**Team Multiplexers**

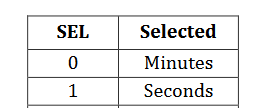
Alexander Graening (804732740), Xilai Zhang (804796478), Joseph Miller (504744848)

**Lab Report 3 (Team Multiplexers)**

**Section 1: Design Description**

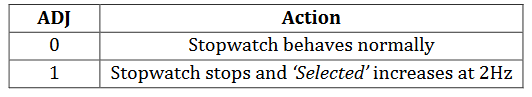
**Overall Description:**

For lab 3 we designed a stopwatch utilizing the onboard seven segment display, buttons, and switches. The first two seven segment display digits indicate how many minutes have ticked by since the start of the timer while the last two display the number of seconds for the current minute being measured. There are two buttons that implement a stop and reset for the stopwatch. While stopped, the counter does not increment until the stop button is pressed again and when reset, the seven segments should display 0 minutes 0 seconds. There are two switches used to set adjust mode and select either minutes or seconds.



**Figure 1:** Mappings of the Select Switch

When in adjustment mode, the selected digits will increment at 2Hz and blink at 5Hz to indicate to users which segment is being adjusted.

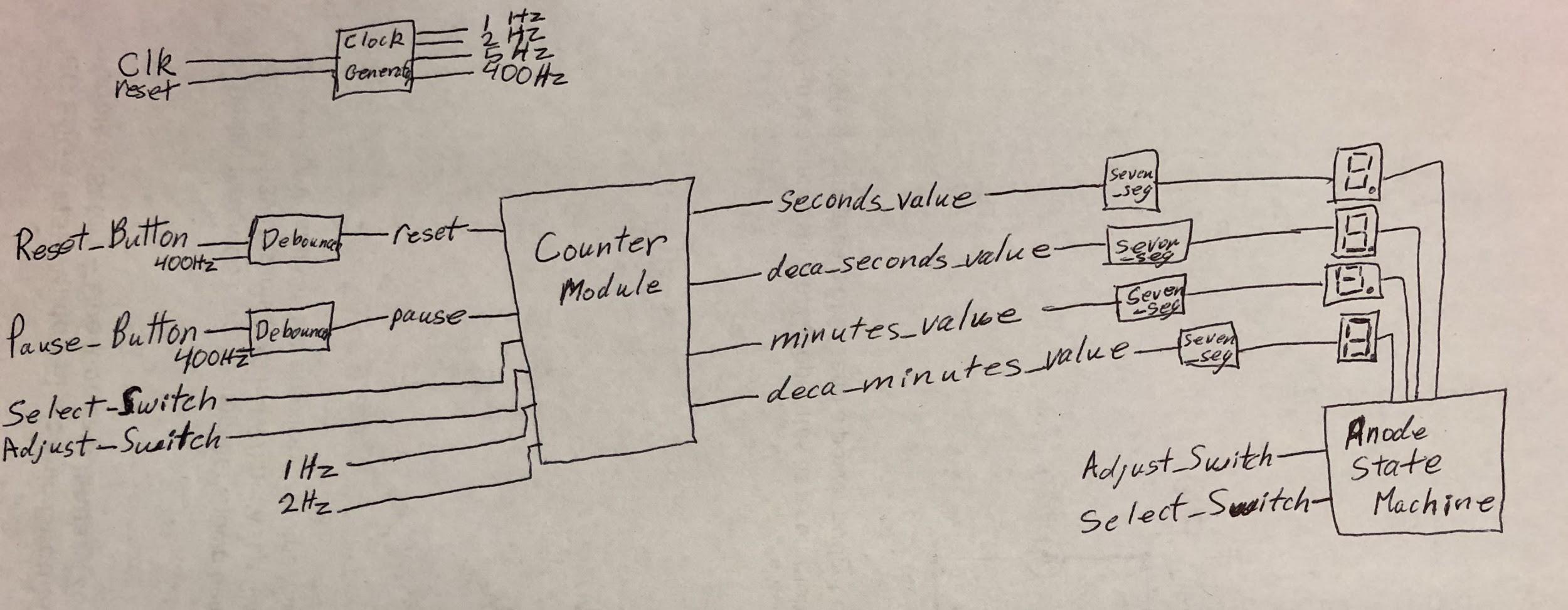


**Figure 2:** Mappings for the Adjustment Switch

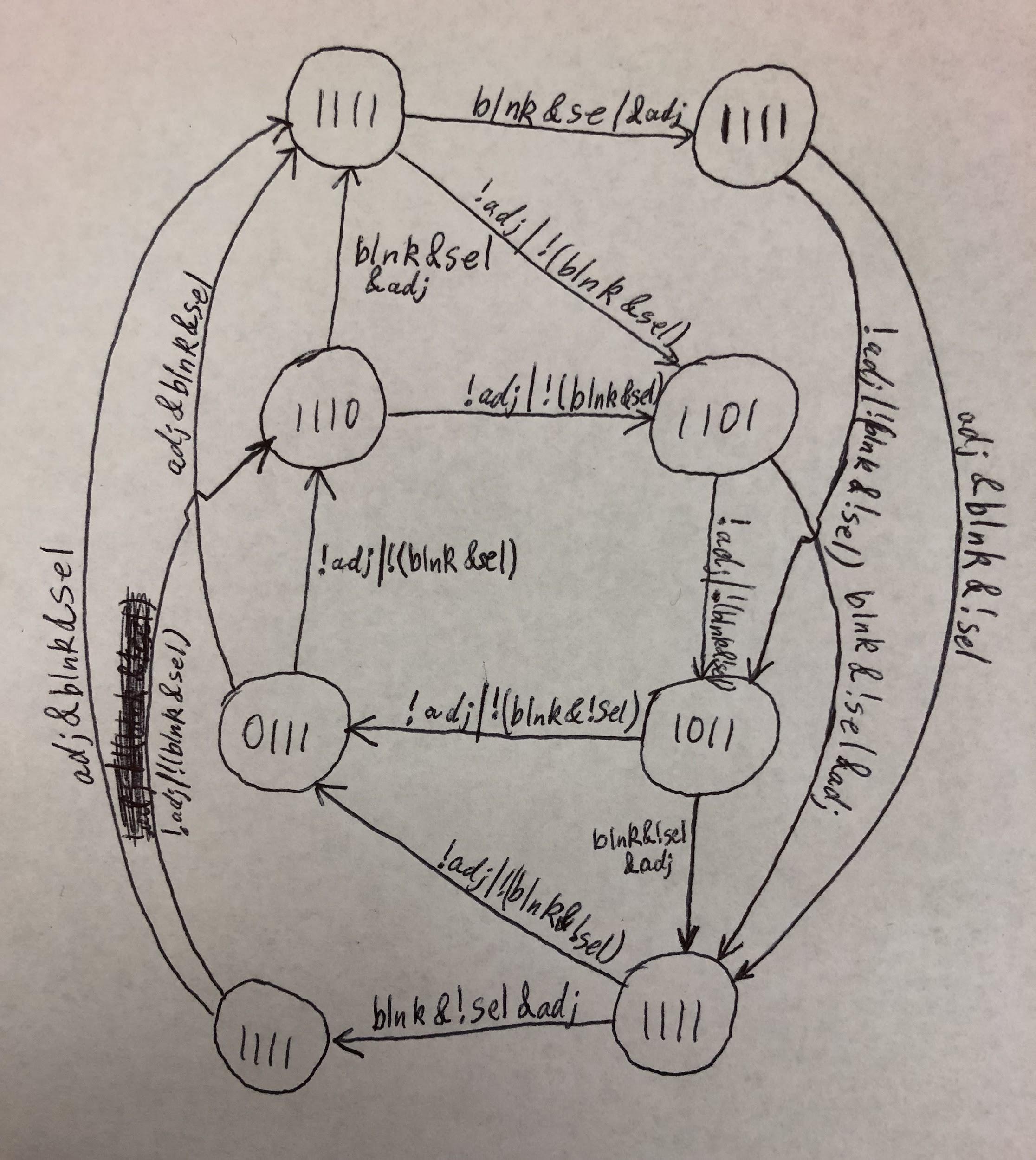
**Stopwatch Top Module:**

We separated functionality into 5 modules. We used one module (clock\_generator.v) to generate clock signals with different frequencies to handle blinking, incrementing of the counter, fast counter incrementation in adjust mode, and anode switching on the seven segment display. The clock generator module is described in detail below. We used a counter module (counter.v) to generate the values to display for both minutes and seconds. The counter module includes logic for handling adjustment mode, select, stop, and reset. This module is explained in further detail below. We used a third module to contain the logic for mapping counter values to the correct cathodes on the seven segment display. We used one instantiation of this module for each digit of the display. Our fourth module contains the logic needed to debounce signals from the stop and reset buttons. Our top module controls blinking and instantiates all the necessary instances of the previously described modules. A diagram of the high level logic associated with each of these modules is shown below in figures 3 and 4.

**Top Module Implementation:**

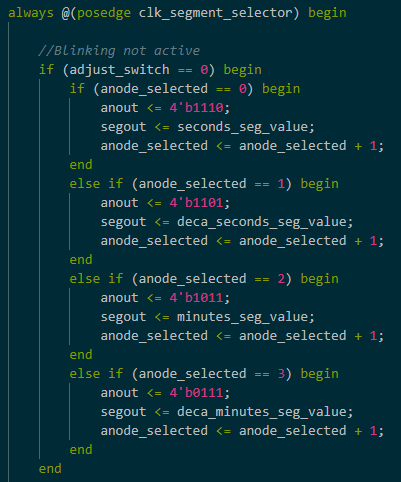
****

**Figure 3:** Schematic Showing Mapping of Inputs and Outputs From Top Module. The Anode State Machine is shown below in Figure 4.

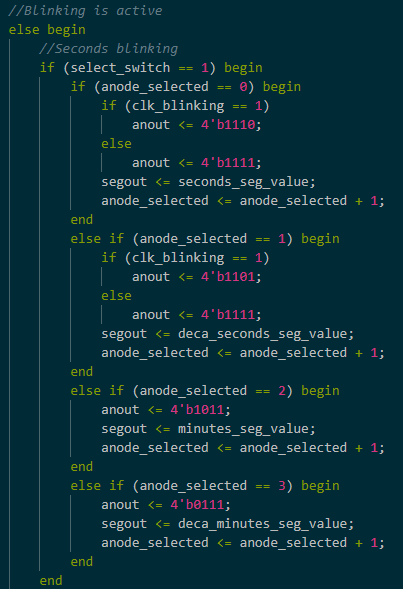
****

**Figure 4:** Top Module State Machine for Anodes. Anodes are selected if they are low in this case, so 1110 selects the anode for the seconds digit. Note: Signals listed as adj, sel, and blnk are short for adjust\_switch, select\_switch, and clk\_blinking signals respectively in the code and previous schematic.

The finite state machine used to control the anodes is implemented in the always block of the top module. The 400 Hz clock is used to cycle through anodes to display the correct values for each segment. There are two separate modes of operation for this stopwatch. For normal operation, we cycle through the anodes to display each of the digits in turn faster than the eye can detect to make it seem as if all 4 numbers are displayed at the same time. During the adjust mode either the seconds or the minutes are periodically turned off to blink the digits depending on which is selected using the select switch.



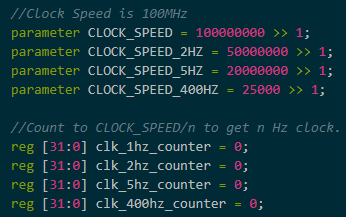
**Figure 5:** Verilog Code Segment for Normal Operation. Note: Anodes are selected when they are written low, so 1110 selects the seconds digit.



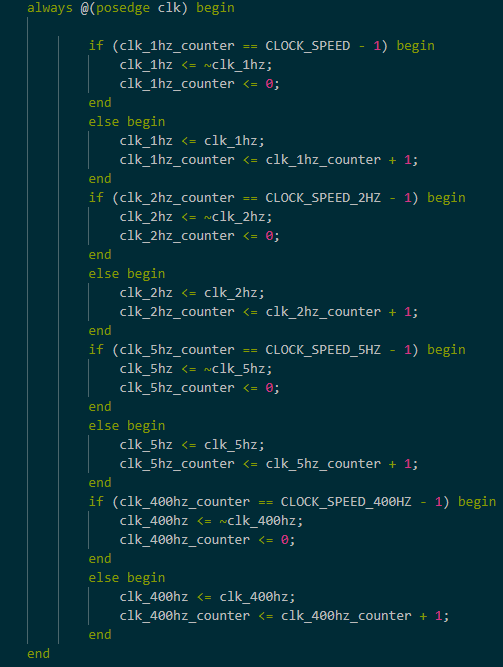
**Figure 6:** Verilog Code Segment for Blinking the Seconds Digits. Note: Similar logic was used for minutes. See the uploaded stopwatch\_top.v for full implementation.

**Clock Generator Module:**

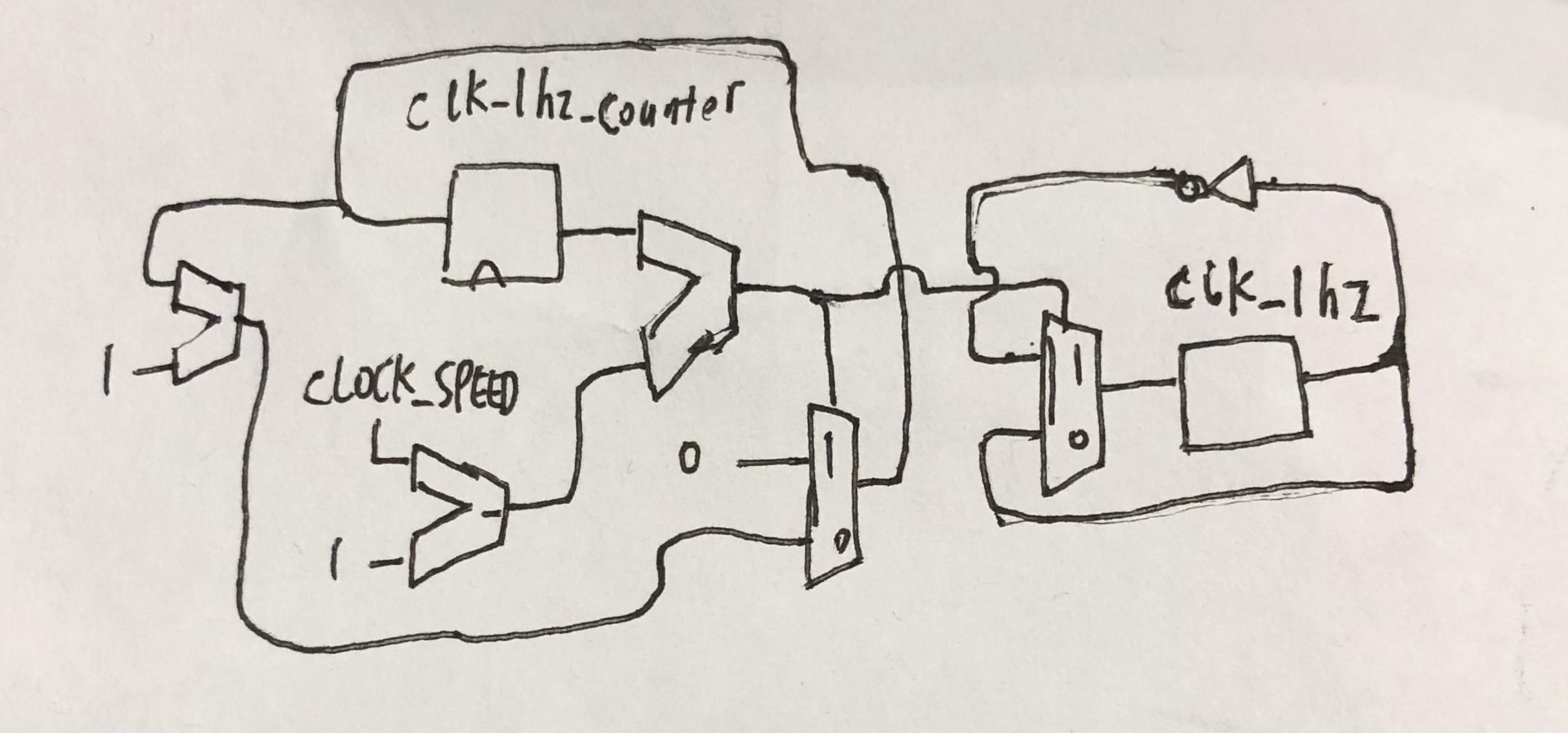
We used the clock generator module to generate four different clock frequencies: 1Hz, 2 Hz, 5 Hz, and 400 Hz. We used these clock frequencies throughout the other modules. To generate these frequencies, we essentially created 4 clock dividers using counters where we reset the value in the counter each time it reaches the correct number of pulses from the default 100 MHz clock in half of the period of the desired frequency. Each time we reset the clock, we toggled the output signal. This resulted in a waveforms with a 50% duty cycle at the desired frequency.



**Figure 7:** Clock Speed Constants and Counters Used to Generate Clock Signals.



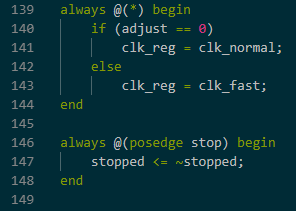
**Figure 8:** Logic for Incrementing Counter Registers and Toggling Output Signals.



**Figure 9:** Schematic for 1Hz Clock in the Counter Module. The other clocks used the same logic with different clock speed constants.

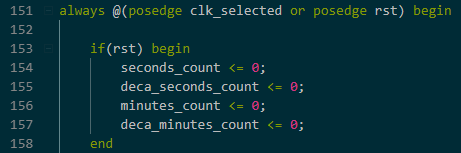
**Counter Module:**

The counter module is composed of logic for four separate counters: one for each digit. The deca\_minutes, minutes, and seconds counters count up to 9 before overflowing and the deca\_seconds counter counts to 5 before overflowing. The counters are linked so the faster signals trigger the slower numbers when they overflow. (Seconds overflows to deca\_seconds, deca\_seconds overflows to minutes, and minutes overflows to deca\_minutes.) The counter stops incrementing if the stop signal is high. If adjust mode is selected, only seconds or only minutes will increment at a faster clock rate of 2 Hz depending on the value of the select switch.

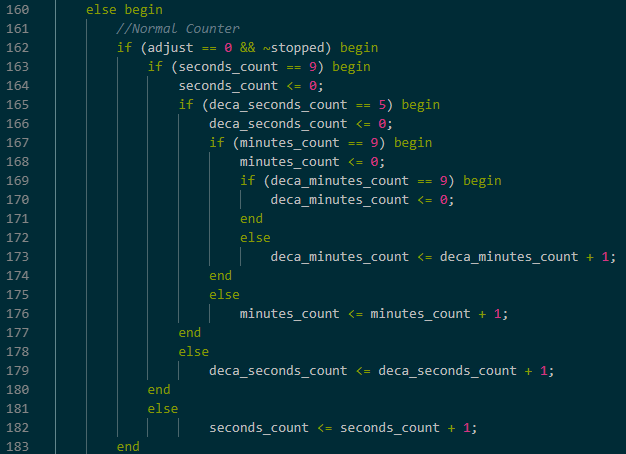
****

**Figure 10:** Code Segment for Stop and Adjust Modes.

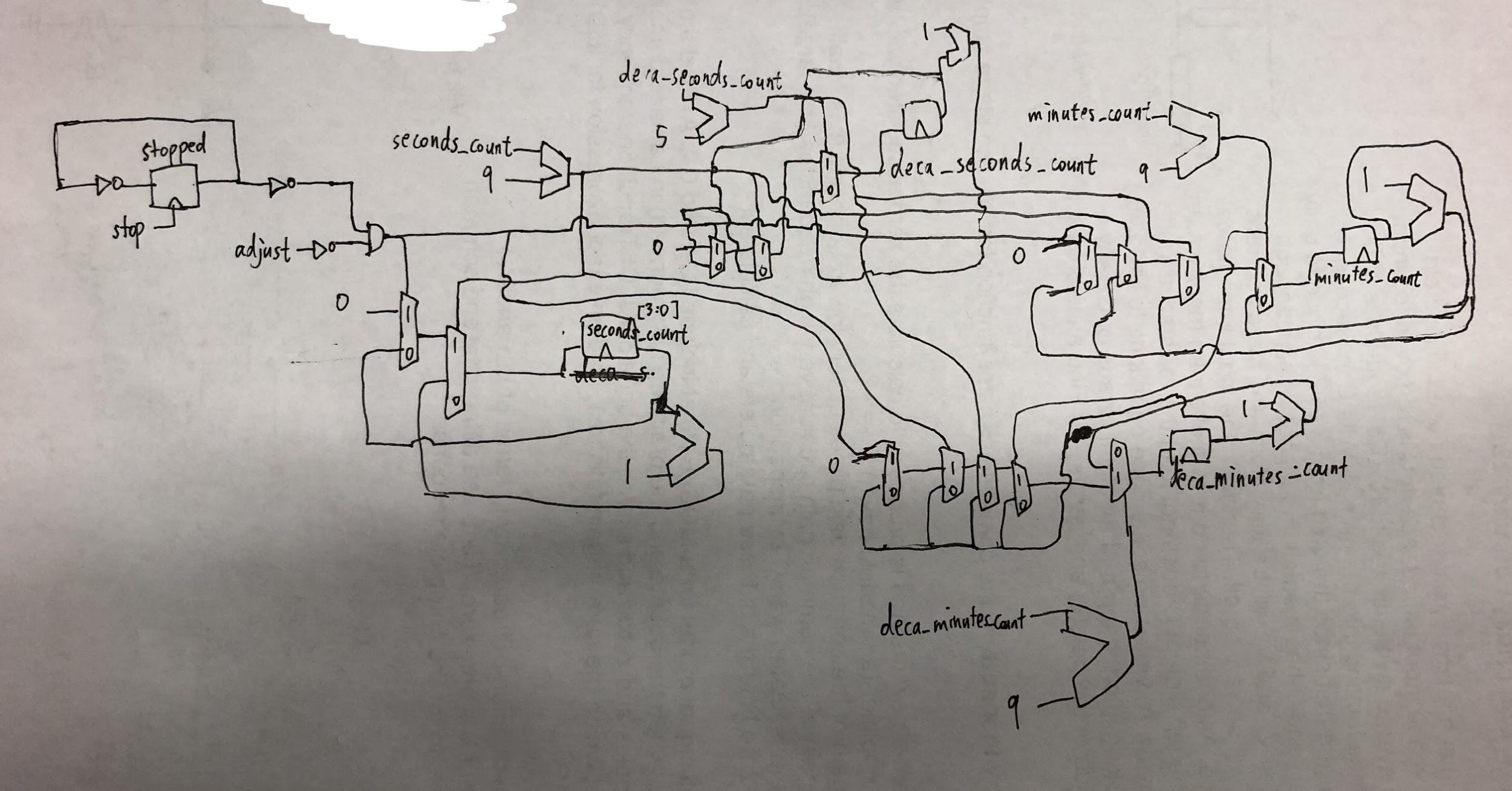
We used a series of if statements to determine when to increment seconds, deca\_seconds, minutes, and deca\_minutes and handle the fast incrementing that occurs when in adjustment mode. The logic is as follows. If reset is received set all values to zero.



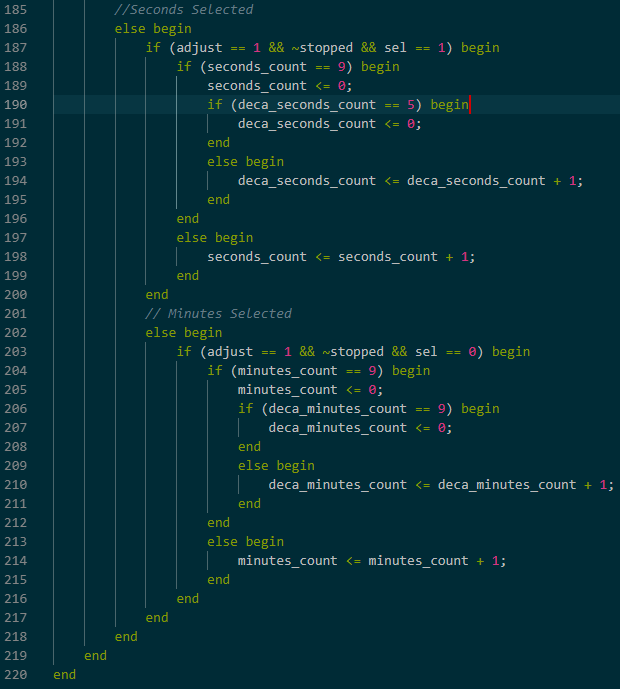
**Figure 11:** First Part of Main Always Block. This handles resetting of counters.



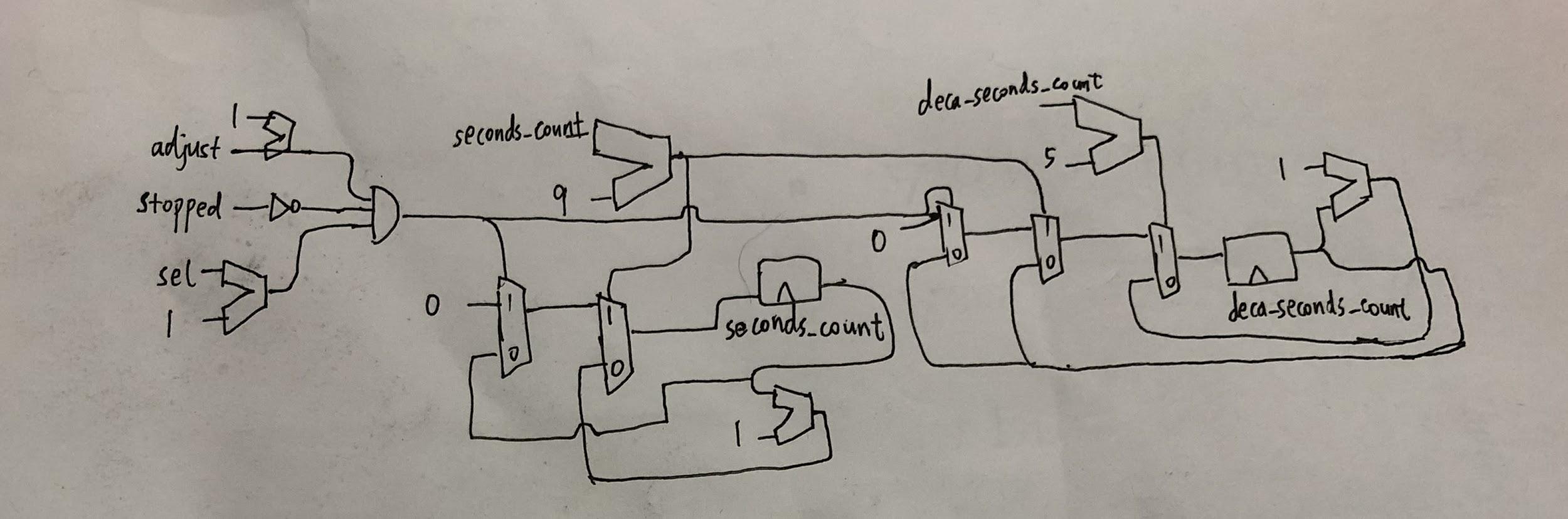
**Figure 12:** Implementation of Normal Counter Behavior as Described Above.



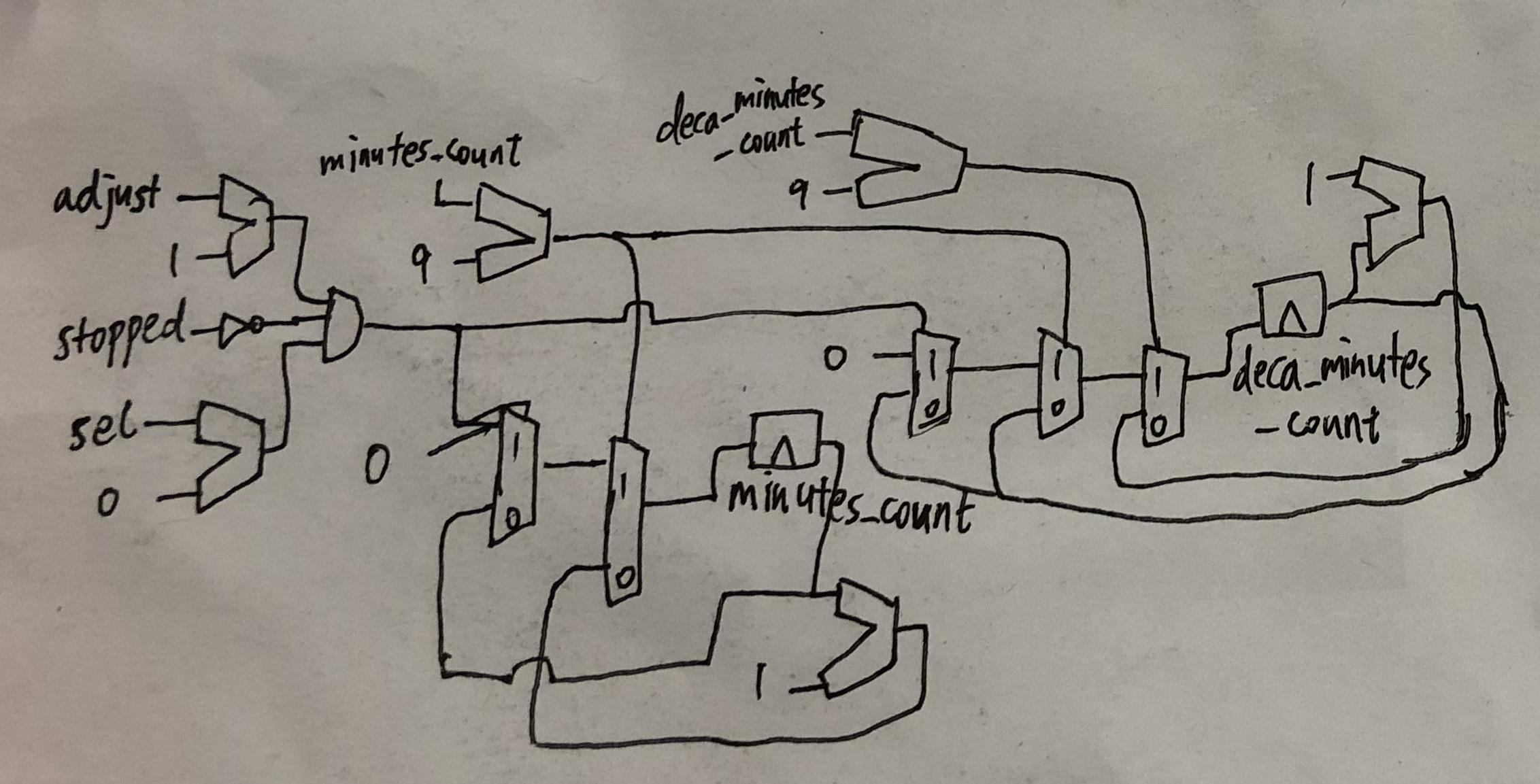
**Figure 13:** Schematic for Counter Module.



**Figure 14:** Implementation of Adjustment Counter.



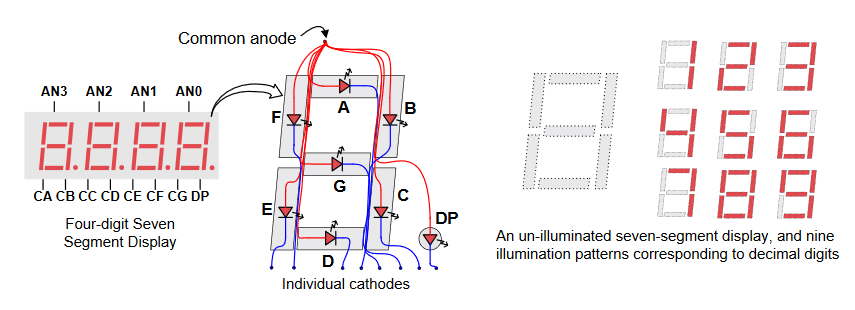
**Figure 15:** Schematic for Seconds Selected.



**Figure 16:** Schematic for Minutes Selected.

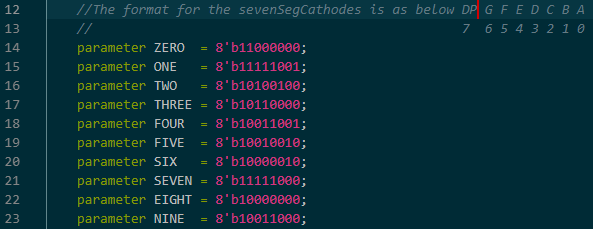
**Seven Segment Module:**

The seven segment display is split up into 8 different cathodes that share a common anode. Each of the cathodes is selected when the signal is written low. This module is responsible for taking in the output from the counter module for each digit and setting the associated cathode mapping to be used for the seven segment display associated with that digit.

****

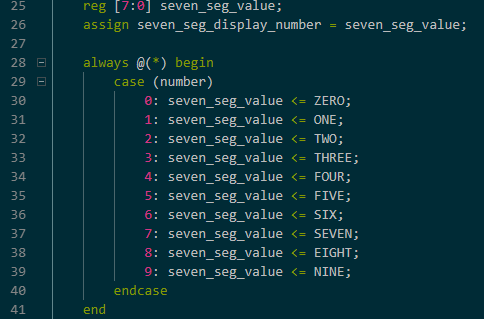
**Figure 17:** Mappings of Cathodes to Segment Leds. This figure is taken from the Nexys3 Reference Manual. We determined by trial and error that the a segment is turned on if both the pin connected to the anode and the pin connected to the cathode in question are set to 0.

We used the mappings in Figure 18 below to set each of the digits 0-9.

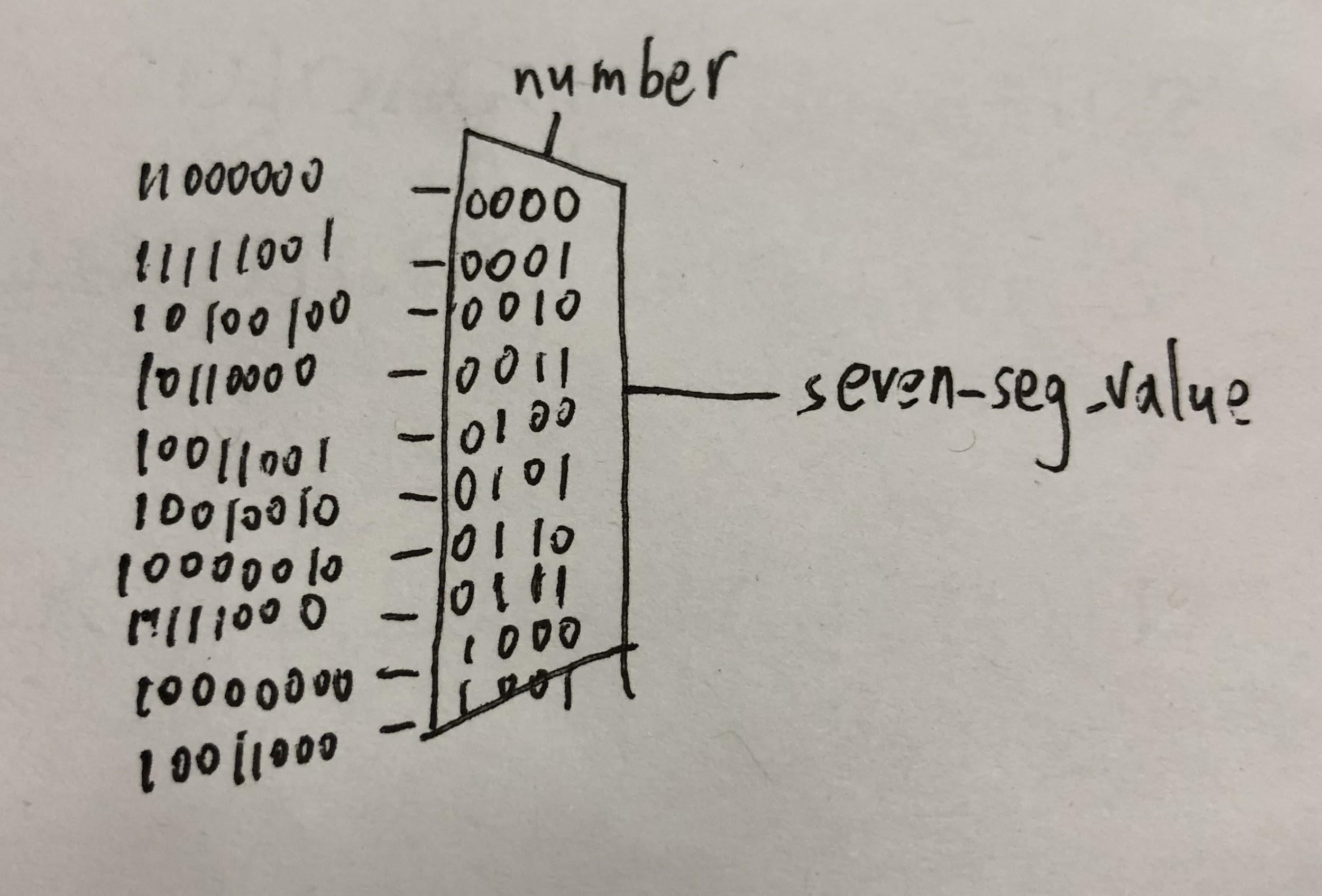


**Figure 18:** Segment mappings for numbers 0-9 by setting cathodes low for each segment they corresponded with according to figure X

This module converts the number it receives from the counter module into a segment mapping of cathodes. This mapping is used in the top module finite state machine to display the correct number on the seven segment display. The following figure shows the always block that performs this conversion.



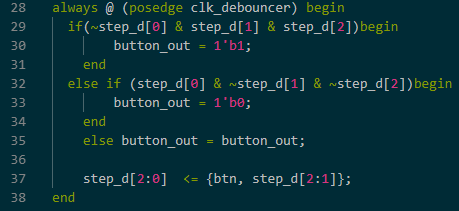
**Figure 19:** Case Statement to Set Seven Seg Value to the Correct Mapping.



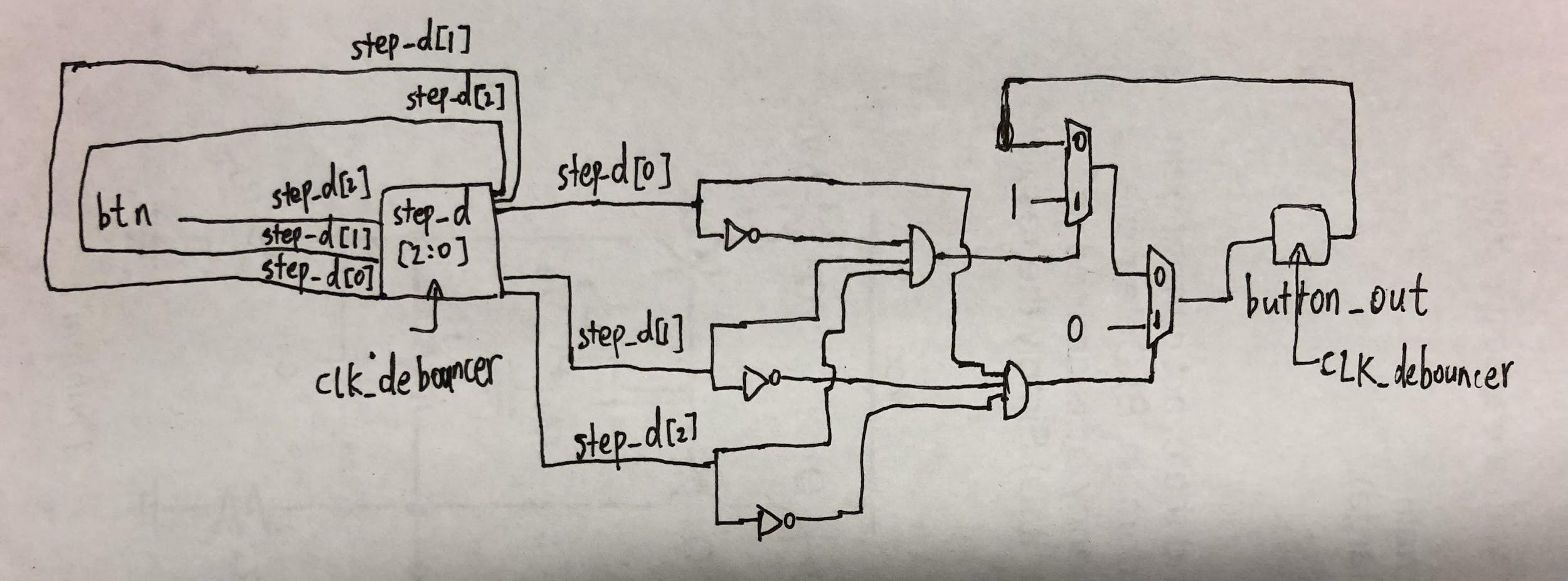
**Figure 20:** Schematic for Seven Segment Module. We used this relatively simple logic to convert a number value to the correct selection of cathodes to display the number on one of the seven segment display digits.

**Debouncer Module:**

We copied the idea of using a shift register to handle debouncing from Lab 2. The debouncer module uses a 400 Hz clock. Every 1/400 second, the current state of the button is stored in a register, high if the button is pressed, low if the button is not. Every 1/400 of a second, we right-shift the register by one, discard the rightmost bit and append the most recent state to the left. A sequence of states represented by [0, 1, 1] in the register means over the period of last 3/400 seconds, the button was pressed, and we set the output signal high. A sequence of states represented by [1, 0, 0] in the register means over the period of last 3/400 seconds, the button was released, and we set the output signal back to 0. This gives time for the bouncing to settle before registering a new value.



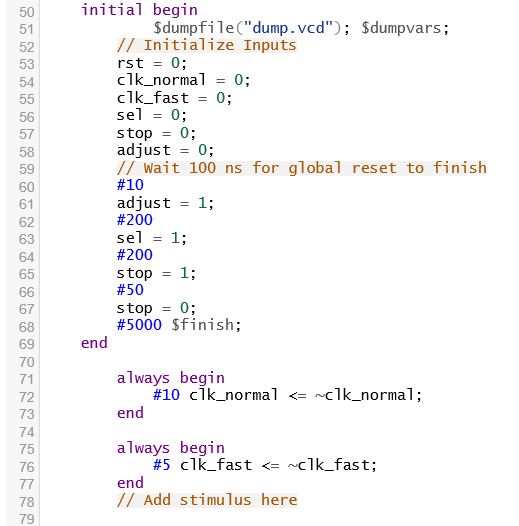
**Figure 21:** Verilog Implementation of Debouncer Logic.



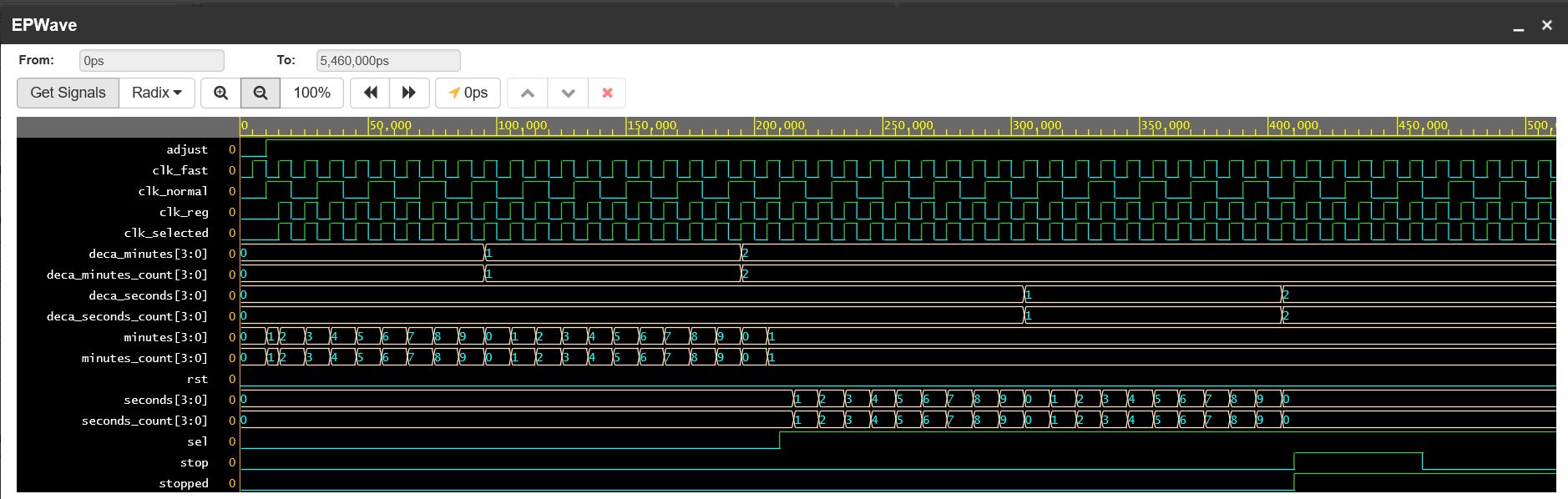
**Figure 22:** Schematic for Hardware Implementation of Debouncer.

**Section 2: Simulation Documentation**

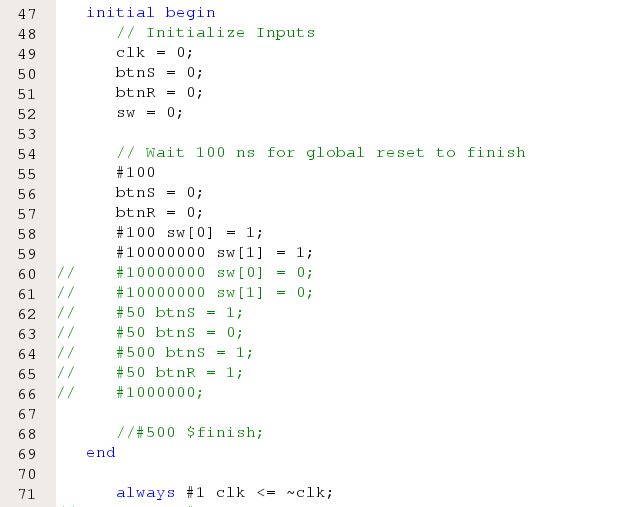
We ran a large number of simulations to verify correct operation of our modules. In addition to running simulations on the top module with different combinations of input signals to test operation of the system as a whole, we also used testbenches for the counter and debouncer modules, since those were more difficult to test in the system as a whole.



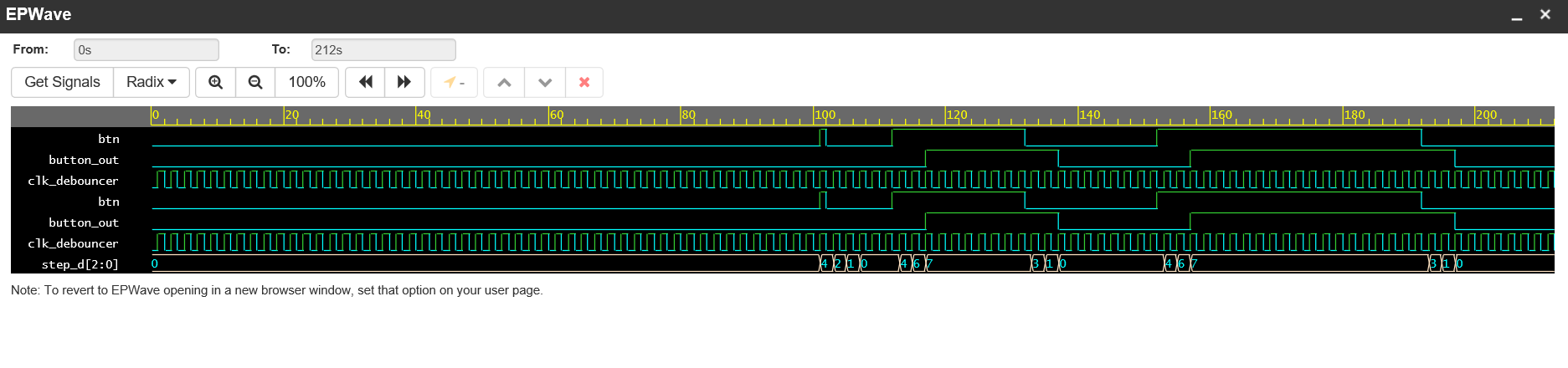
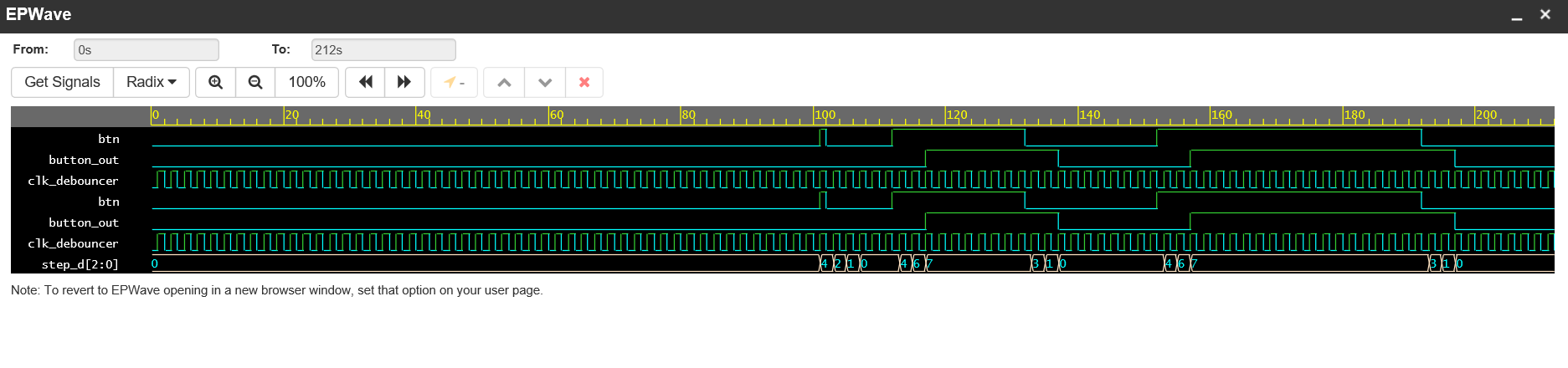
**Figure 23:** Screenshot from Counter Testbench. See uploaded code for complete version.



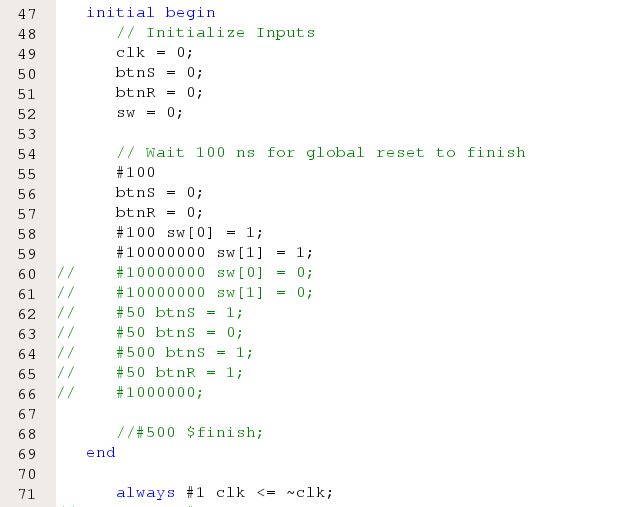
**Figure 24:** Screenshot of Counter Waveforms Verifying Correct Operation.



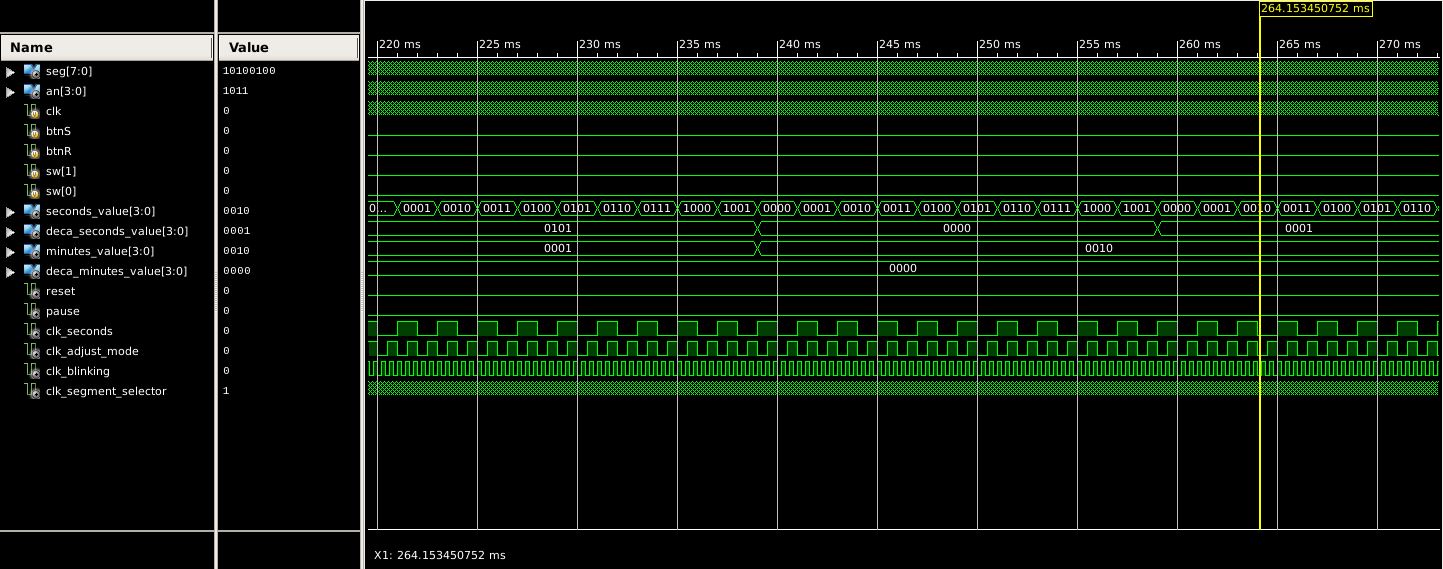
**Figure 25:** Screenshot of Debouncer Testbench. See uploaded files for complete version.



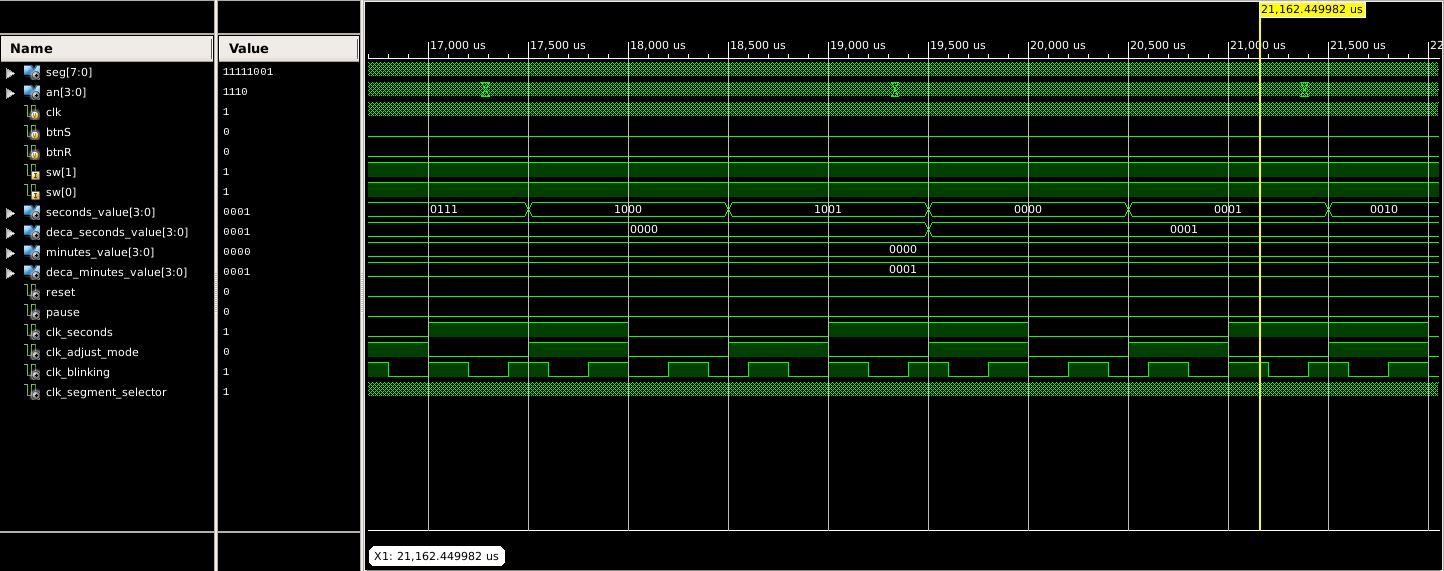
**Figure 26:** Screenshot of Debouncer Waveforms Verifying Correct Operation. This waveform is from EDA Playground and has been truncated in the middle to make viewing easier.



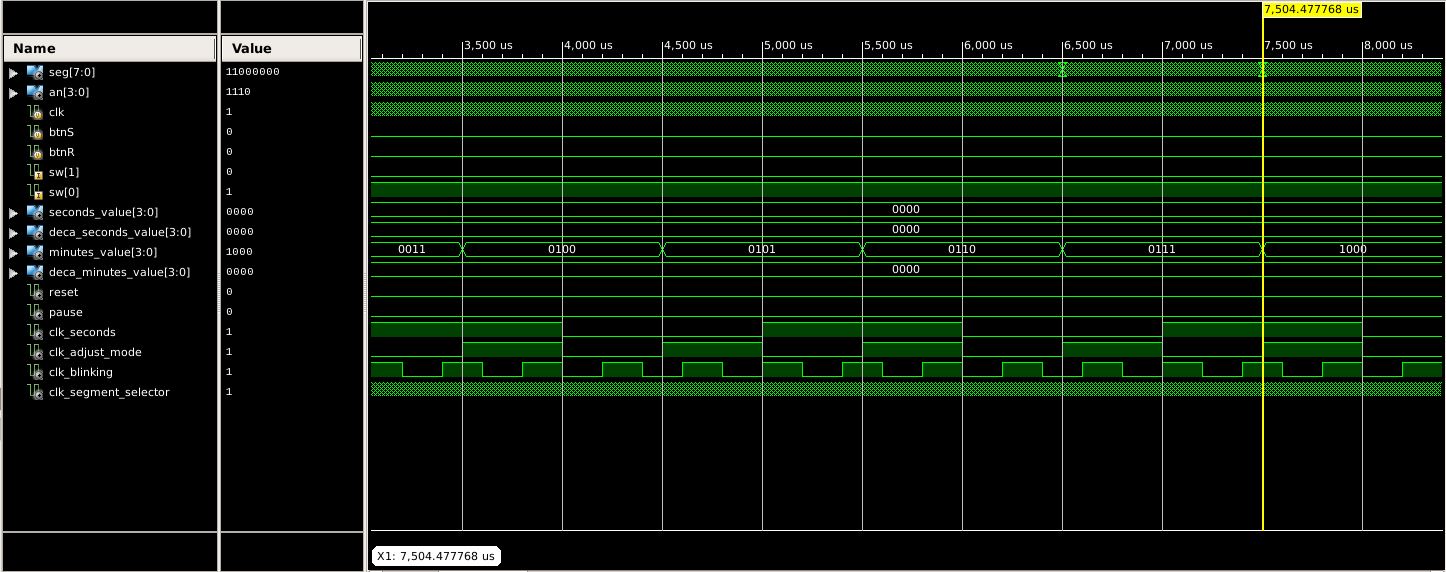
**Figure 27:** Screenshot From Top Module Testbench. See uploaded files for complete code. This shows the logic used to drive signals for the waveforms shown below. Signals sw[0] and sw[1] were set low for normal operation simulation.



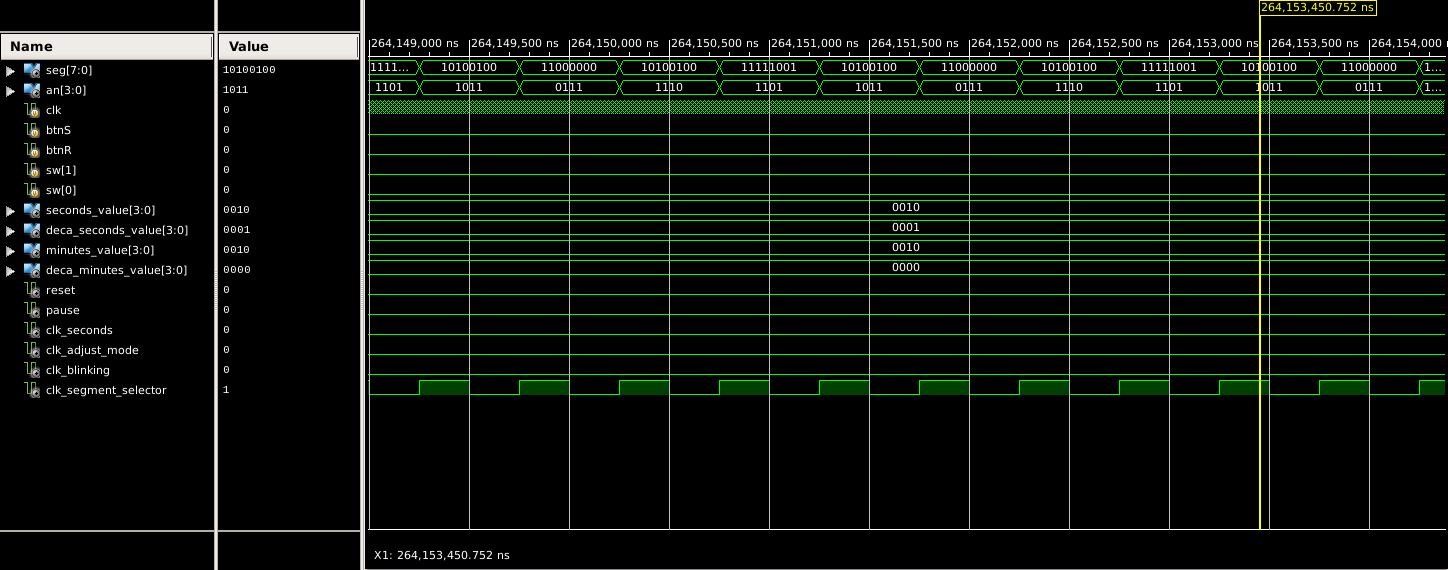
**Figure 28:** Screenshot of Top Module Waveform Verifying Correct Operation. This screenshot shows that minutes properly ticks over to the next value when deca\_seconds is 5 and seconds is 9. It is also clear that the seconds is incrementing every posedge for the 1 Hz clock.



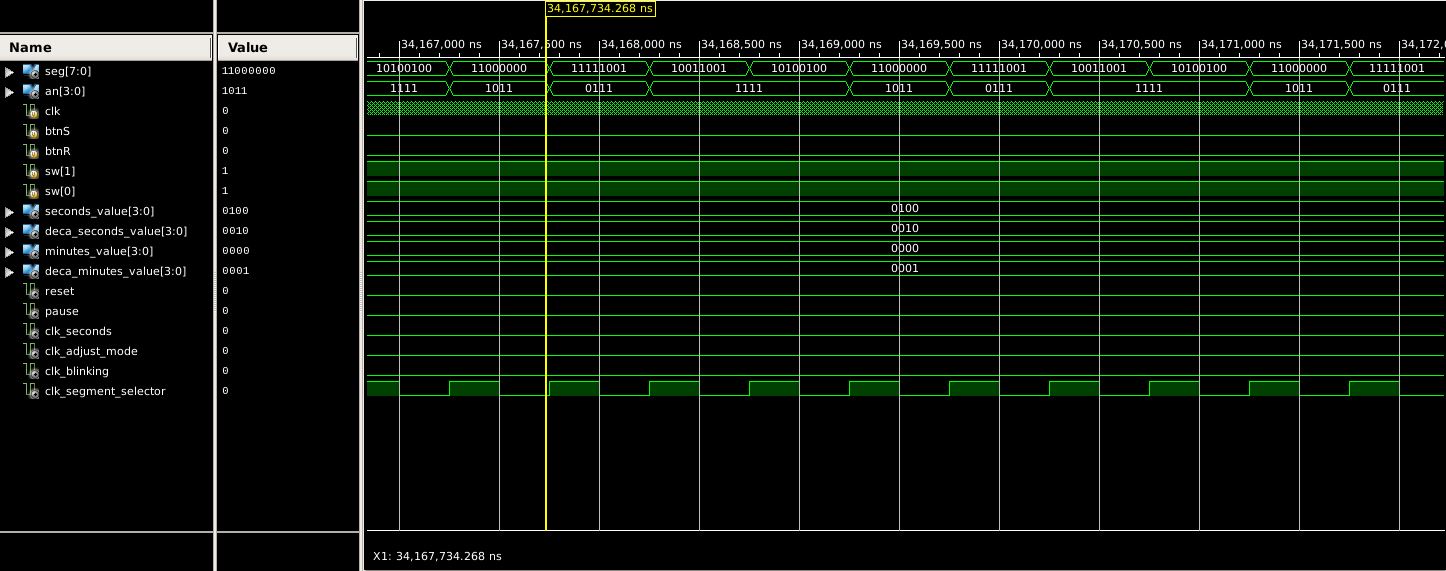
**Figure 29:** Adjusting Seconds. This figure shows that when adjust mode is active and select is set to seconds, seconds and deca\_seconds will tick up, but minutes will not. It is also shows that the faster 2Hz clock is driving these changes.



**Figure 30:** Adjusting Minutes. Similar to Figure 29, when minutes is selected, the faster clock is triggering the incrementation of minutes and deca\_minutes while seconds remains unchanged.



**Figure 31:** Switching Seg and An Signals. This figure shows that the seg values corresponding to seconds, deca\_seconds, minutes, and deca\_minutes are as expected from Figure 19.



**Figure 32:** Blinking Activated for Anodes. This screenshot shows a segment where the anodes for seconds are held low. This behavior turns on and off as specified in the spec. The same logic is used for minutes, but with minutes being affected instead of seconds.

**Section 3: Conclusion**

The stopwatch project worked after we fixed a number of problems with our original code. We learned the importance of using testbenches to examine our code as we develop our project to avoid logic errors that are difficult to troubleshoot. This problem seems relatively simple from a high level, but becomes very complicated if there are multiple errors. Once we started using this more efficient development technique, we were much more efficient.