Team Kublinh
Kubilay Agi 304784519
Baolinh Nguyen 104732121
Lab 2
February 26, 2018

Design Description

The design of this project was approached from a bottom-up perspective. We divided the project into four major submodules:

1. counter submodule
2. clock divider submodule
3. debouncer submodule
4. seven-segment display submodule

These four submodules are all under the single stopwatch top module.

Counter Submodule

The counter submodule increments the actual seconds and minutes counts of the stopwatch. The frequency of this incrementation depends on which mode the clock is in: adjust or regular mode. As input, this clock takes in the clock, the one Hz and two Hz clocks, and reset, pause, adjust, and select wires. From this input, the clock outputs the current minutes and seconds.
The module works by first selecting the correct clock to use, depending on whether the clock is in adjust mode or not. If the program should be in adjust mode, the clock needs to use the two Hz clock because the numbers are supposed to increment faster on the display. Also, depending on select input, the circuit decides whether to increment the minutes or the seconds. The clock increments all values regardless, but depending on the value of select, it will increment one set of values by 1 and the others by 0, meaning that one set stays the same and the other increases.

The module detects whether the state is paused or not. Then, the clock begins to increment, adjusting for overflow. When sec_one, min_one, and/or min_ten reach a value of 9, the value will reset to 0 and increment the succeeding value by 1. For sec_ten, we need to reset it and increment the next value whenever it reaches 5. Thus the maximum value for our counter is “99:59”.

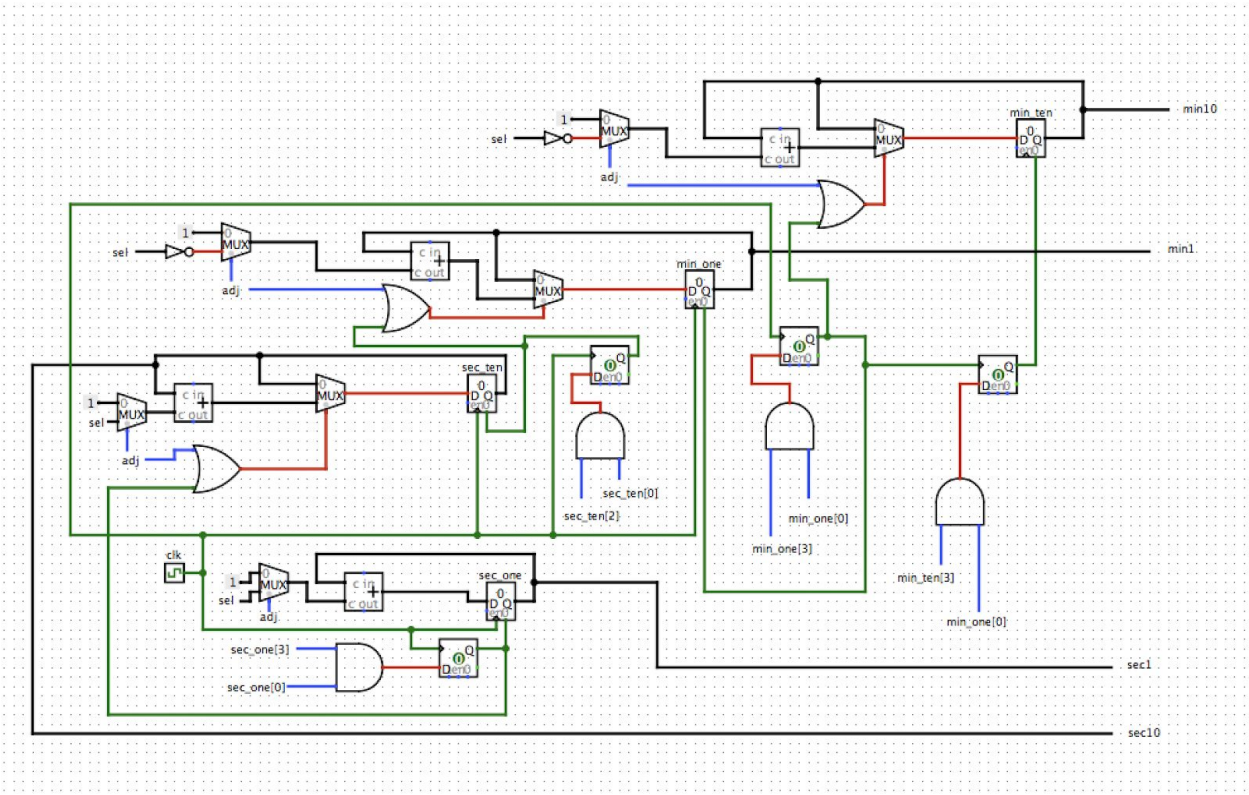

Figure 1: Schematic for the counter module

Clock Divider Submodule

The clock divider creates the appropriate clock for four different situations:

1. A one hz clock
2. A two hz clock to be used in adjust mode
3. A blinking clock, slower than the one hz clock, used for adjust mode
4. A fast display clock, used to display the numbers when not in adjust mode

Using the approach taken in Lab 1, there are four different clock divider numbers used to create the four different clocks. These clocks are used in the other submodules, including the counter submodule, to get correct timing behavior. We know that the clock on the board is 100 MHz, so we deliberately picked different values for clock dividers. In order to create the different frequencies, 100 MHz was divided by the aforementioned values depending on the functionality that was desired for the clocks. The lower the desired frequency, the higher the value of the clock divider that we used.

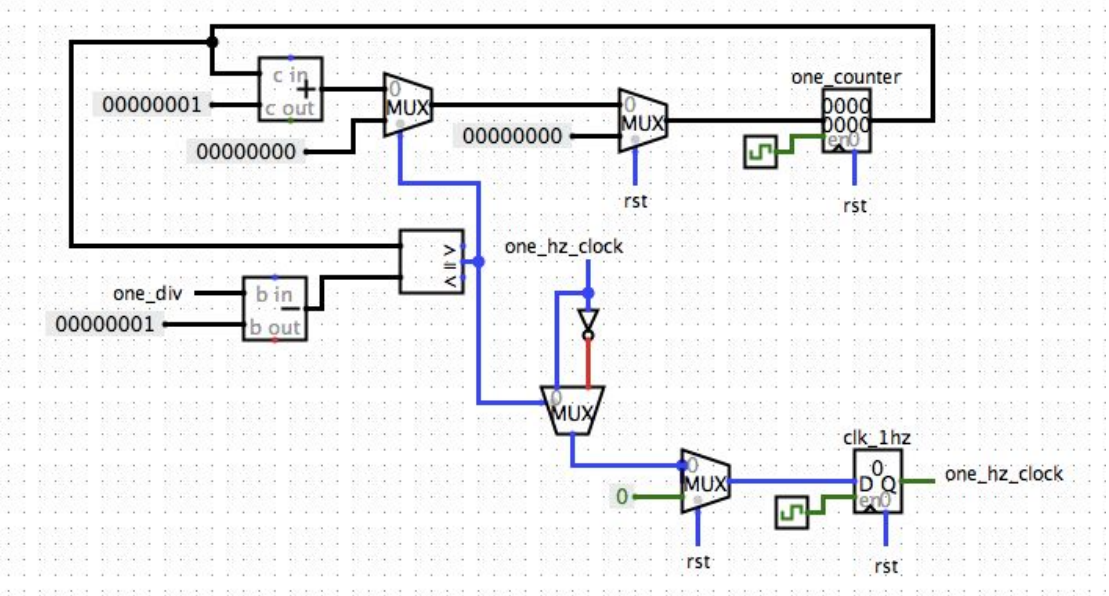

Figure 2a: Schematic diagram of the one Hz clock.

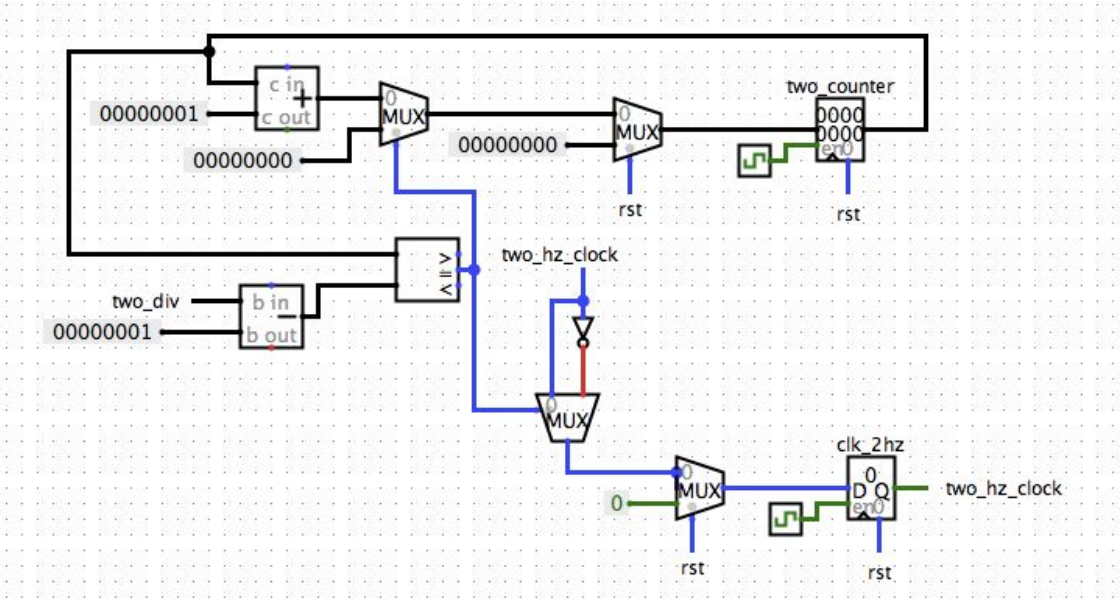

Figure 2b: Schematic diagram of the two Hz clock.

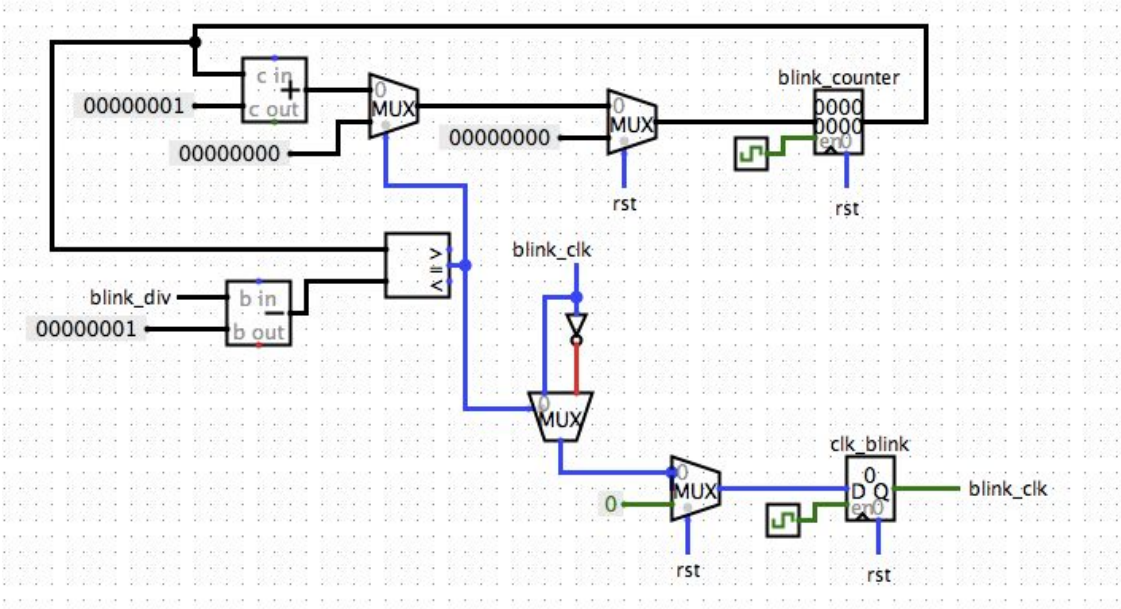

Figure 2c: Schematic diagram of the blinking clock.

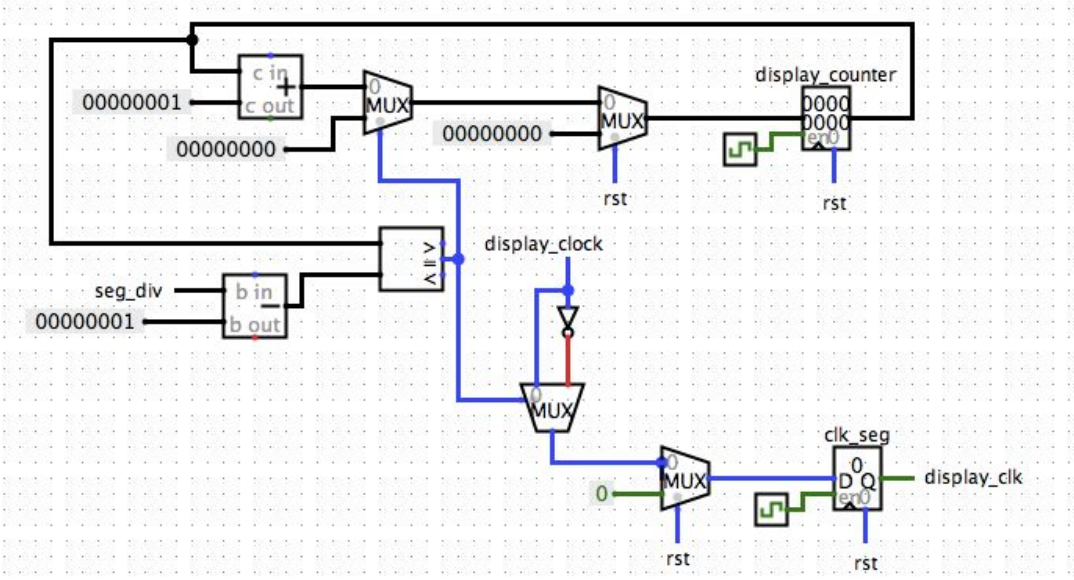

Figure 2d: Schematic diagram of the four Hz display clock.

Debouncer Submodule

The debouncer submodule that we originally implemented involved checking whether a button was pressed for a certain amount of time. However, the timer would sometimes not pause even when we pressed the pause button. Below is the schematic of our original implementation.

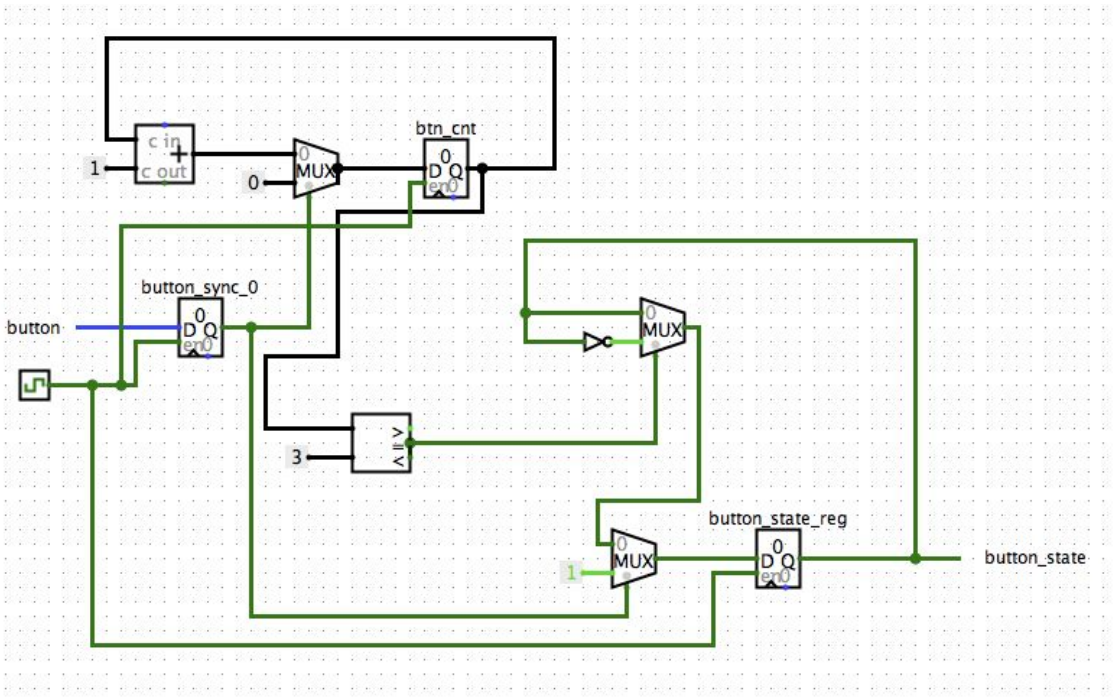

Figure 3a: Schematic diagram of the debouncer submodule.

It is possible that because the clock on the board is running at such a fast pace, we reach our threshold value twice (or any even number of times), which swaps the button state from not pressed (initially) to pressed (after one cycle, and then back to unpressed after reaching the value for the second time. This is possible because our register was declared as being 16 bits in size, but the value that we checked for was an 8'b1111_1111. Therefore, if the bottom 8 bits reached to 8'b1111_1111 twice or not once, we would get a value of zero as the button state, which the program would recognize as being not pressed. Therefore, the clock would stop only sometimes when we pressed it for just the right amount of time. At other times, the clock would not stop because we either did not press it long enough to reach the threshold value or we pressed it for too long and ended up in a time frame where the value would be flipped back to zero. If this was indeed the problem with our debouncer, the solution to this would have been to either make the threshold value bigger, or to add a break statement once the counter reached the threshold value. This way, we would not have to worry about it coming back around to the first value.

```
25  reg [15:0] btn_cnt;
26
27  always @ (posedge clk)
28  begin
29      if (button_state == button_sync_0)
30          btn_cnt <= 0;
31      else
32      begin
33          btn_cnt <= btn_cnt + 1'b1;
34          if (btn_cnt == 8'b1111_1111)
35              button_state_reg <= ~button_state_reg;
36      end
37  end
38
39  assign button_state = button_state_reg;
```

Figure 3b: Our code for the debouncer

Seven-Segment Display Submodule

The seven-segment display submodule works by using a switch statement, implemented by a multiplexer, to choose what set of binary numbers to send to the seven segment display hardware on the board. This was determined from the Nexys 3 manual that was given to use in the lab. For this module, we only care about numbers from 0 through 9 because those are the only ones that can be displayed on each digit of the seven segment display. We instantiated this module four times, once for each of the four digits that can be used on our display.

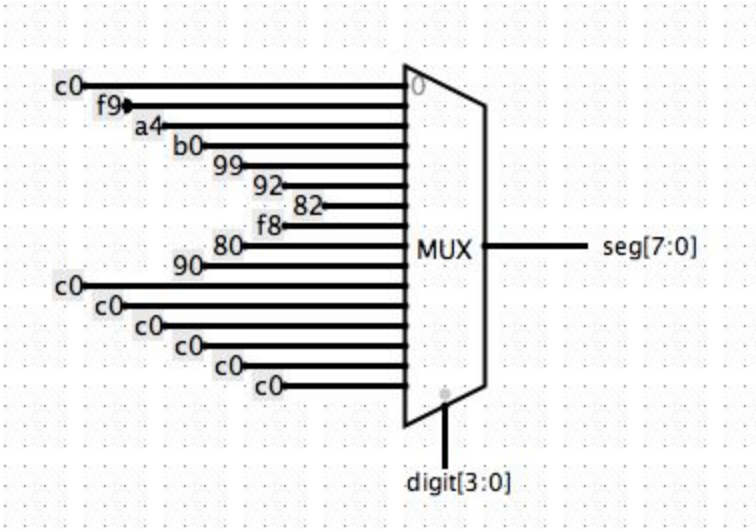

Figure 4: The seven-segment display submodule.

Stopwatch Top Module

The stopwatch top module contains instantiations of all the submodules: the counter, the debouncer, the clock, and the display modules. These all work together to produce the correct behavior. Added in this top submodule is the portion of the code that enables the display to work. This is done by cycling quickly through the displays of each of the four digits using a counter

called cnt. The digit to be displayed is determined not only by this cnt register but also by several other factors, including the sel and adj wires, to account for the select and adjust times. Moreover, the blink clock is taken into account when the adjust mode is switched on.

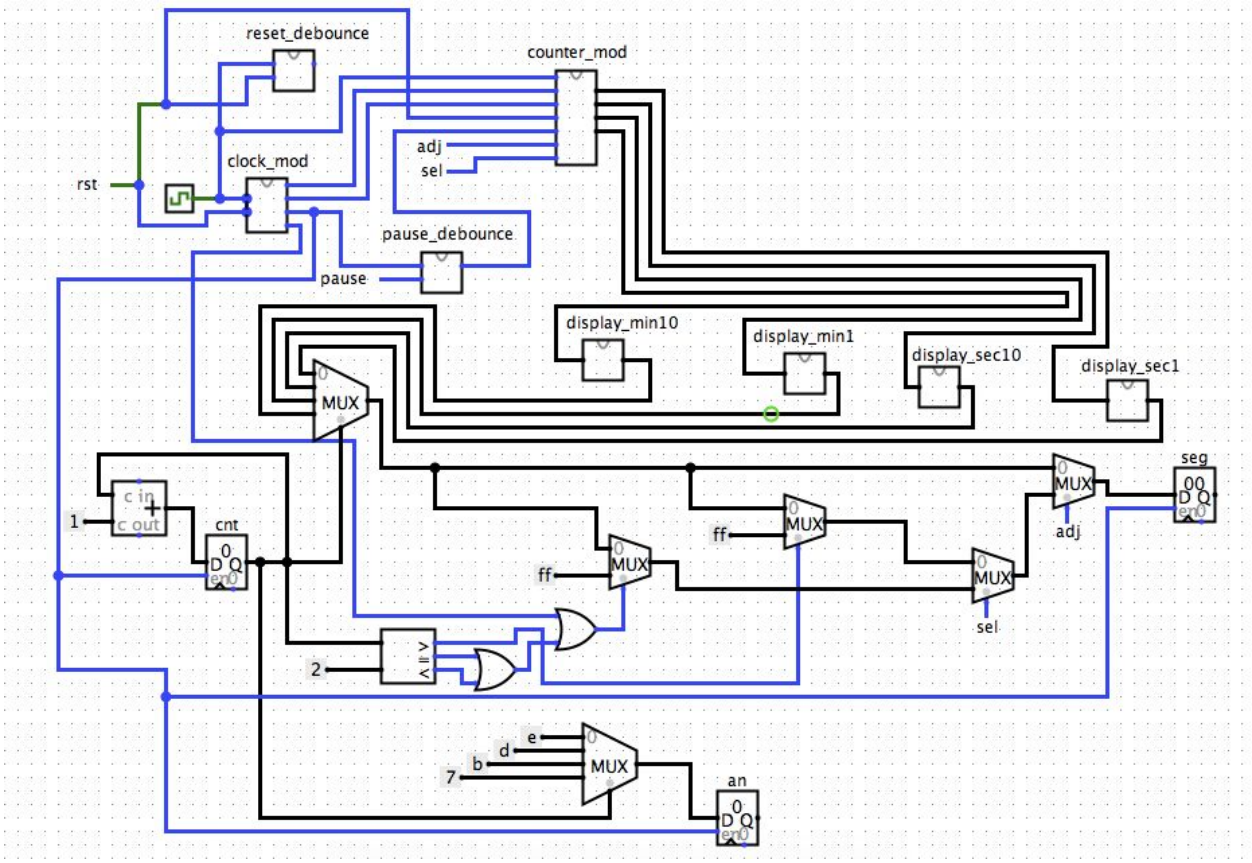

Figure 5: Stopwatch top module schematic.

Simulation Documentation

The testing of many of the submodules, including the display module, the debouncer, and the stopwatch top module, was done through programming the FPGA and testing the results by hand. We tested the functionality of the pause and reset buttons, as well as the adjustment switches. However, before we were able to do that, we first tested through simulation, two of the more crucial submodules we had created: the clock submodule and the counter submodule.

Testing the Clock Submodule

The clock submodule is used by almost all of the other modules, meaning that the clock submodule must perform its function correctly. In order to test our clock divider, we simply ran it, looking to see if the various clocks of different frequencies we created ran at the correct frequencies.

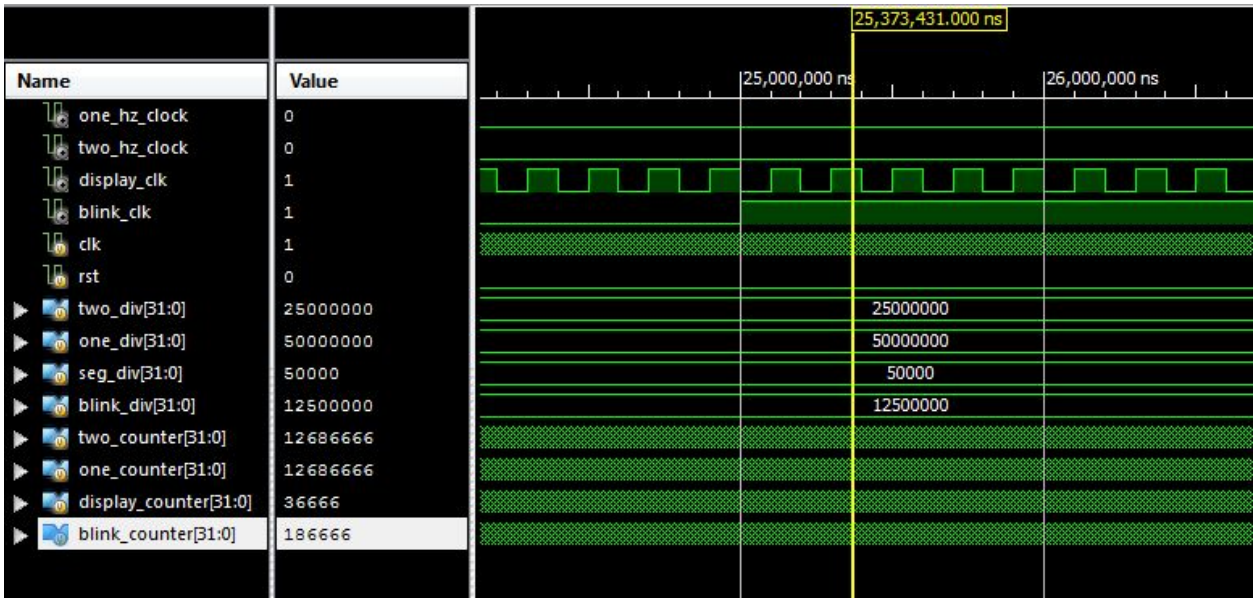

Figure 6: The simulation waveform of our clock module. Seen are the clock divider numbers and the current counts for each of the various clocks that were created. It can also be seen that the different clocks have different frequencies.

Testing the Counter

We tested the counter by looking at whether the counter would reset at different values. To do this, we simply ran the counter, modifying the one_hz_clock itself to make the run faster. We first checked if the minutes would increment by 10 correctly.

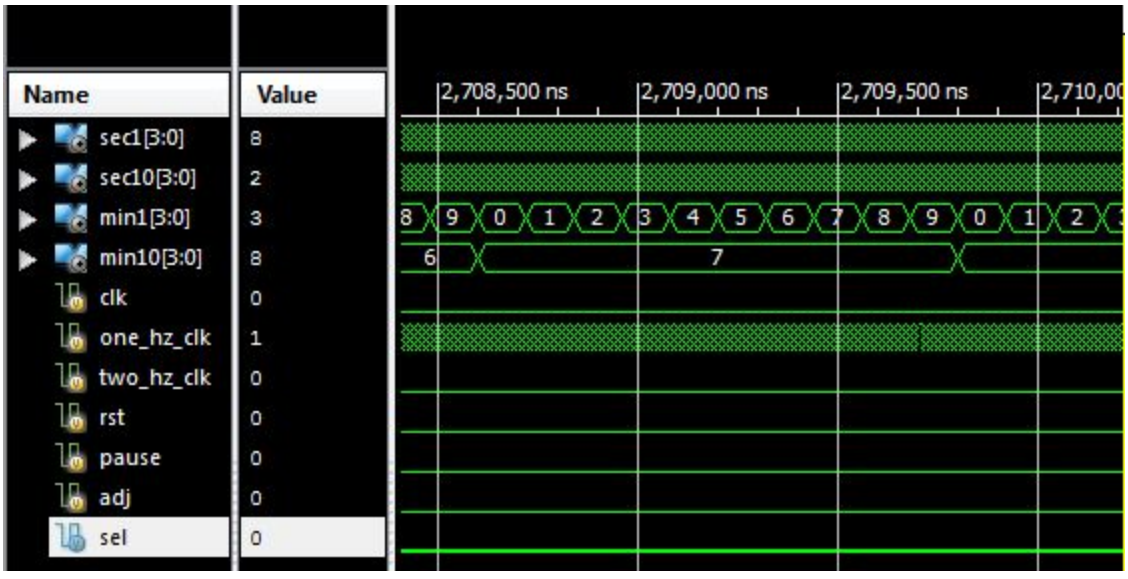

Figure 7: The above figure shows that when the minutes count hits a 9, the ten count of the minutes is incremented by 1 while the one count of the minutes is reset to 0.

We then checked if the minutes would increment by 1 after 59 seconds.

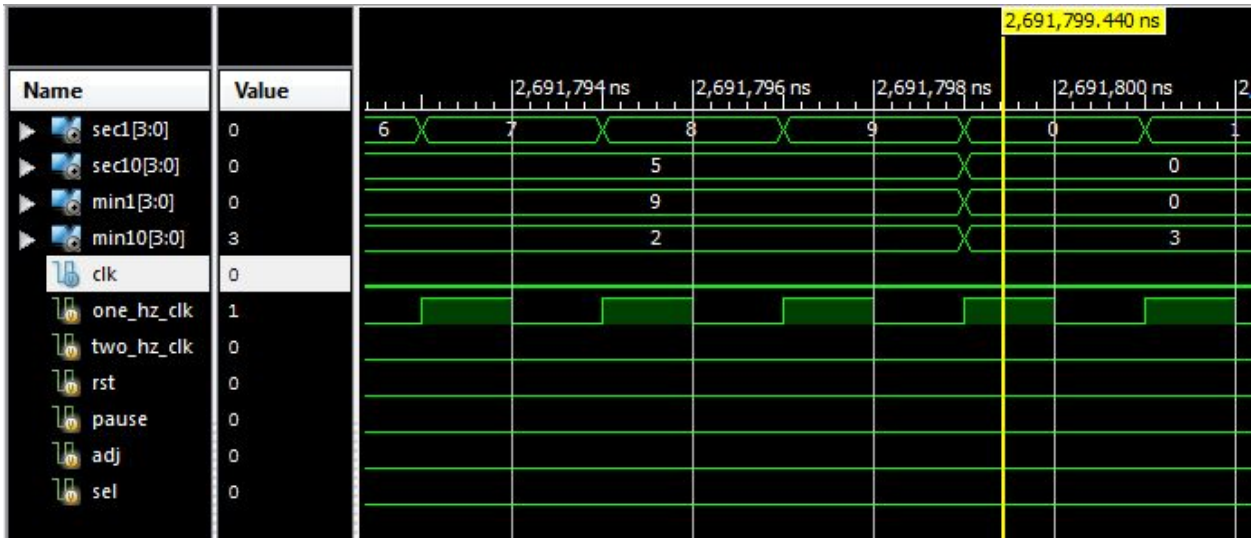

Figure 8: As shown, when the seconds hits 59, the minutes will then increment by 1.

Once we ensured that worked, we then checked the case where if the counter hits its max, 99:59. In this case, the counter should be reset.

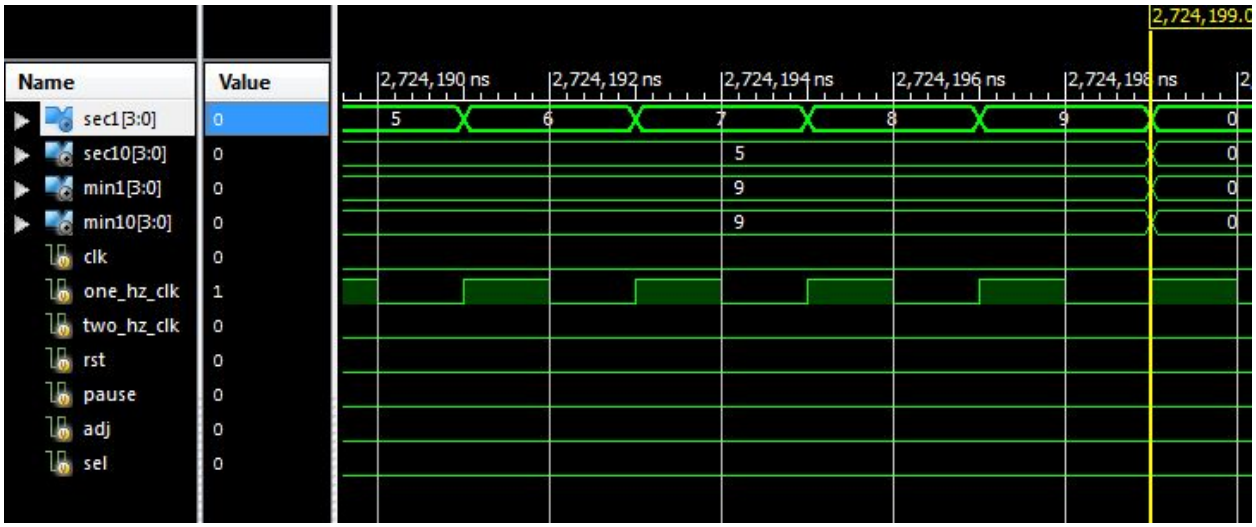

Figure 9: As shown, the counter in fact does return back to 00:00 when it hits 99:59.

Our final test was of the pause functionality. We tested to see if this would work independent of the button by flipping one_hz_clock several times, setting the pause, then flipping the one_hz_clock and seeing if the counter would, correctly, stop incrementing.

```
#100
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
pause = 1;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
#1 one_hz_clk=~one_hz_clk;
```

Figure 10: The code that was placed in the test bench to test the pause

Figure 11: The result, showing that the counter correctly stops incrementing when the pause is flipped, even if the one_hz_clock is still being flipped. It is worth noting that the pause functionality worked on the simulation because all of the lows and highs are precise. However, when we moved the project onto the board where the signals from the button bounce, our debouncer was not able to 100% successfully handle this.