when high would have the clock decrease by a second each time.

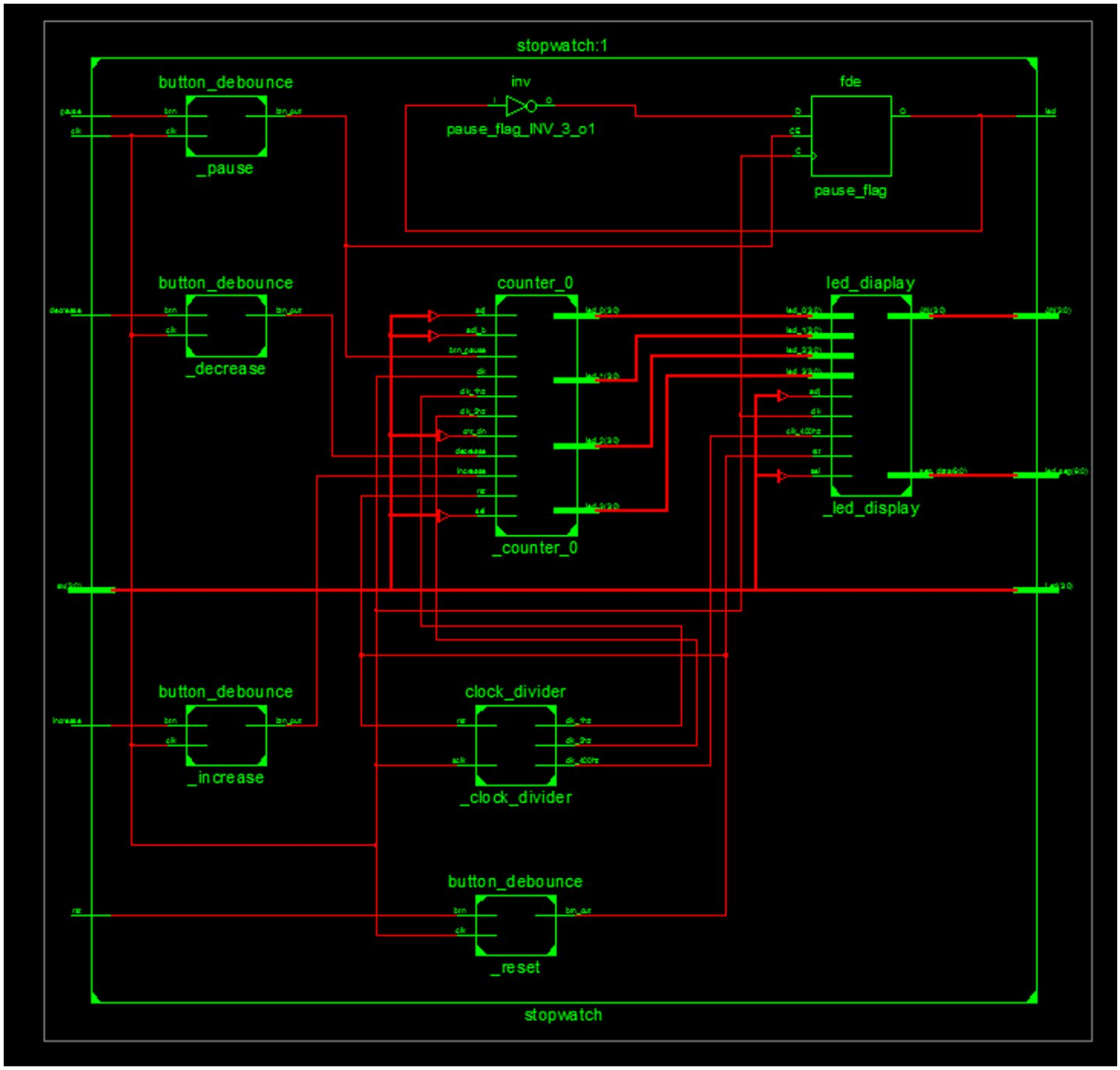
1.1) Design Requirements:

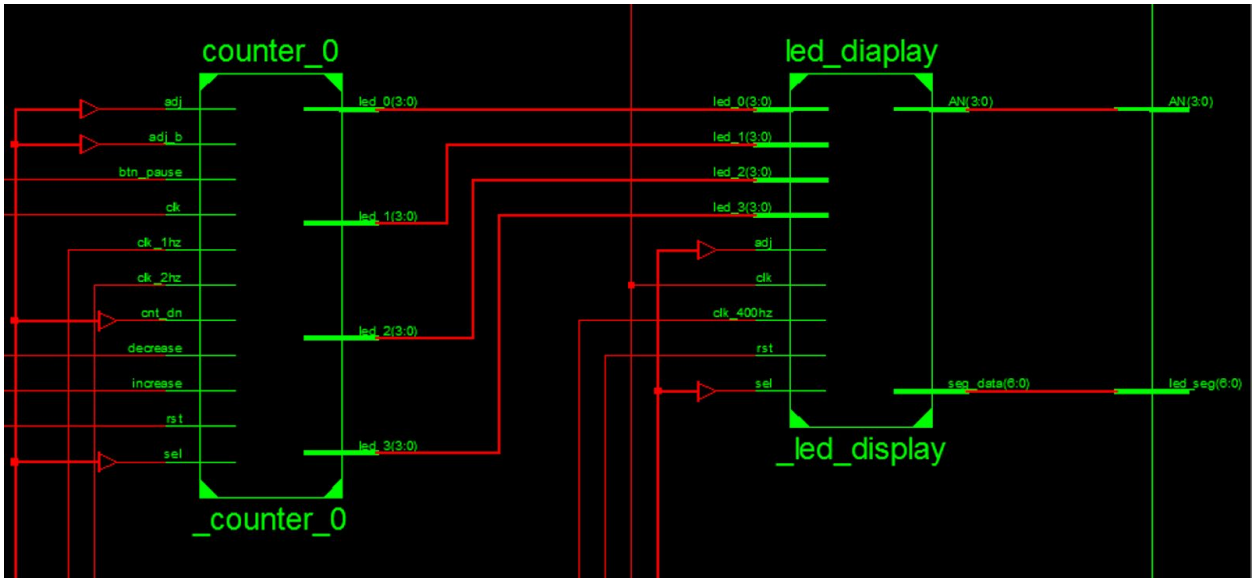
- **Stopwatch:** A stopwatch that begins from 00:00 and goes up to 59:59. It increments one second at a time.
- **Pause and Reset:** Two buttons, pause simply pauses the stopwatch at the current time. Reset resets the stopwatch to 00:00.
- **Sel:** A switch to select if the seconds of minutes are to be adjusted when the stopwatch is in adjustment mode. When the switch is high the selection is seconds, when the switch is low the selection is minutes.
- **ADJ:** A switch to enter the adjustment mode on the stopclock. The minutes or seconds will adjust based off the Sel switch.
- **ADJ:B:** A switch that selects if we are going to adjust the numbers based on the buttons to decrement and increment, or if the clock will increase by 2 each time at a rate of 2Hz.
- **CNT_DN:** A switch to decide which mode we are in. We are either going to count down if the switch is high and stop at 00:00, or if the switch is low we will count up normally at a rate of one second and stop at 59:59. Do note that when it reaches 59:59, if count down switch is high, it would automatically counts down.

2) Implementation:

There are 5 modules intotal: top, clock divider, button debouncer, counter, and led_dispaly.
Here is the RTL schematics:

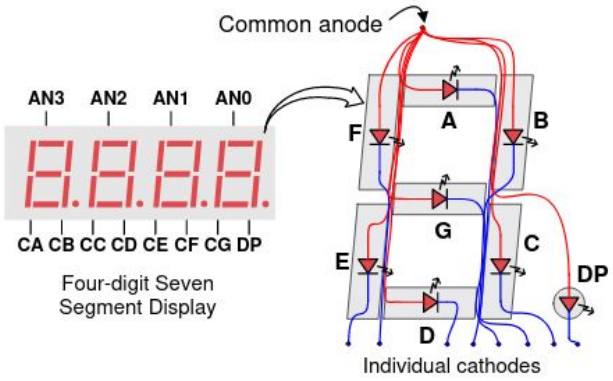


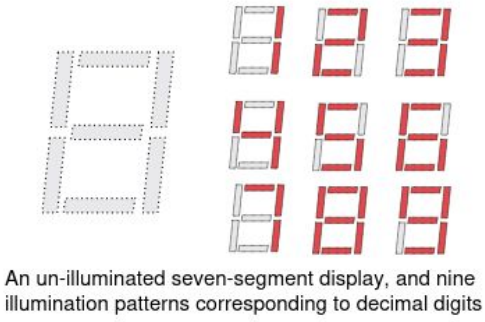

stopwatch:1
button_debounce
_pause
inv
pause_flag_INV_3_o1
fde
pause_flag
button_debounce
_decrease
counter_0
_counter_0
led_diaplay
_led_display
button_debounce
_increase
clock_divider
_clock_divider
button_debounce
_reset
stopwatch



2.1) 7-Segment Display

As a preliminary step we were to use the 7-segment display to display digits in sequence alternating from AN0 to AN3 using a simple 4-state counter modeled as a finite state machine.





Getting the 7-segment to display at the same time was one of the harder parts of the project. We started by just trying to hard code numbers onto the display. All of the display are not turned on at the same time, instead, the clock divider has a 400hz clock, and that was used to continuously select one of those four displays. Once we have all display lights up at the same time, we then started to implement the counter logic, however we found out that we first needed to learn how to make the individual digits respond separately. We used the following codes to encode the numbers and then would select the individual digit based on the chosen register. Here, clk_sel is a signal will change between clk_2hz and clk_400_hz, for blink the display and turn them on at the same time accordingly. For example, when the clk_sel is set to 2 hz in the adjusting mode, seg_data will be all 1s, which means off, or numbers at frequency of 2 hz.

```
case(led_data_hex)
        4'b0000:seg_data = (clk_sel)? 7'b1111111:7'b0000001; // 0
        4'b0001:seg_data = (clk_sel)? 7'b1111111:7'b1001111; // 1
        4'b0010:seg_data = (clk_sel)? 7'b1111111:7'b0010010; // 2
        4'b0011:seg_data = (clk_sel)? 7'b1111111:7'b0000110; // 3
        4'b0100:seg_data = (clk_sel)? 7'b1111111:7'b1001100; // 4
        4'b0101:seg_data = (clk_sel)? 7'b1111111:7'b0100100; // 5
        4'b0110:seg_data = (clk_sel)? 7'b1111111:7'b0100000; // 6
        4'b0111:seg_data = (clk_sel)? 7'b1111111:7'b0001111; // 7
        4'b1000:seg_data = (clk_sel)? 7'b1111111:7'b0000000; // 8
        4'b1001:seg_data = (clk_sel)? 7'b1111111:7'b0000100; // 9
endcase
```

2.2) Clock divider

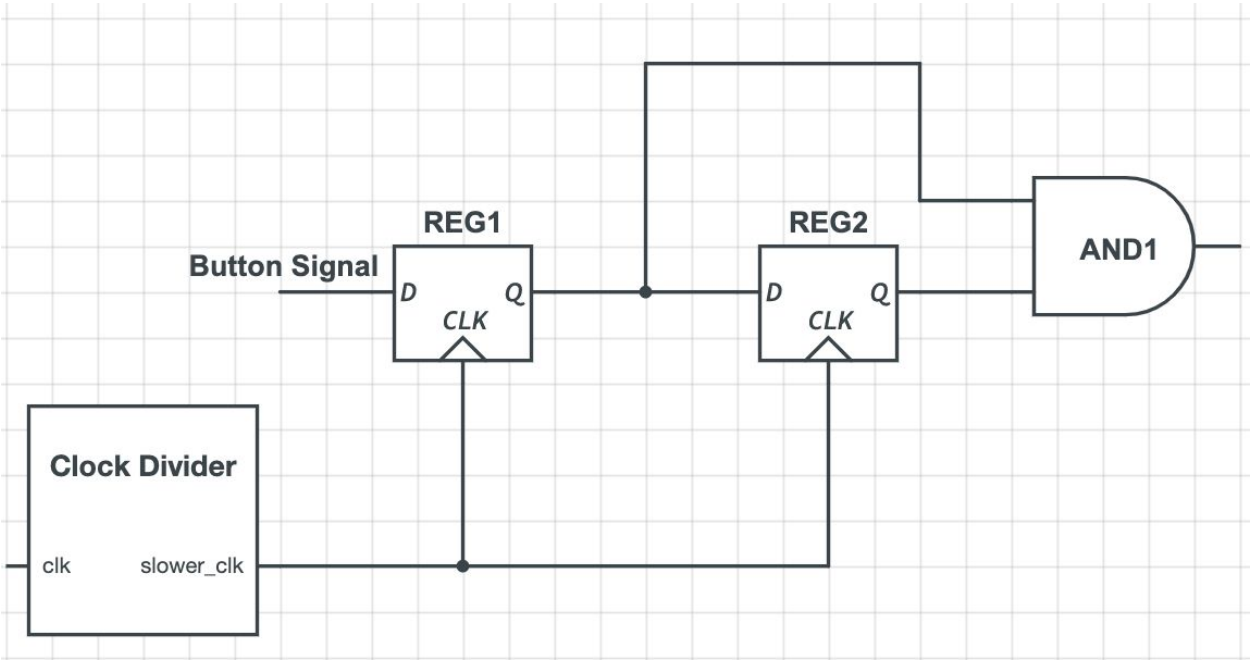
A clock divider was needed as we needed a 400Hz clock to select different display. We also needed a 1Hz clock to count up by a second each time. A 2Hz clock was also needed for the counting by two at frequency of 2 hz in the adjusting mode. We created this as a separate module.

2.3) 1Hz counter

We next began to implement the heart of our project: the counter. We created four 4-bit registers, corresponding to the ones and tens places for both seconds and minutes, and initialized all of them to 0. I'll refer to these registers as s0, s1, m0 and m1 respectively. On the positive edge of our clock posedge, we first check for a reset signal, in which case we return to the initial state of all zeros, otherwise we had four conditional statements. The first checks if s0 is greater than 9 and if so sets s0 to -1 and increments s1 by 1. The second conditional checks if s1 is greater than 5 and if so sets s1 to -1 and increments m0 by 1. This pattern continues a third time for m0 and m1. The last conditional checks if m1 is greater than 5 and if so simply maintains its state until reset. Finally we increment s0.

2.4) Debouncer

Since the four buttons, reset, pause, increase and decrease, needed to be able to differentiate between being turned on and being turned off, a button debouncer is created. The board can correctly pause, only reset once even if the user does not release the button press. A simulation for debounced button signal is also included in the following testing section.



2.5) Reset

After creating the debouncer, we assigned it to our reset button and mainly just had checks for when the reset button was pressed. If it was pressed we would simply set everything on the display to 00:00.

2.6) Pause

We then needed to apply pause. This button was also debounced and when pressed we would enter in a state where nothing was incrementing or changing. It would therefore give the appearance of pause in the display.

2.7) Adjust

Adjust was the next big step in the design after we got the stopwatch logic working. In adjust mode we are able to increment either the seconds or minutes based on the select. Adjust becomes the first signal to check in every clock cycle. Reset, pause, and normal counting up state are put to the else loop.

2.7.1) Select

To implement select we added an input to our counter.v file to allow us to have a flag to know which part is blinking between the seconds and minutes, and so we can increase the part accordingly.

We added another input adj in our counter.v file. We then essentially encapsulated our entire counter code within an if(adj) statement. Where we would perform adjustment operations if the adj flag was high.

2.7.2) Adj:B

We created another flag adj_b which then further encapsulated our code that was in the if(adj) flag. So we had two nested if statements using these two adj cases.

Depending on both the select and Adj_b flag we either are able to increase or decrease using the buttons or count at the 2Hz clock. We again nested these cases within if statements.

2.8) Count_DN

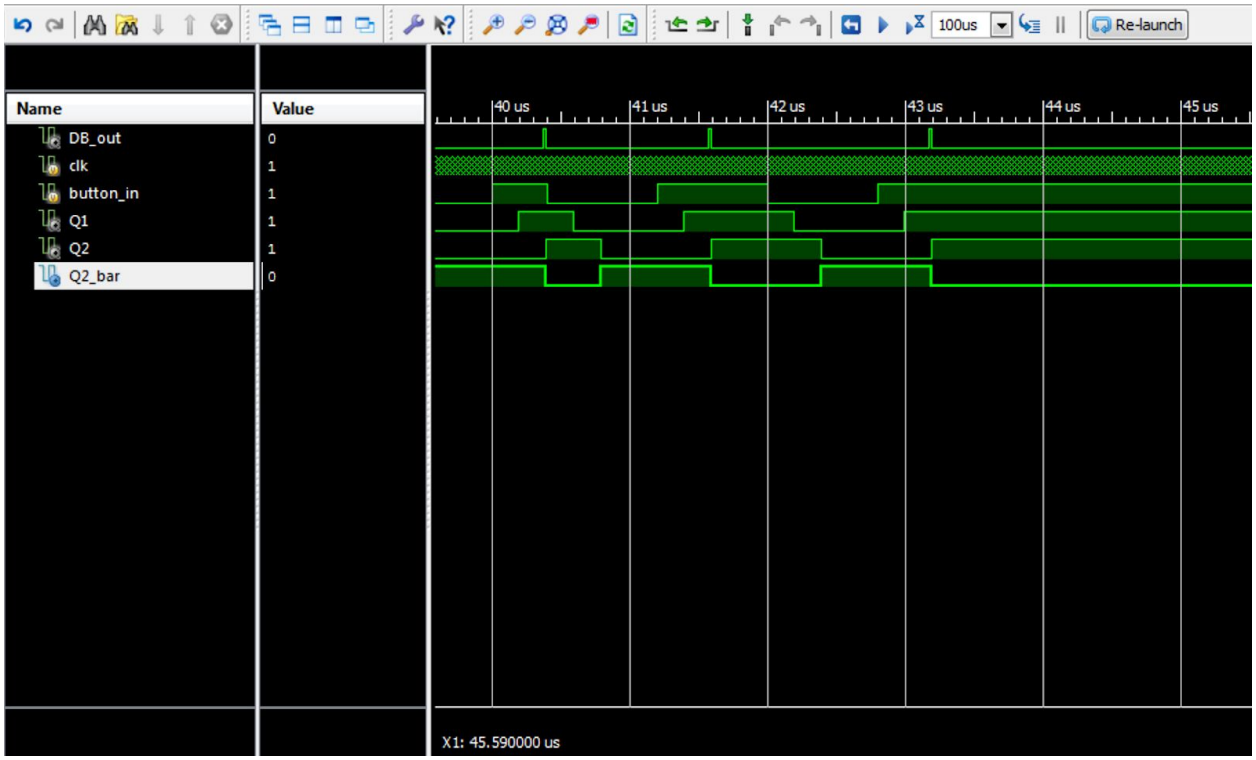
To implement countdown we had another flag in our counter.v file. We again had another nested if statement where we essentially just reversed out logic for counting up. This was one of the easier parts of the project to implement.

2.9) Stopwatch

We then tie all this together in our top module which we named stopwatch.v. This module is in charge of instantiating all of our modules(7-seg, counter, clock divider) and debouncing our buttons.

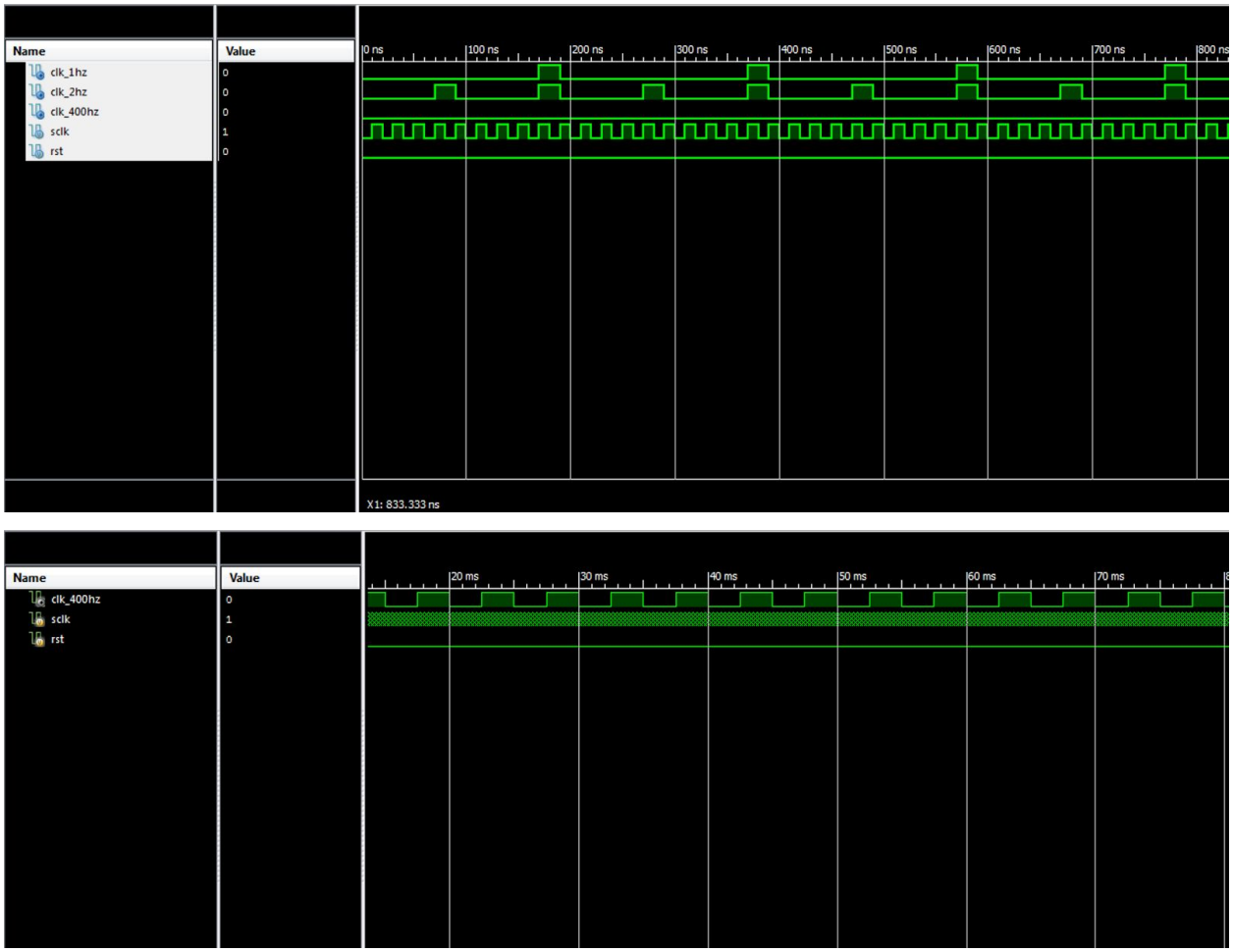
3) Testing

Button Debounce:



Here, the two Flip Flops Q1 and Q2, and Q2_bar are shown in the last three signal in picture. It clearly shows that for three presses with different period of time. The DB_out, which is the debounced sigal, always has only one tick. For example, when a user presses the “reset” button, no matter how long the button is pressed, the reset signal in the module only tick once.

Clock divider:



Here, we mostly showed that the tree output clk_1hz, clk_2hz, and clk_400hz are correct. In the first simulation window, it's clear that clk_1hz generated 2 times slower than clk_2hz. Do note that for the simulation purpose, we changed the ratio accordingly, because if we don't, the simulation time would be very long. As of the second picture, it's shown that the clk_400hz are a slower clock with frequency 400hz.

4) Conclusion

This project was a large step in both design and implementation in comparison to the other projects. We found that it was important to have a good idea of your implementation before you started. This became clear when we needed to stop working on our counter and go back to the 7-segment display and try and get the individual digits flashing. Once we were able to choose a single digit we went back to the counter and got that to translate to that 7-segment display as it should. Another design change we could make would be to somehow abstract our counter.v file a bit more. It got a big confusing and messy with the large nested if statements and we feel that this could have been cleaned up a bit. This was also a big step in time management. The other

projects we were mainly able to get them done with a full class in hand. However, we found we up against the clock and had to utilize office hours to complete the project.