

CS M152A: Introductory Digital Design

Laboratory

**Lab #3**

**Stopwatch**

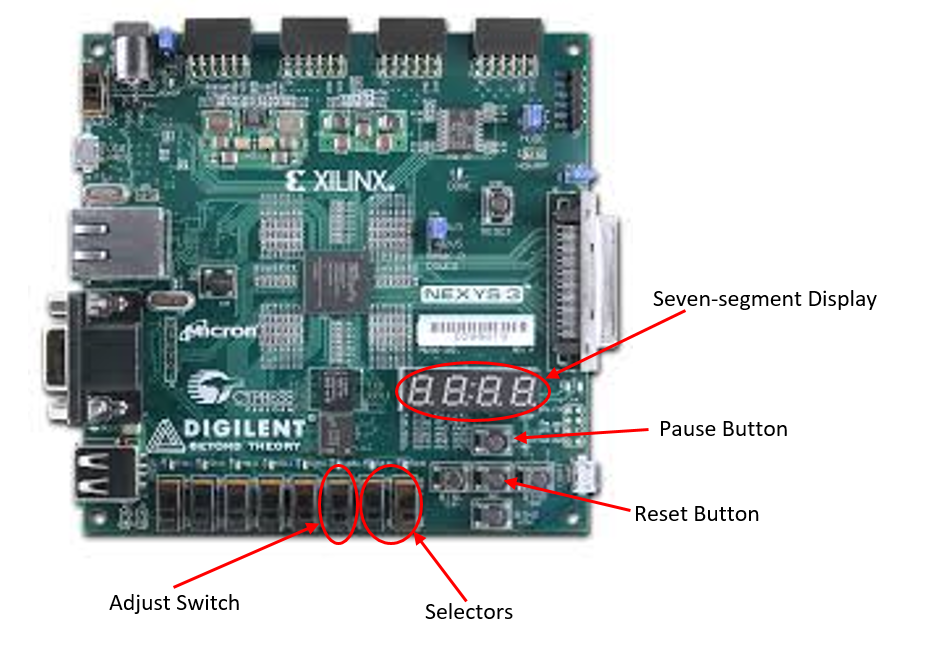
Name: Kenny Chan, Kenny Luu, Kyle Reidy

UID: 004769092, 104823244, 104839297

Lab Section: 5

Due Date: May 24, 2018

**Introduction**

The goal of this lab was to design a basic stopwatch on the Nexys™ 3 Spartan-6 FPGA Board, using the knowledge and background we learned in previous labs. The stopwatch should behave like a digital clock and display minutes and seconds as four digits on the seven-segment display.

Functionalities of the stopwatch include the ability to pause, reset, and set the clock.

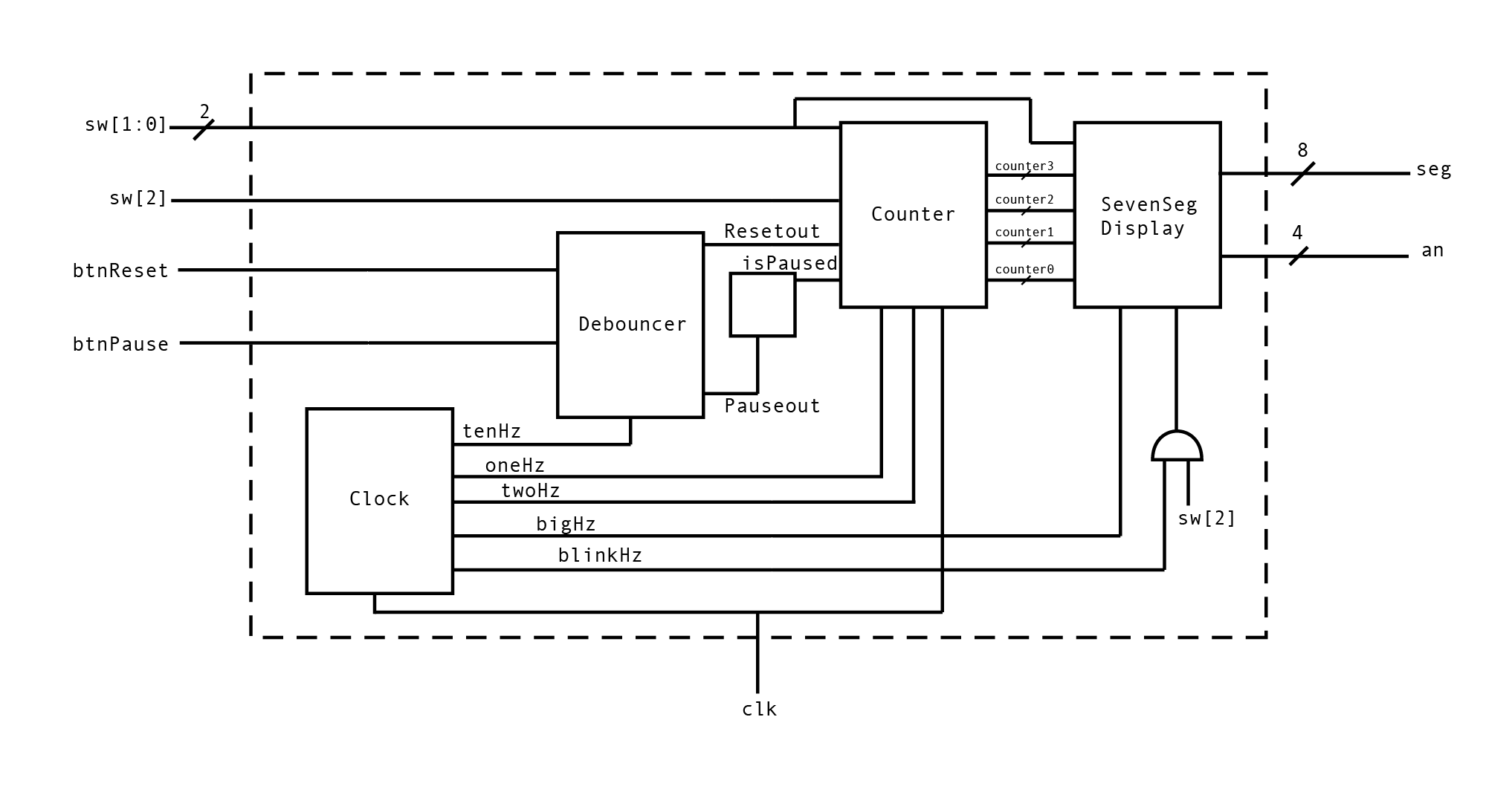
To pause the time, we used the top button on the FPGA board. To reset the clock to the initial state of 00:00, we used the middle button. Setting the clock was implemented using the switches on the FPGA board. To set the clock, one switch was used to turn on the adjustment mode. During the adjustment mode, two other switches were used as selectors for which digit of the display to adjust. The digit selected by the selector is indicated by the following table.

**Selector Correspondence**

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

**Design Description**

For our design, we used one main stopwatch top module that utilizes four submodules: Clock, Counter, SevenSegDisplay, and Debouncer.

**

**Clock**

The purpose of the clock module is to create several clocks with different frequencies using the master clock from the FPGA board, which runs at 100MHz. Specifically, we created a 1 Hz clock to update the time of our stopwatch every second. We used a 2 Hz clock for the rate at which the selected digit will increment during the adjustment mode. We used 100 Hz clock, labelled “bigHz”, for the seven-segment display to seemingly represent four digits at once, while it actually displays each digit at a rate faster than we can process. Another clock, “blinkHz”, ticks at a rate of 8 Hz, which is the rate the selected digit will blink during adjustment mode. Finally, we utilized a 10 Hz clock for the debouncer.

To create these clocks with lower frequencies, we increment unique counters for each of the desired clocks at the posedge of the master clock. Once the counter reaches a certain value, we invert the clock and reset the unique counter. The value can calculated using the formula:

, X = desired clock rate, V = value at which to reset the counter

For example, consider the one Hz clock. We want to invert the oneHz clock twice every posedge of the 100Mhz master clock. There are 100,000,000 Hz in 100Mhz and we want to invert the oneHz clock twice to complete a cycle. Therefore we will reset the unique counter at 50,000,000, and invert the oneHz clock. This process is repeated for each of the other clocks.

**Counter**

The counter module takes in multiple clock signals, which are the 10Hz, 1Hz, and 2Hz clock signals generated by the clock divider module. We also pass in signals to detect whether the stopwatch is paused, being reset, or being adjusted. The output signals are the values of the four internal counters, which correlate to the four seven segment display numbers.

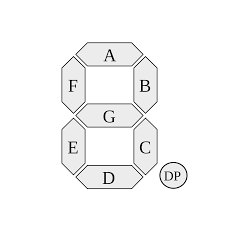
Inside the clock module, we use the 10Hz signal to “refresh” and check all of its inputs. To check for rising edges of the 1Hz and 2Hz clocks, the counter module internally buffers the values of the 1Hz and 2Hz clock inputs taken at the previous three 10Hz rising edges. When the is buffer holds a 0 bit followed by a 1 bit, this indicates a rising edge (this is a similar implementation to the Debouncer module, which will follow).

If a rising edge from the 1Hz clock signal is detected, and the stopwatch is not paused or being adjusted, then we increment the lowest counter by one. Once it reaches 9, it resets to 0 and add one to the next most significant counter. This pattern continues and flows into the most significant counter. If a rising edge to the 2Hz clock signal is detected, and the stopwatch is not paused but is being adjusted, then it will increment only one of the counters based on the value of the “select” switches. This counter will not be allowed to overflow into the next until it is no longer being adjusted. Upon detecting a reset signal the module resets all counters back to 0. Upon detecting a pause signal the module will not increment any of the counters until the pause returns to low.

**SevenSegDisplay**

The SevenSegDisplay module’s input are the counter values of each digit and its task is to update the FPGA’s seven segment display accordingly. Using the bigHz clock, this module will handle the display for each digit one at a time. We used a counter up to four to indicate which the module should display each clock cycle of bigHz.

To display a digit, the module adjusts seg[7:0] and an[3:0] of the UCF file. According to the Nexys3 reference manual, each bit of seg[7:0] selects which a segment of the display light up. The following figure and table indicate which portion of the segment each bit corresponds to.



|  |  |
| --- | --- |
| **Bit** | **Segment** |
| seg[7] | DP |
| seg[6] | G |
| seg[5] | F |
| seg[4] | E |
| seg[3] | D |
| seg[2] | C |
| seg[1] | B |
| seg[0] | A |

The an[3:0] bits indicate which of the four digits to display. The least significant bit corresponds to the rightmost bit. An important note about seg[7:0] is that the bit 0 actually indicates the segment to light up, while 1 will turn off the light. Similarly, 0 selects which of the four digits to light up. For example to display eight as the fourth digit, an[3:0] would be set to 4b1110 and seg[7:0] would be 8b10000000. For simplicity, we used a function that converts digits to the appropriate bits for seg[7:0].

If the adjustment mode switch was set, the selected digit should blink instead. To do so, we only displayed that digit when blinkHz was high.

**Debouncer**

The debouncing module takes in a 10Hz clock signal, along with the state of pause and reset buttons. The debouncer works using the same method as lab 2’s debouncer. At every rising edge of the 10Hz signal, it stores the state of the buttons in their own 3 bit register concatenated with the state of the previous 2 cycles. It determines whether a rising edge has occurred by checking for a transition from 0 to 1. The module then outputs a signal that represents the rising edge of the buttons, it is driven to 1 when it sees a rising edge and is pulled down to 0 otherwise. For the reset signal, the output is passed directly to the Counter module. The pause signal, on the other hand, is used by the main module to flip the value of a 1-bit register that indicates whether or not the stopwatch is paused, and it is the value in this register that is passed to the Counter.

**Simulation Test Benches**

**Clock**

To test the Clock Module, we set the input to alternate between 1 and 0 very rapidly, which simulates a clock signal. After that, we looked at the waveforms to check whether our clock signals were alternating appropriately. For example our bigHz (500Hz) output would alternate once every couple thousand of normal clock signal rising edges. Our 10Hz output should alternate once every couple hundred of bigHz rising edges, our 1Hz output should alternate once every two rising clock edges of our 2Hz output, etc.

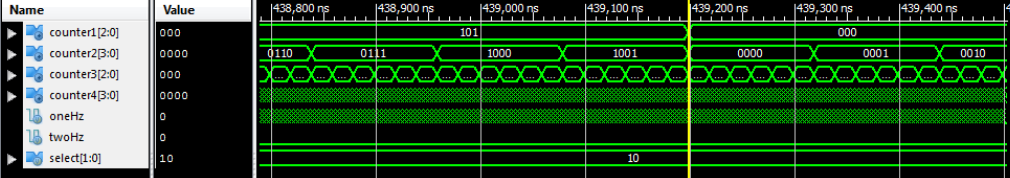
|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| while (1) begin  clk = ~clk;  #1;  end | Output clocks were output at the right frequency based on simulation.  Successful demo further demonstrates it’s accuracy. | Pass |

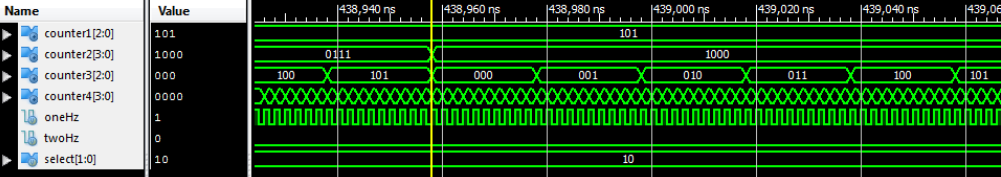
**Counter**

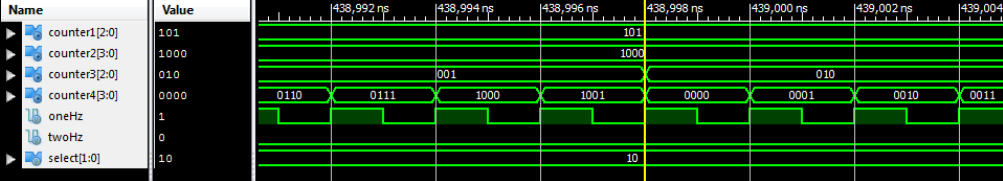
To test the Counter Module, we repeated the the setup with the Clock module testbench where we alternated our oneHz and twoHz inputs repeatedly. Along with that, we chose our reset, pause, and adj signals based on the state that we were testing. We checked the waveforms to see whether the counters would perform the right behavior.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| Only oneHz alternating;  Adj = 0;  Select = XX;  Reset = 0;  Pause = 0; | Counters incremented correctly and overflowed into the next most significant counter appropriately.  (Figures 1, 2, and 3) | Pass |
| Only twoHz alternating;  Adj = 1;  Select = 00, 01, 10, or 11;  Reset = 0;  Pause = 0; | Counter that was chosen by the select input was incremented on the twoHz posedge and did not overflow into other counter.  (Figure 4) | Pass |
| Both oneHz and twoHz alternating;  Adj = 0;  Select = XX;  Reset = 0 to 1 transition;  Pause = 0; | Counters were incrementing correctly and overflowing correctly.  Counters were reset to 0 after 0 to 1 transition. | Pass |
| Both oneHz and twoHz alternating;  Adj = 0;  Select = XX;  Reset = 0;  Pause = 0 to 1 to 0 transition; | Counters were incrementing correctly and overflowing correctly.  Counters stopped incrementing while Pause was 1. | Pass |
| Both oneHz and twoHz alternating;  Adj = 1;  Select = 00, 01, 10, or 11;  Reset = 0;  Pause = 0 to 1 to 0 transition; | Counter chosen by select was incrementing correctly and not overflowing.  Incrementing stopped while Pause was 1. | Pass |

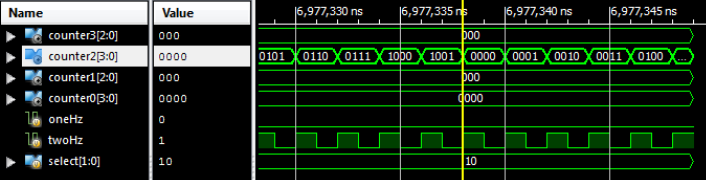
**Figure 1, 2, and 3**







**Figure 4**

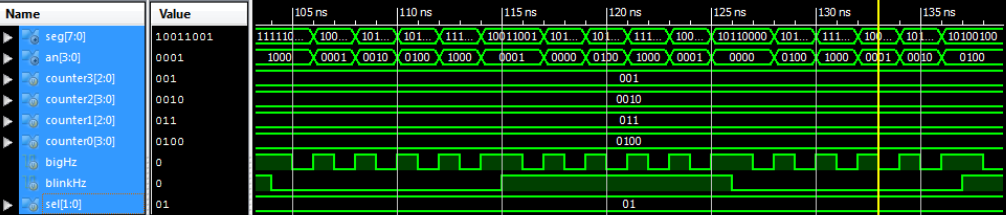


**Seven Segment Display**

To test the seven segment display, we used the same set up by alternating the bigHz signal more frequently than the blinkHz signal. Even though blinking is 8Hz while bigHz is 500Hz, we alternated them at a different scale for simplification. We also set the counter inputs to a static value to see whether the seg[7:0] output matches these static values appropriately. To test the blinking of the number when adj is turned on, we set our “select” input to be values from 0-3 to make sure the appropriate seven segment display blinks.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Result** | **Pass/No Pass** |
| Alternate only bigHz;  counter0 = 4;  counter1 = 3;  counter2 = 2;  counter3 = 1;  sel =XX; | The an[3:0] output successfully alternated sequentially from 1110, 1101, 1011, and 0111;  The seg[7:0] output successfully matched the counter value for the appropriate seven segment display;  (Figure 5) | Pass |
| Alternate bigHz and blinkHz;  counter0 = 4;  counter1 = 3;  counter2 = 2;  counter3 = 1;  sel = 00, 01, 10, or 11; | The an[3:0] output successfully alternated sequentially from 1110, 1101, 1011, and 0111;  The seg[7:0] output successfully matched the counter value for the appropriate seven segment display;  The value of seg[7:0] would become all 1’s (off) if blinkHz was 0 and turn back on if blinkHz was 1; | Pass |

**Figure 5**



**Debouncer**

Debouncer was tested by uploading to the board and testing button response.

**Conclusion**

Overall, we were able to successfully implement the stopwatch module onto a working FPGA as opposed to pure simulation. By implementing the stopwatch one module at a time and verifying the behavior with simulation, we eventually programmed the FPGA with a fully working and functioning stopwatch implementation. This includes the ability to count time by the second, display the result on a seven segment display, reset values, pause, and increment selected parts of the stopwatch.

We encountered most of our difficulty when implementing the button debouncer. This was because the debouncer and counter module had to be synchronized in order to successfully detect whenever the debouncer detected a rising edge. At the same time, this module couldn’t be debugged with iSim since it had to be uploaded onto the board to debounce its hardware buttons. This area of difficulty became our most time consuming issue, but we were eventually able to work through the difficulty and debounce our buttons.