Lab 3: Stop Watch

CS M152A Lab 3
TA: Gu, Hongxiang
May 20, 2019

Nathan Marsh, [illegible]
Denise Wang, [illegible]

Introduction

The goal of this lab was to design a stopwatch circuit that counts minutes and seconds using the seven segment display, buttons, and slider switches on the Nexys™3 Spartan-6 FPGA Board. The seven-segment display on the Nexys™3 Spartan-6 FPGA Board will display the minutes in the left two digits and the seconds in the right two digits. This was done by going through the complete FPGA design flow.

Design Description

The stopwatch consists of two modes: the normal mode and the adjust mode. The two user-defined buttons on the stopwatch controls the pause and reset function. In the normal mode, the pause button will pause all of the counters displayed once the button is pressed, and once it is pressed again, the counters will continue. In the adjustment mode, the pause button sets the counter digits to the corresponding value. The reset button will restart all of the counters displayed back to the initial state 00:00.

SEL	Selected
11	Tens Digit, Minutes
10	Units Digit, Minutes
01	Tens Digit, Seconds
00	Units Digit, Seconds

Table 3.1- SEL. This table displays SEL inputs, the two switches that will choose among the four individual digits to be adjusted in the adjustment mode.

ADJ	Action
0	Stopwatch behaves normally
1	Stopwatch stops and selected digits are adjustable

Table 3.2- ADJ. This table displays the ADJ inputs, where each switch indicates whether the stopwatch is in adjustment mode or normal mode.

The stopwatch is in adjustment mode when the ADJ input is set to logic high (1) and in normal mode when the ADJ input is set to low (0). In the adjustment mode, all of the

counters of the stopwatch is paused, and the selected digit (digit chosen by the two SEL switches) of the counter can be set to a specific number selected by NUM. NUM are the four switches that represent binary numbers. To set the value of selected digit, the pause button should be pressed. The digits of the counters that are not selected remain paused while the digits being selected remain blinking.

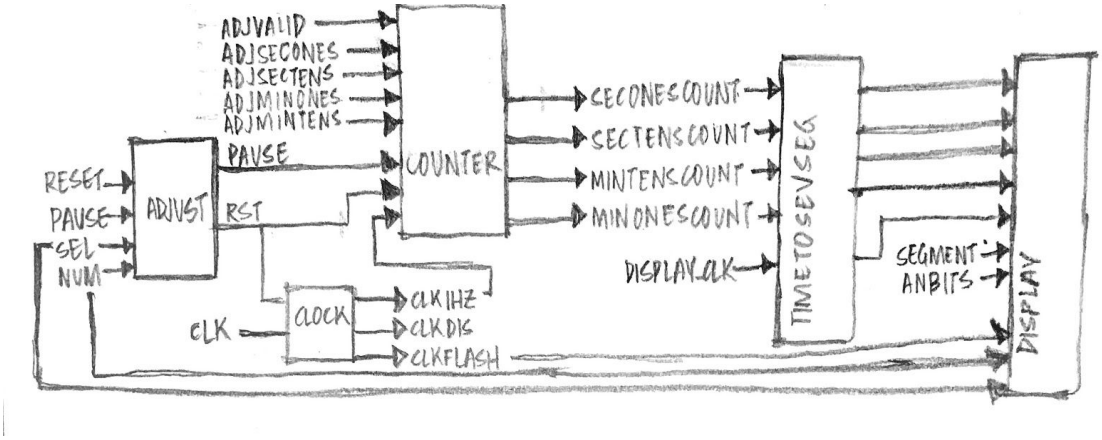

Figure 3.1- The block diagram above displays the design of our implementation of the stopwatch overall.

For the debouncers module, the buttons are bouncy, meaning that once a button is pressed, an oscillation of signals due to poor metal contact is produced. This causes a considerable amount of noise in the input from the physical contact instability. The noise in the input could be filtered out by sampling at a frequency lower than that of the noise.

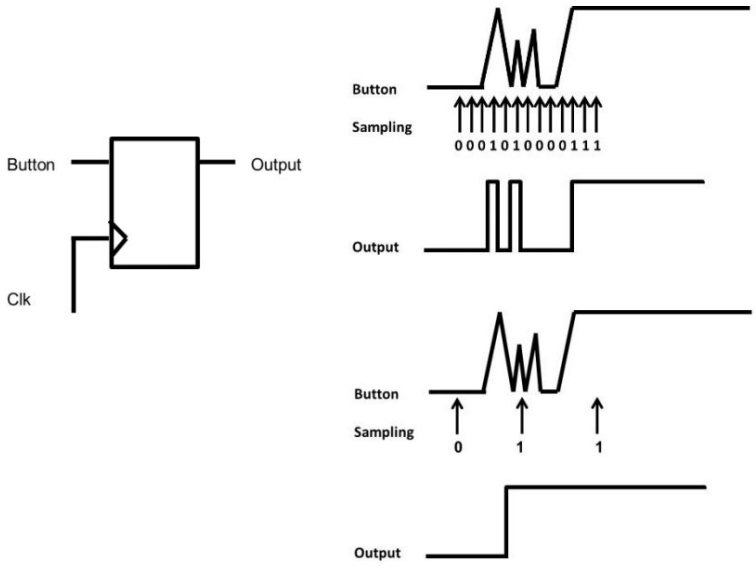

Figure 3.2- Noise of a push of a button is filtered out. In the figure above, once button is sampled at each of the positive edges, the following output is displayed. If sampled at a very fast frequency, the sequence would have extra 0s and 1s from the noise, so the sequence should be sampled at lower frequencies.

Aside from the noise caused by physical instability, another issue is metastability due to the asynchronous nature of the button and slider switch input. The input signals may come at any unpredictable time, which leads to different outputs to other modules and signals. This could be prevented by using a flip-flop after the asynchronous input and regarding the output as the input so that all other modules are consistent.

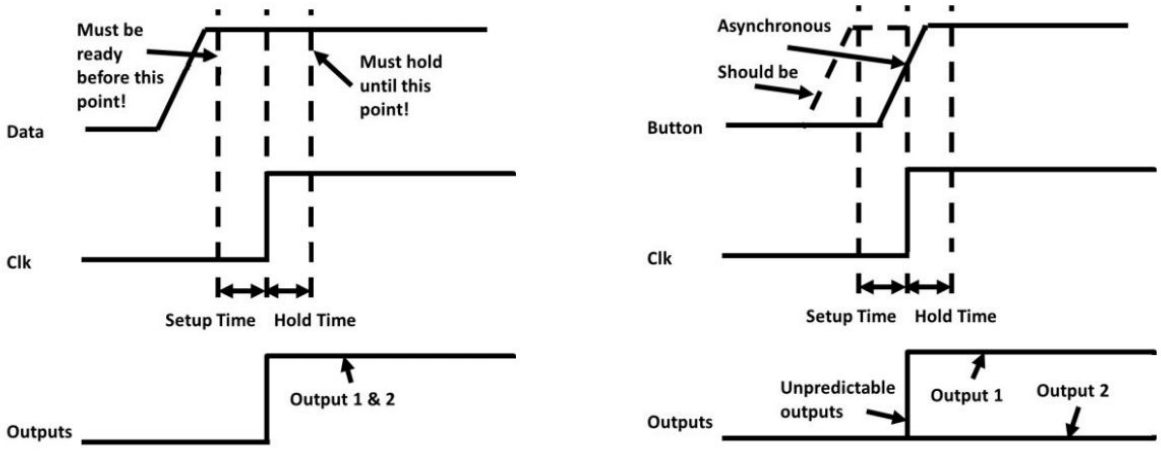

Figure 3.3- Synchronizing the signal to the frequency of the clk, it stabilizes the inputs and regards the output as inputs, removing the unpredictable outputs.

Design Implementation

Clock Module

The clock module we wrote takes the 100 MHz clock as input and divides it into three different clock signal outputs. The first clock signal we derived was the 1 Hz clock signal for the clock counter to increment every second. Next, we had a display clock signal that was around 381 Hz which helped to alternate between each digit in the seven segments display so that all the different digits could be shown at the same time. The third clock signal was used to flash the digit selected in adjust mode and was set to around 3 Hz. The first clock was implemented by inverting the clock signal when counting to 50 MHz. The second two signals were implemented by looking at the first bit in a certain length bit register. We calculated that every x number of bits we used

decreased the frequency of the clock by 2x. We were able to generate the 381 Hz and 3 Hz signals with 18 and 25 bits respectively.

Counter Module

The counter module contained all the control for the digits on the seven-segment display. The inputs are the 1 Hz clock, a reset signal, a pause signal, an adjust signal, an adjusted valid signal, and the four adjusted digits. The outputs for the module the were the digits to be displayed in the seven-segment display. This module was implemented by setting initial seven segment display digits to 0 and using an always block to determine whether to increment, pause, adjust, or reset the digits. To increment the display digit values, we used logic about the lower order digits to determine if the digit needs to be incremented. For example, for the tens digit of the minutes to be incremented the ones digit of the minutes and seconds must be nine and the seconds tens digit must be five. When the pause or adjust signals are asserted the next digits are set to the current digits. The adjust valid signal is asserted when the number must be changed because pause was pushed in adjust mode. In this case, the seven segment digits are set to adjust digits that come from the adjust module. If reset is asserted the digits are set to 0.

Display Module

The display module called timetosevseg.v takes as input the digits to be displayed, the display clock, the flash clock, the sel, adjust, reset, and pause signals, and it outputs the segment and an bits. This module has a shift register containing 0111 which shift at the positive edge of display clock. When a bit in the an bit is 0 that means that that digit is the one to be displayed., so if we set the an bits to this shift register we would get the stopwatch without the adjust functionality working. To get the adjust functionality we must logical or the shift with a flash mask which determined which bit should be flashing and set the selected bit to one if the adjust signal is asserted and we are at the positive edge of the flash clock. The segment is obtained by using the digit to seven segment display module by inserting the digit to be displayed and getting the segment output.

Decimal Digit Value	7-Bit Display Value
0	1000000
1	1111001
2	0100100
3	0110000

4	0011001
5	0010010
6	0000010
7	1111000
8	0000000
9	0010000

Table 3.3- The table above presents the conversion between decimal digits to the seven bit display value needed for the seven segment display. The zero represents an illuminated segment and the one represents an unilluminated segment. The 7 bits of the display value is in the order of GFEDCBA (see Figure 3.4 below).

Digit to Seven Segment Display Module

This module takes a digit as input and outputs the segment display corresponding to that digit. This module was implemented with a simple case statement that mapped from digit to seven segment display bits.

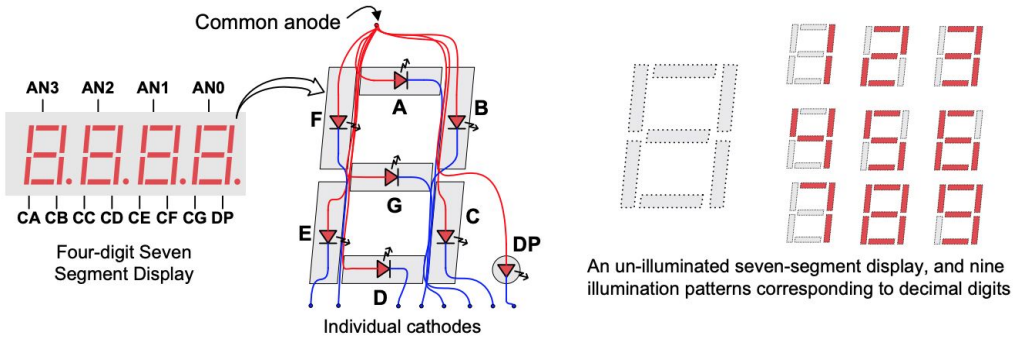

Figure 3.4- The figure above presents each segment of the four digit seven segment display.

Adjust Module

The adjust module controls what digits should be sent to the counter to be updated. As inputs it takes the display digits, the pause signal, the adjust, the sel, and num control signals from the switches and outputs the adjusted counter bits and an adjust valid signal. The module also checks if the number is in range by checking if the num is below 10 if it's a ones digit and below 6 if it's a tens digit. The numbers are set based on the adjust valid signal which is set if the num is in range, the adjust switch is asserted, and the pause button is pushed. The output digits are determined by the sel bits, if the

sel bits correspond to the digit then set the digit to NUM otherwise set the digit to the corresponding input digit.

Nexys3 Module

The nexyx3 module is that top module that connects the ucf file to the modules that were previously described. As input it takes the switches, the pause button, the reset button, and the clock. The module outputs, the seven segment display segments and the an bits. The module also debounces the pause button for better pressing and creating a pause valid signal which indicated that the button was pushed. A pause signal is also generated based on the pause valid signal to determine if the display should remain paused. The module then instantiates the clock module, the counter module, the display module, and the adjust module and properly wires them together sending the output of the counter to the display module and adjust module, sending the output of the adjust module back to the counter module, and sending the clock module outputs to counter and display modules.

Simulation Documentation

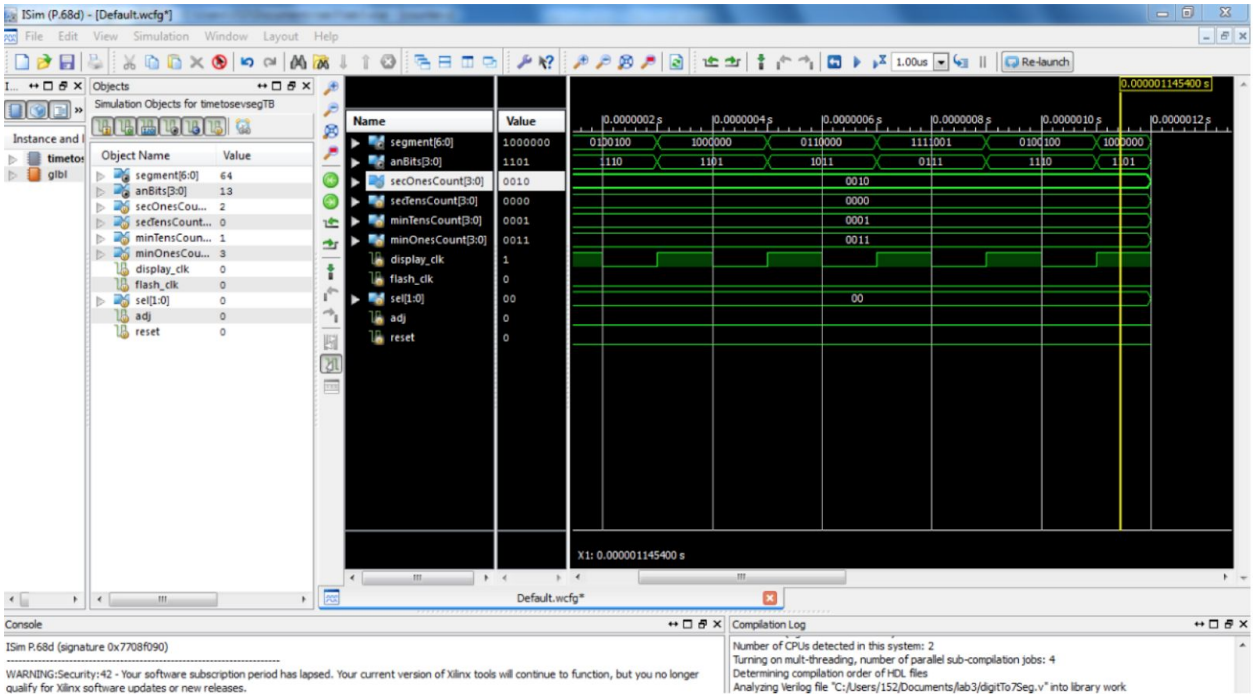

FIgure 3.5- Timetosevseg. This screenshot displays the waveform of the timetosevseg module. The inputs include [3:0] secOnesCount, [3:0] secTensCount, [3:0] minOnesCount, [3:0] minTensCount, display_clk, flash_clk, [1:0] sel, adj, pause, and

reset. The right side represents the resulting simulation of each of the test cases inputted, with the outputs being the [6:0] segment and [3:0] anBits.

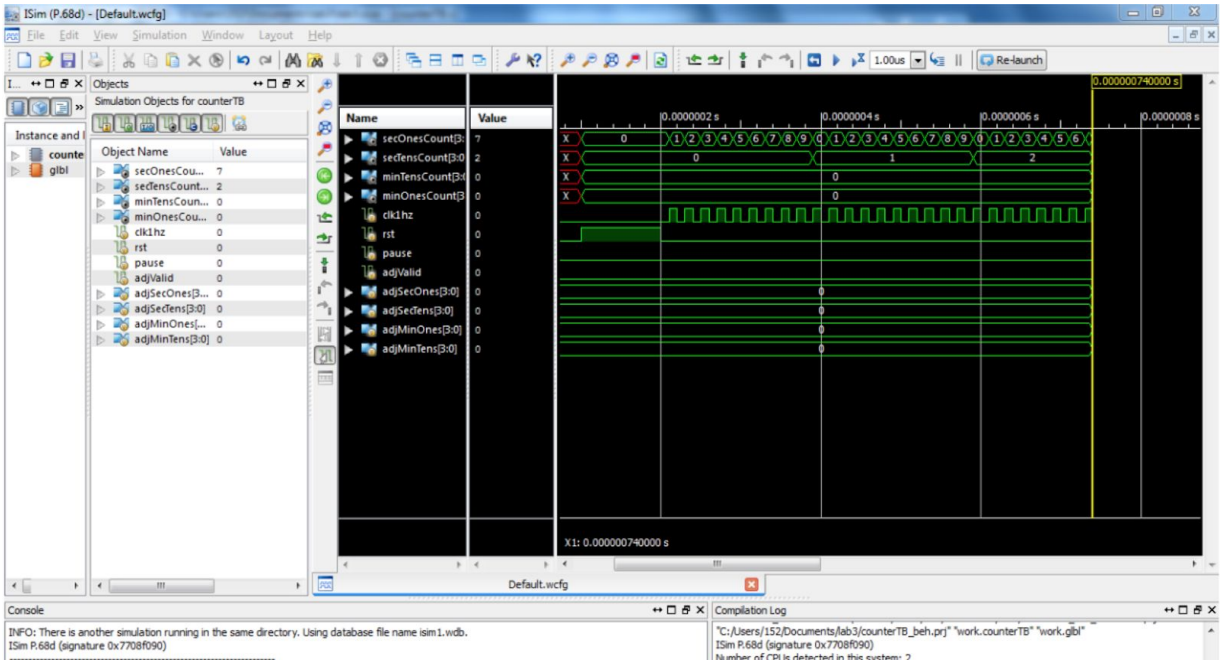

Figure 3.6- This screenshot displays the waveform of the counter module. The inputs include the clk1hz, rst, pause, adj, adjValid, [3:0] adjSecOnes, [3:0] adjSecTens, [3:0] adjMinOnes, and [3:0] adjMinTens. The right side represents the resulting simulation of the test cases inputted for the secones and sectens output.

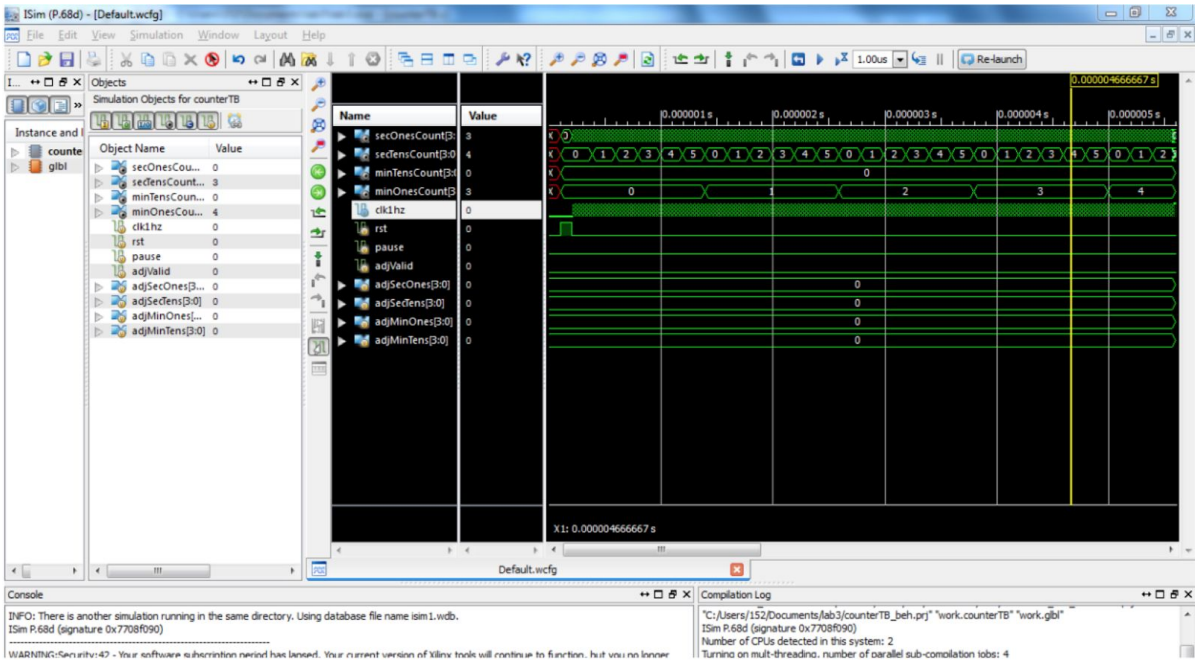

Figure 3.7- This screenshot displays the waveform of the counter module. The inputs include the clk1hz, rst, pause, adj, adjValid, [3:0] adjSecOnes, [3:0] adjSecTens, [3:0] adjMinOnes, and [3:0] adjMinTens. The right side represents the resulting simulation of the test cases inputted for the minones output.

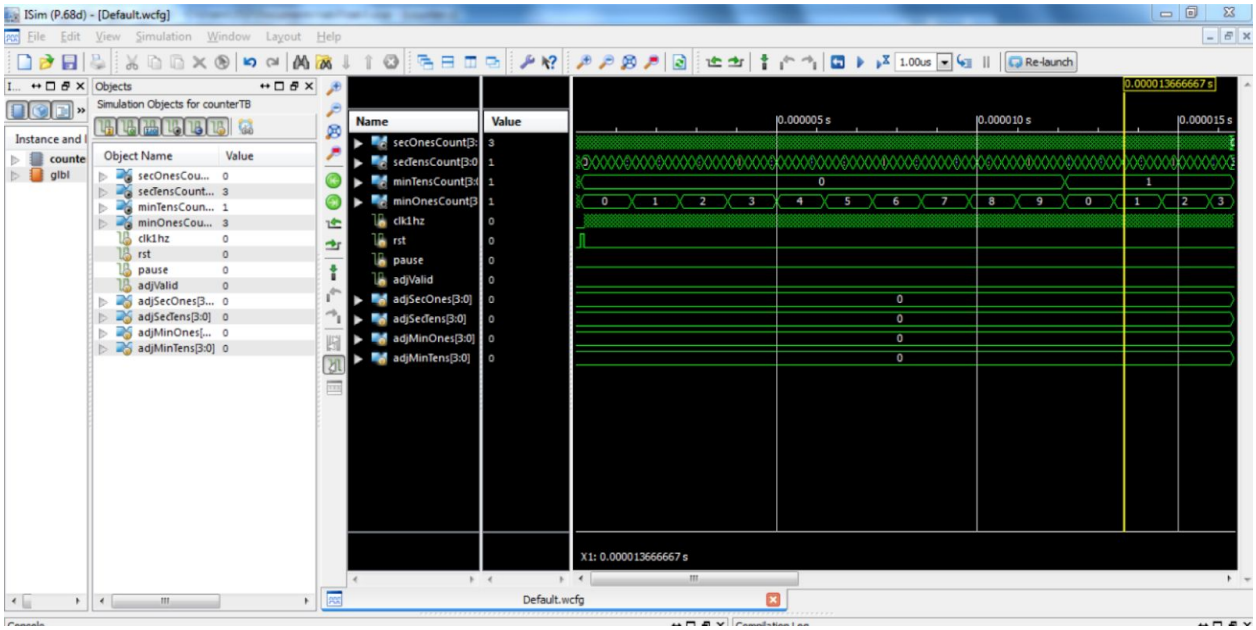

Figure 3.8- This screenshot displays the waveform of the counter module. The inputs include the clk1hz, rst, pause, adj, adjValid, [3:0] adjSecOnes, [3:0] adjSecTens, [3:0] adjMinOnes, and [3:0] adjMinTens. The right side represents the resulting simulation of the test cases inputted for the mintens output.

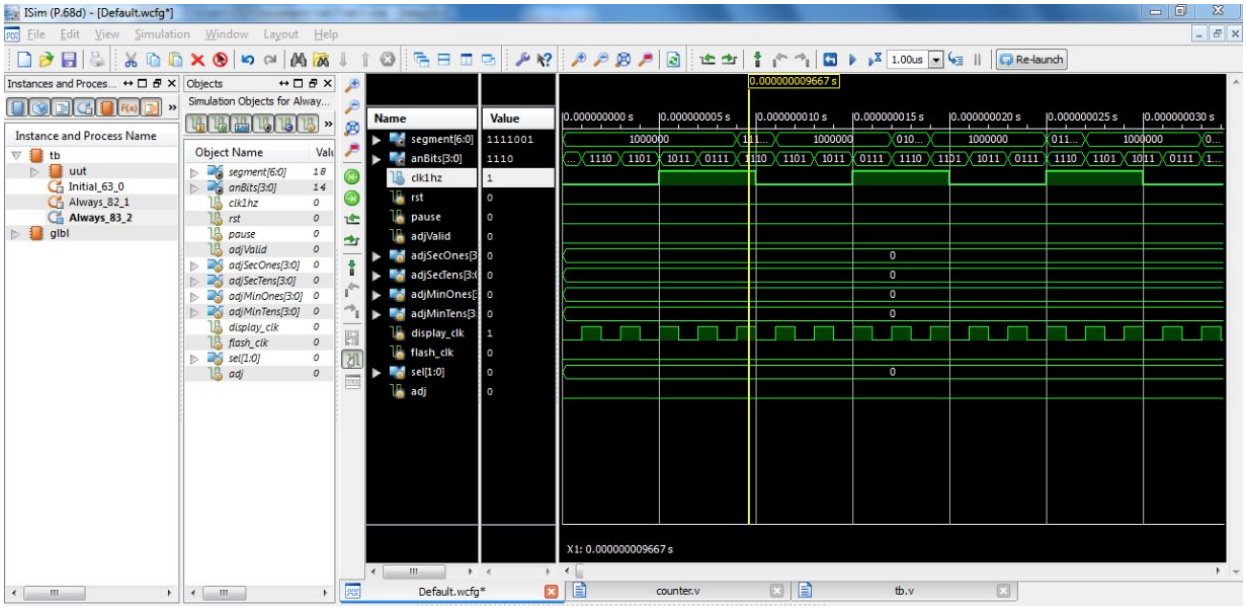

Figure 3.9- This screenshot displays the waveform of the countplusdisplay module. The inputs include the input clk1hz, rst, pause, adjValid, [3:0] adjSecOnes, [3:0] adjSecTens, [3:0] adjMinOnes, [3:0] adjMinTens, display_clk, flash_clk, [1:0] sel, and adj. The right side represents the resulting simulation of the test cases inputted for the mintens output. This module is for simulation purposes only.

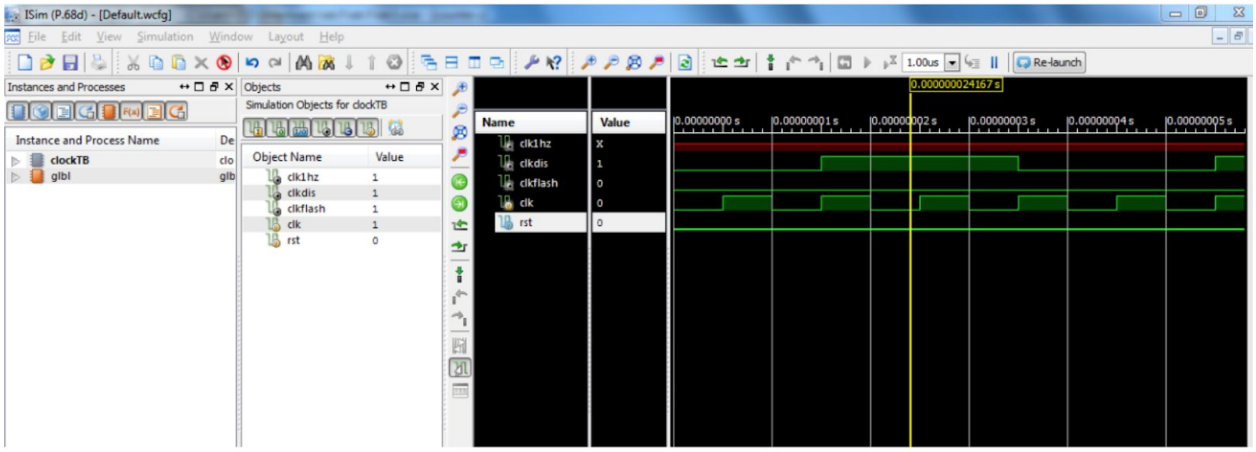

Figure 3.10- This screenshot displays the waveform of the clock module, where the inputs are the clk and rst. The right side represents the resulting simulation of each of the test cases inputted for the clk1hz, clkdis, and clkflash outputs.

Conclusion

Overall, each of the modules combined to create a stopwatch using the seven segment display, buttons, and slider switches on the Nexys™3 Spartan-6 FPGA Board. The counter module increments the seconds count every second, and once it reaches 60 seconds, the minutes counter will start to increment as well. The clock module consisted of the three different clocks: a 1 Hz clock, a much faster clock (50 – 700 Hz), and a clock for blinking in the adjust mode (>1 Hz). The seven-segment display module was implemented using the Nexys3 Reference Manual and takes as input the digits to be displayed. The debouncer module checks ensure that the buttons are implemented properly without issues such as noise and metastability. The UCF module consists of the instructions given to the FPGA implementation tools to direct the mapping, placement, timing or other guidelines for the implementation tools to follow while processing an FPGA design. These modules overall resulted in the implementation of a stopwatch.

Throughout this lab, we ran into several problems while trying to implement the different modules. For instance, no digits were displaying onto the FPGA board initially, which was eventually solved by fixing clock methodology to get the correct frequency.

Initially, we thought was two times too fast, and setting initial values to zero also helped. Another example of an issue we ran into was that the adjust logic was wrong because we didn’t read the specification thoroughly, however we were able to change the logic to reflect the spec by reading it closely and having the details clarified. Furthermore, the adjust would set the selected bit to one and all of the other bits to zero. We were unable to get the digits from the counter module because we changed the implementation for testing, but we solved this issue by getting the output from counter to adjust mode.