Lab 3:
Stopwatch

Matthew Wong 704569674
Wenlong Xiong 204407085
Vincent Jin 604464719

CS M152A Winter 2017
Monday - Wednesday, 10 - 12

Introduction

For this lab, we designed and implemented a stopwatch circuit that runs on the Nexys3 board. The stopwatch displays minutes and seconds through the board's four-digit seven-segment display in the form MM:SS. Like a normal stopwatch, the digits start at 00:00, with the seconds digits incrementing at 1 Hz. The minutes digits will increment when the seconds go from 59 to 0. When the stopwatch reaches 59:59, the next increment will recycle the display to 00:00.

Additionally, the stopwatch has inputs from the buttons and slider switches on the board. The right button, btnR, signals a reset, causing the stopwatch to set its time value to 00:00. The center button, btnS, toggles a paused state, during which the stopwatch stops incrementing and freezes its time value. The rightmost switch, sw[0], sets the stopwatch into an adjustment state, in which the stopwatch will freeze either the minutes digits or the seconds digits. The unfrozen digits will increment at a faster rate of 2 Hz while blinking at a rate of 5 Hz. The adjacent switch, sw[1], determines whether minutes or seconds will be the values affected by the adjustment state.

Our implementation will be tested on Xilinx simulation and the Nexys3 board. If the test simulation for a module requires an extensive period of time and produces only limited results (for example, the 1 Hz clock divider), the testing will be performed directly on the Nexys3. The implementation will ultimately be run on the Nexys3 board. Simulation tests are listed below in the simulation documentation section.

Design Description

The top module, called "stopwatch", has inputs for the clock, two buttons, and two switches that are provided by the Nexys3 board. The clock has a frequency of 100 MHz and is divided by the clocks submodule into several other slower frequencies. The btnR button signal triggers a reset of the stopwatch, and the btnS button signal is sampled by the debouncer submodule to create a clean pause signal. The two switches control the adjustment state of the stopwatch. There are also outputs of an[3:0] and seg[7:0], which control the seven segment display on the board.The module itself contains four submodules and the connecting wires.

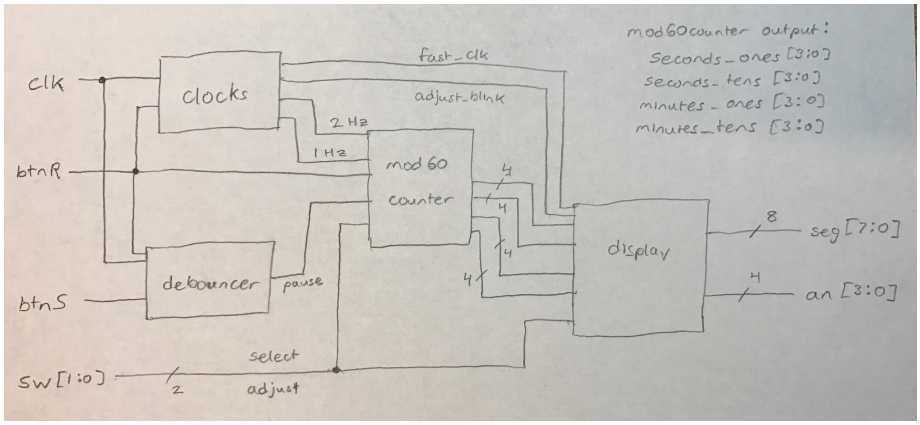

Figure 1: **High Level Design of Top "stopwatch" Module.**

The debouncer submodule is used to filter noise from the pause button whenever it is pressed. To achieve this, the debouncer contains its own clock divider that creates a 763 Hz timing signal. The debouncer samples the button input at this slower frequency to determine a valid button push. When there is a valid button push, the debouncer will briefly output a high signal that will be sent to the mod60counter submodule.

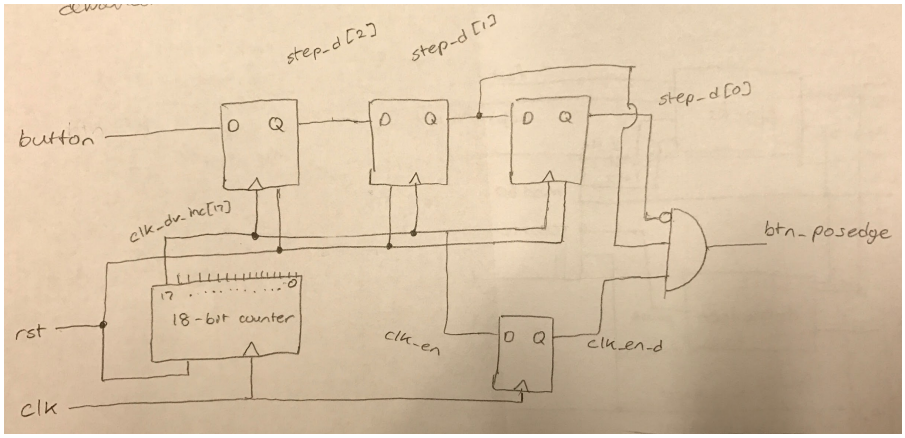

Figure 2: **High Level Design of "debouncer" Module.** The input from the button is sampled at a frequency of 763 Hz. The values are stored in stepping registers that are used to determine valid button presses.

The clocks submodule contains the four clock dividers that are used by other submodules in the stopwatch. A typical clock divider maintains an internal register counter that is compared to a constant. The divider flips its clock signal output every time the counter equals the constant, creating a clock with a frequency that is much slower than the input clock. In this submodule, the input clock comes from the Nexys3 board and has a frequency of 100 MHz. To create the 2 Hz clock signal, the clock divider uses a constant of 5 x 10^7. The 1 Hz clock uses a constant of 1 x 10^8, the 200 Hz clock (fast clock) uses a constant of 5 x 10^5, and the 5 Hz clock (blink clock) uses a constant of 2 x 10^7. The 1 Hz and 2 Hz signals are sent to the mod60counter submodule to control the time counters. The 200 Hz and 5 Hz signals are sent to the display submodule to control the cycling and blinking of the digits.

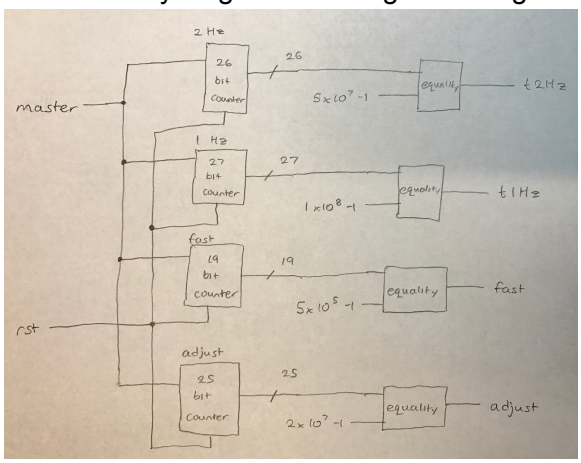

Figure 3: **High Level Design of "clocks" Module.** There are four counters that each correspond to a unique clock divider. The outputs are 2 Hz, 1 Hz, 200 Hz, and 5 Hz clock signals.

The display submodule is responsible for generating the output that controls the seven segment display on the Nexys3 board. The first output is anode[3:0], which controls which of the four digits of the display to light up. The other output is segment[7:0], which controls which of the seven segments to turn on to show the desired value. The anode output uses the 200 Hz clock to cycle through the values of 4'b1110, 4'b1101, 4'b1011, and 4'b0111. This causes the 4 different digits on the display to rapidly cycle through turning on and off, giving the illusion that all four digits are turned on. Depending on which digit is being shown, the submodule selects one of the 4-bit digit inputs to translate into an 8-bit seven segment representation.

Also, when the stopwatch is in the adjustment state, the display will blink either the seconds digits or the minutes digits at a rate of 5 Hz. This blinking is caused by setting segment to not display a number when anode is at the appropriate digits and the blink_flag is set. If the select input is 0, the seconds digits will blink, and if the select input is 1, the minutes digits will blink.

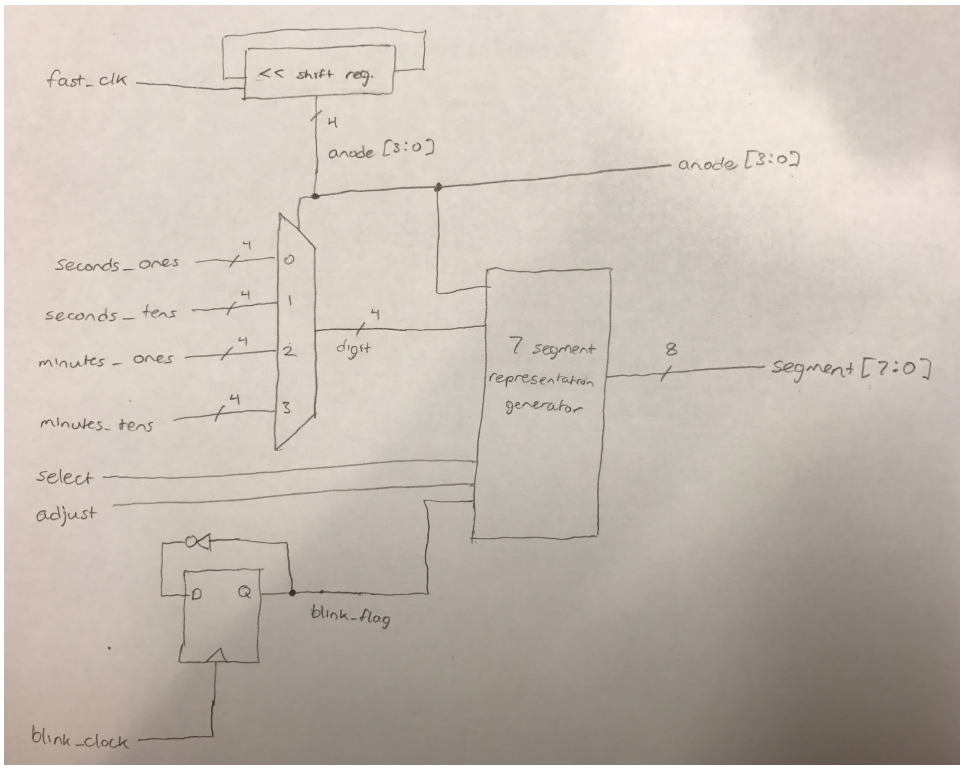

Figure 4: **High Level Design of "display" Module.** The anode[3:0] output uses the 200 Hz clock to quickly cycle through each of the digits. The anode[3:0] determines which of the 4-bit digit inputs to translate into a 8-bit seven segment output.

The mod60counter submodule is responsible for controlling the time values in the stopwatch. There is a mod-60 counter for the seconds and a mod-60 counter for the minutes. These two counters will increment based on the clock signal that is output by the select_clock submodule. Also, the two counters will output 4-bit values for each of the 4 digits of the stopwatch. The rst input signal will reset both counters to 0, causing the stopwatch to have a time of 00:00.

During a normal counting state, the seconds counter increments on the 1 Hz clock signal and will increment the minutes counter when the seconds overflow from 59 to 0. When adjust =

1, the stopwatch is in the adjustment state and will increment either the seconds counter or the minutes counter on the 2 Hz clock signal. If select = 0, the seconds will increment at 2 Hz while the minutes will be frozen. If select = 1, the minutes will increment at 2 Hz while the seconds will be frozen. Also, there is a pause signal from the debouncer submodule that toggles the paused state of the stopwatch. When the stopwatch is paused, neither counter will increment regardless of the adjustment state.

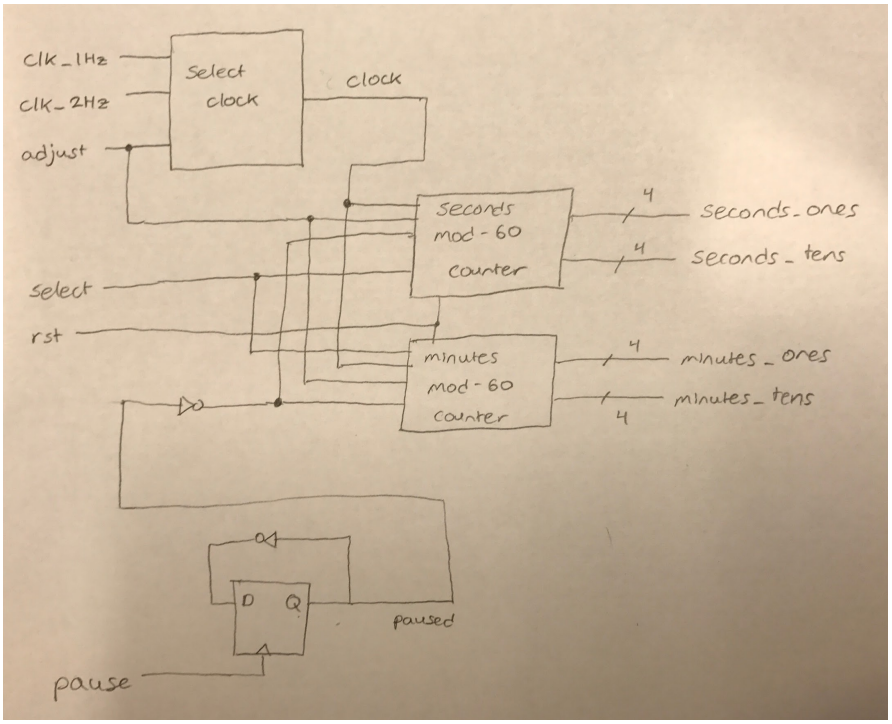

Figure 5: **High Level Design of "mod60counter" Module.** The select_clock submodule determines the clock signal that the two counters will use for their incrementation. The two counters will output the digits of the time as 4-bit numbers. Also, there is a register to store the paused state of the stopwatch, during which the counters will stop.

The select_clock submodule helps the mod60counter determine which clock signal it should use to increment its counters. There are inputs for the 1 Hz clock, the 2 Hz clock, and the adjust signal. If adjust = 0, the stopwatch should count normally on the 1 Hz clock. If adjust = 1, the stopwatch should be adjusting and would count faster on the 2 Hz clock.

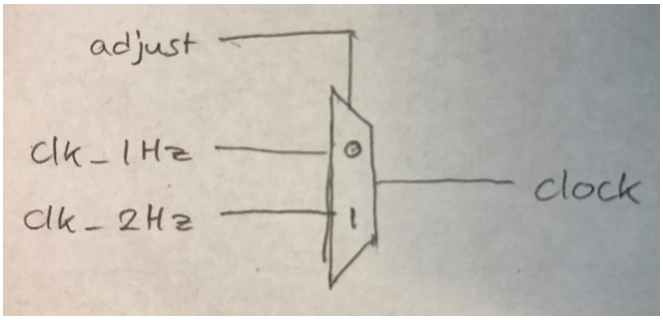

Figure 6: **Gate-Level Design of "select_clock" Module.**

Simulation Documentation

select_clock module:

Test Case	Result	Pass/No Pass
adjust = 0	clock output should be the same as the clk_1Hz input	Passed
adjust = 1	clock output should be the same as the clk_2Hz input	Passed

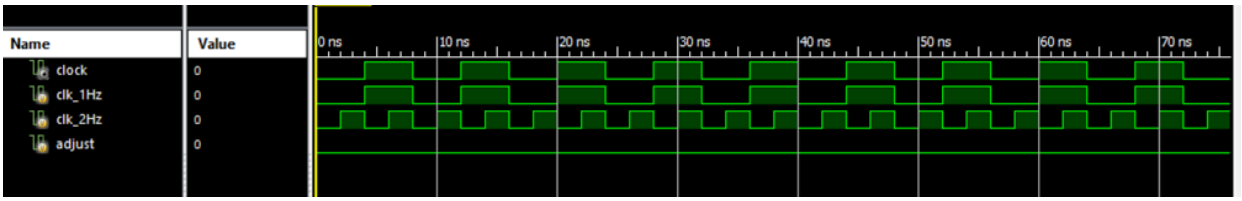

Figure 7: **Waveform of "select_clock" Module for adjust = 0.** If adjust = 0, the clock output is directly linked to the input of clk_1Hz, signaling the "mod60counter" module to count normally at 1 Hz.

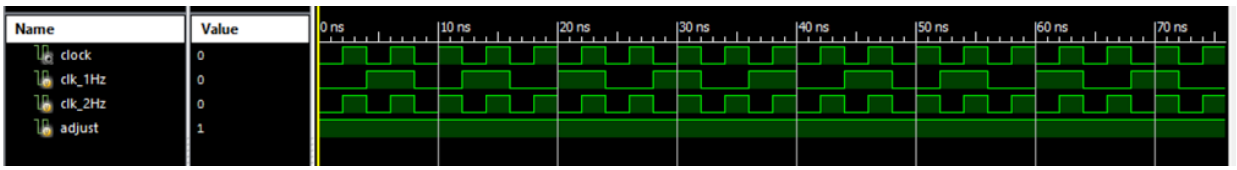

Figure 8: **Waveform of "select_clock" Module for adjust = 1.** If adjust = 1, the clock output is directly linked to the input of clk_2Hz, signaling the "mod60counter" module to count faster at 2 Hz.

mod60counter module:

Test Case	Result	Pass/No Pass
rst = 1	seconds_ones, seconds_tens, minutes_ones, and minutes_tens should all remain at 4'b0000	Passed
select = 0 or 1, adjust = 0	Normal counting	Passed
select = 0, adjust = 1 (adjust the seconds)	seconds should increment on the clk_2Hz signal, while minutes should hold steady	Passed
select = 1, adjust = 1 (adjust the minutes)	minutes should increment on the clk_2Hz signal, while seconds should hold steady	Passed
paused = 1	No counting. All four digit outputs should remain at their values	Passed

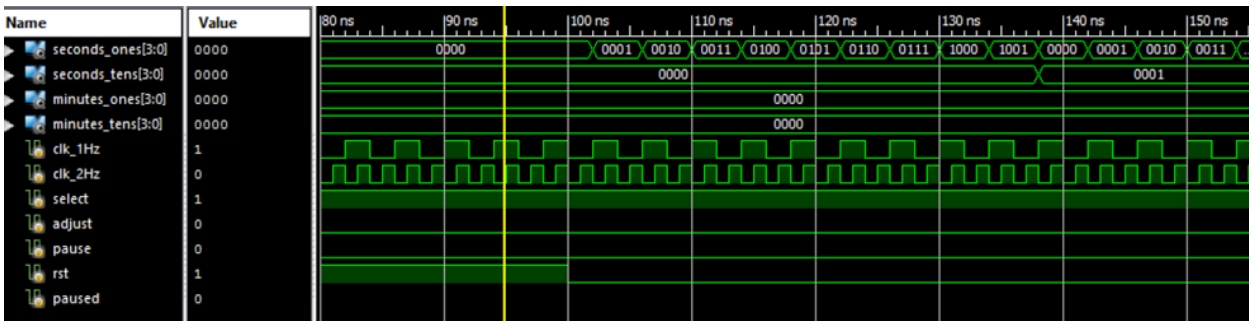

Figure 9: **Waveform of "mod60counter" Module for rst = 1 and Normal Counting.** When rst = 1, the registers for the digits are all set to 0. Otherwise, since adjust = 0 and the module is not 'paused', normal counting occurs.

Figure 10: **Waveform of "mod60counter" Module for Adjusting Seconds.** Since adjust = 1 and select = 0, the seconds digits increment on the 2 Hz clock and the minutes digits are frozen.

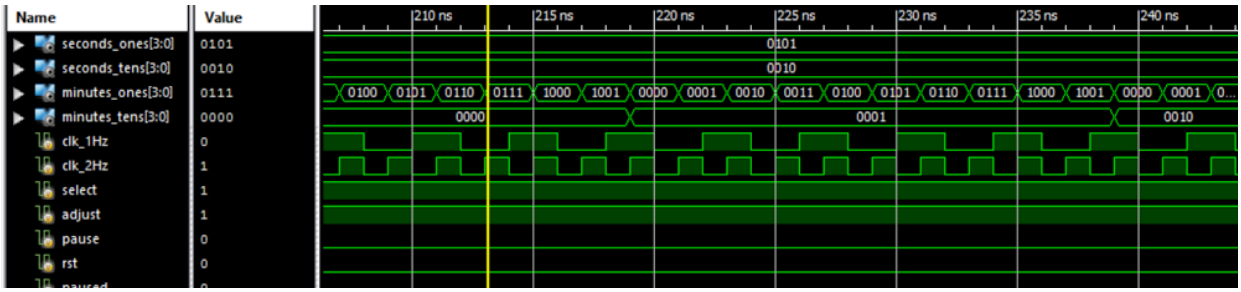

Figure 11: **Waveform of "mod60counter" Module for Adjusting Minutes.** Since adjust = 1 and select = 1, the minutes digits increment on the 2 Hz clock and the seconds digits are frozen.

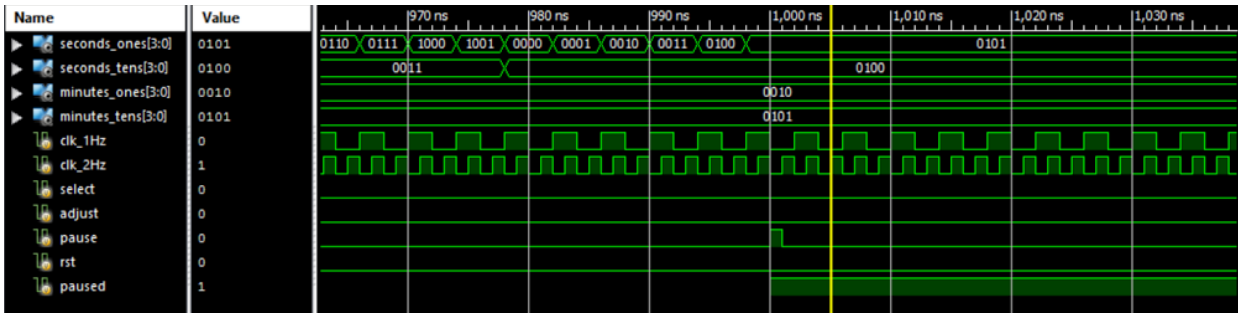

Figure 12: **Waveform of "mod60counter" Module for Paused State.** When in a 'paused' state, which is triggered by the positive edge of the 'pause' input, none of the four digits will increment.

clocks module:

Test Case	Result	Pass/No Pass
rst = 1	Clock dividers do not count and remain at a value of 0	Passed
rst = 0	Clock dividers count normally	Passed

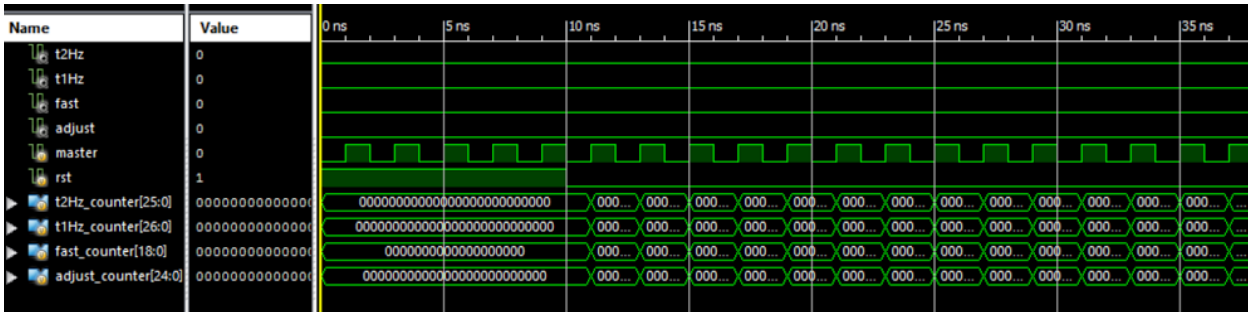

Figure 13: **Waveform of "clocks" Module when rst = 1.** While rst = 1, the counters for the clock dividers will be set equal to 0 and will not increment. Otherwise, normal counting occurs at the positive edge of the master clock.

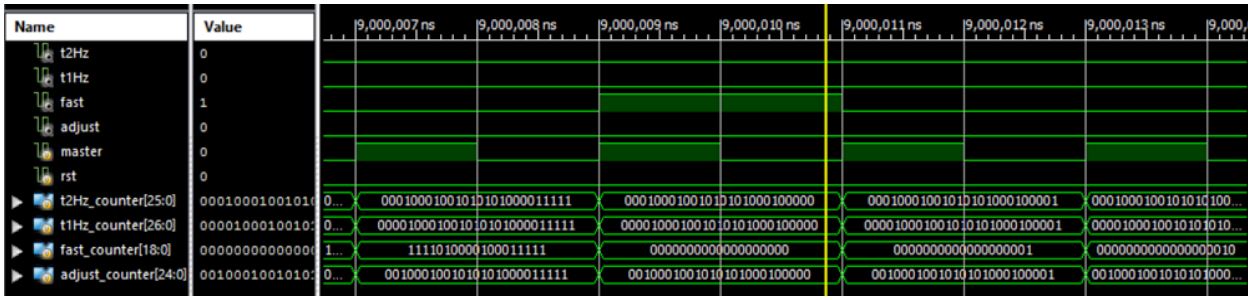

Figure 14: **Waveform of "clocks" Module during Normal Counting.** The counter for the 'fast clock' of 200 Hz has counted to the appropriate value, setting the 'fast' clock output to 1 and resetting the counter to 0.

display module:

Test Case	Result	Pass/No Pass
select = X, adjust = 0, seconds_ones = 1, seconds_tens = 2, minutes_ones = 3, minutes_tens = 4	Normal display state: anode should cycle between 4'b1110, 4'b1101, 4'b1011, and 4'b0111, and segment should cycle between the 7-segment representations of the digit inputs of 1, 2, 3, and 4	Passed
select = 0, adjust = 1, seconds_ones = 1, seconds_tens = 2, minutes_ones = 3, minutes_tens = 4	Blinking the Seconds Digits: anode should continue its normal cycle, but segment will not show the seconds digits if blink_flag is 0. Not showing a digit is equivalent to segment = 8'b11111111	Passed
select = 1, adjust = 1, seconds_ones = 1,	Blinking the Minutes Digits: anode should continue its normal cycle, but segment	Passed

seconds_tens = 2, minutes_ones = 3, minutes_tens = 4	will not show the minutes digits if blink_flag is 0. Not showing a digit is equivalent to segment = 8'b11111111	

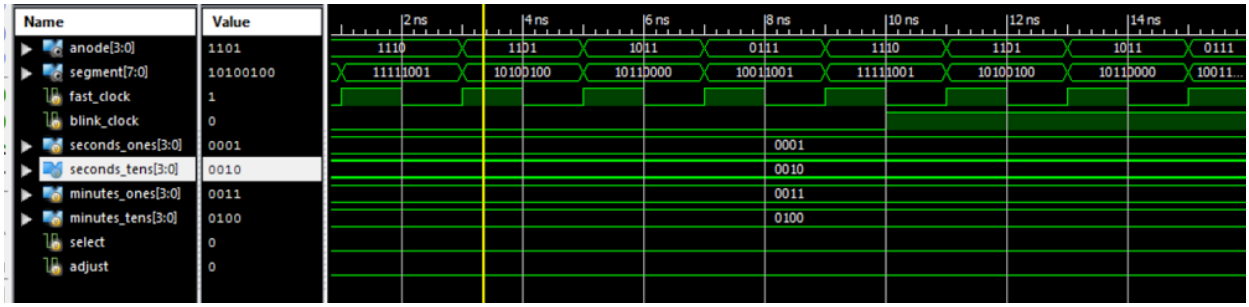

Figure 15: **Waveform of "display" Module during Normal Display State.** During normal display state, the fast_clock of 200 Hz will cycle the anode and segment outputs to correctly show the four digits on the LED display.

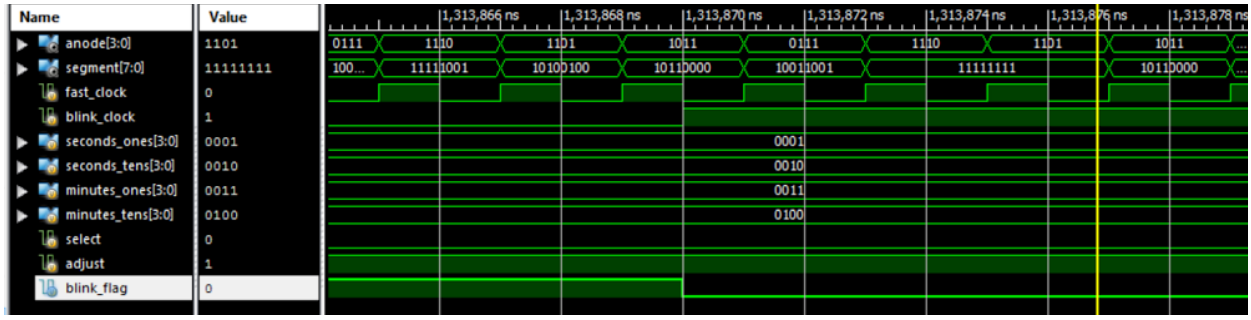

Figure 16: **Waveform of "display" Module Blinking Seconds Digits.** When adjusting the seconds, the fast_clock will continue to cycle the anode and segment outputs. However, the blink_clock of 5 Hz determines whether the seconds digits should be shown at certain cycles, giving a blinking effect.

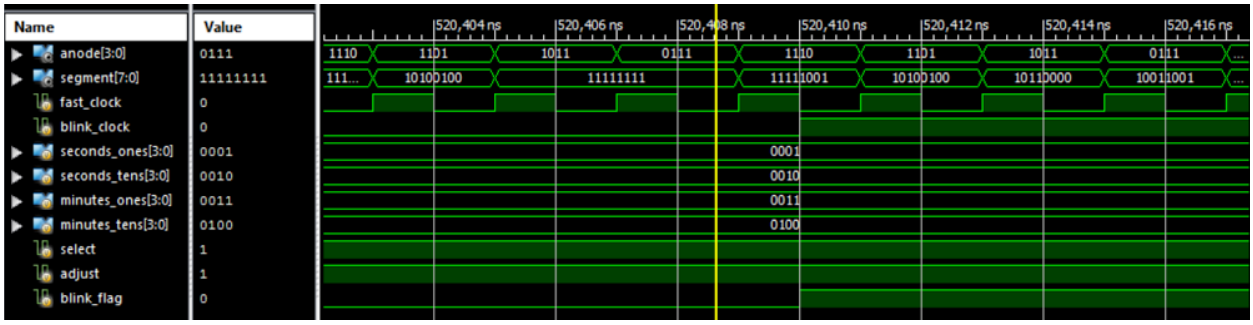

Figure 17: **Waveform of "display" Module Blinking Minutes Digits.** Similar to adjusting seconds, the minutes digits will not be shown at every cycle, giving the same blinking effect.

debouncer module:

Test Case	Result	Pass/No Pass
Push FPGA button connected to debouncer	debouncer module should send a signal to the mod60counter module to pause counting, which should be seen when the	Passed

	display module shows the same 4 digits	

stopwatch module:

Test Case	Result	Pass/No Pass
state = normal	Display should increment at 1 Hz and should recycle to 00:00 after 59:59	Passed
state = paused	Display should stop incrementing until unpaused	Passed
state = adjust seconds	The seconds digits should increment at 2 Hz and should blink at a rate of 5 Hz. The minutes digits should be frozen.	Passed
state = adjust minutes	The minutes digits should increment at 2 Hz and should blink at a rate of 5 Hz. The seconds digits should be frozen.	Passed

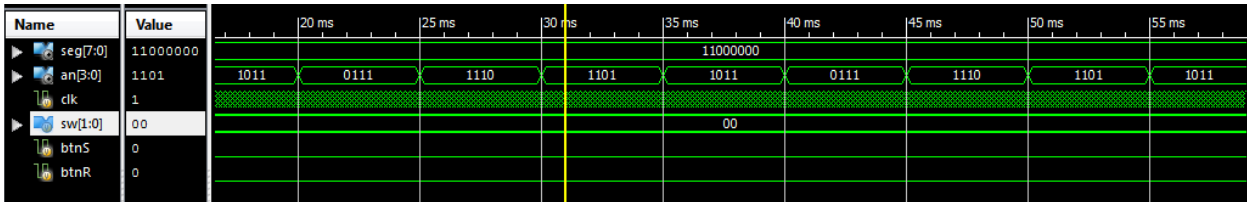

Figure 18: **Waveform of "stopwatch" Module during Normal State.** The 'clk' at 100 MHz is divided by the fast clock which triggers the cycling of an[3:0]. The seg[7:0] remains at the 7-segment representation for the number 0 because the first change that could be seen occurs after an extensive time period when the seconds digit increments at 1 Hz.

Conclusion

The stopwatch circuit we have implemented uses the seven segment display of a Nexys3 board to create a basic clock that counts seconds and minutes. There are four main submodules within an encapsulating top module. The debouncer submodule is responsible for eliminating the noise from button presses on the FPGA board. The clocks submodule contains clock dividers that create slower clock frequencies for the other submodules to use. The mod60counter submodule controls the time within the stopwatch. The display submodule converts the time into output that can be shown by the board's seven segment display.

One of the main difficulties we encountered was creating the board implementation from our code. Our first drafts of our code for the submodules passed our test cases in Xilinx simulation, but when implemented and programmed to the board, the code produced unexpected results. For example, the display submodule would show that it successfully cycles its anode[3:0] output in simulation, but the board would only show 2 of the 4 digits when programmed. We have two possible reasons for why these errors occurred. First, it is possible that the Xilinx compiler made errors in creating the implementation. Or, it is possible that the correct implementation violated various time constraints within the circuit. Either way, we rewrote our code while keeping most of the same logic and reducing the number of nested if-statements within submodules. Eventually, we were able to produce code that successfully passed Xilinx simulation and achieved the expected results on the Nexys3 board.