Xuan Yuan Huang

Alvin Lim

Group 11

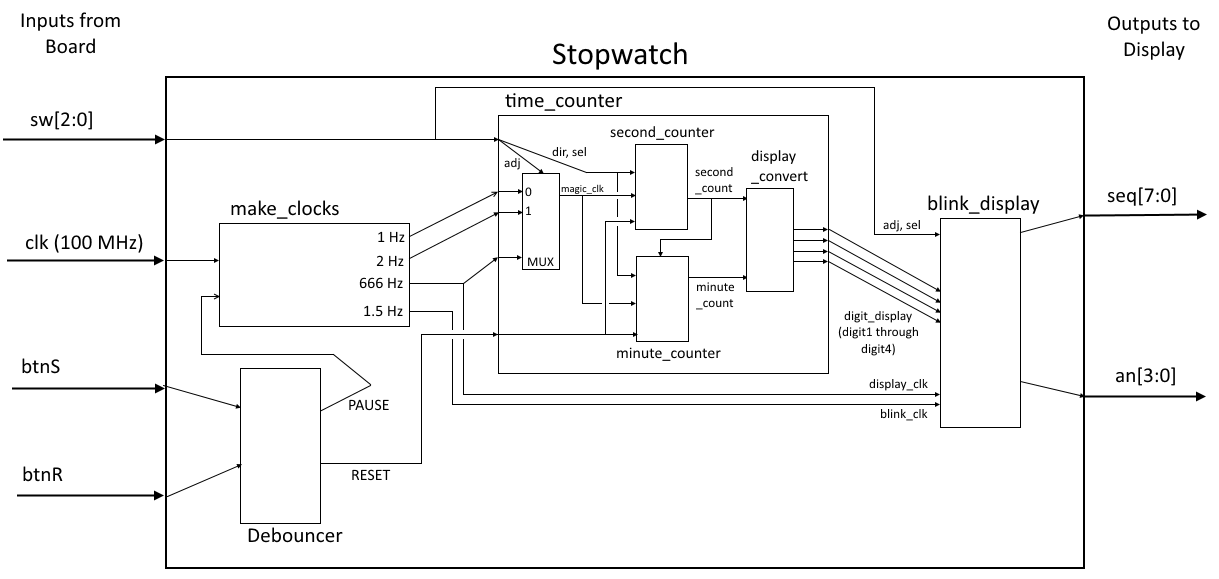
CS M152A

Lab 3

**Introduction**

In this lab, we designed a stopwatch circuit and implemented it on the Nexys 3 Spartan-6 FPGA board. Our stopwatch design has rudimentary functionality: (a) counting up and down, (b) pausing, (c) resetting, (d) time adjustment, and (e) blinking the display during adjustment. Input for our implementation used buttons and slider switches on the FPGA board, while output was displayed on the four on-board 7-segment displays. Users are able to count up or down with a DIR slider switch; adjust either minutes or seconds at 2Hz using ADJ and SEL slider switches; and reset the clock instantly with a button no matter the mode; and pause at any time with a button.

**Design**



Our design for Stopwatch had three primary modules:

* **make\_clocks**: Divides up the 100 MHz onboard clock
  + ***Input:*** 100 MHz clock, PAUSE
  + ***Output:*** 1 Hz, 2 Hz, 666 Hz, and 1.5 Hz clocks
  + ***Implementation:***
    - For each clock, we used a counter that counted up to an appropriate period at the rate of 100 MHz clock (e.g. for the 1 Hz clock, count up to 50,000,000, then invert the output, meaning 1 posedge every second)
    - PAUSE prevents the 1 Hz and 2 Hz counters from incrementing, effectively pausing time for time\_counter
* **time\_counter:** Calculates time
  + ***Input:***sw[2:0] (adj, dir, sel), 1 Hz, 2 Hz, 666 Hz, RESET
  + ***Output:*** digit1\_display through digit4\_display
  + ***Implementation:***
    - A multiplexer, using adj as select input, decides which clock to feed to the two counters
    - If adj is on, either minutes or seconds count at 2 Hz based on sel
    - If adj is off, seconds counts at 1 Hz, overflowing into minutes if necessary
    - Dir tells counters whether to increment or decrement
    - display\_convert takes the current seconds and minutes and converts them into the appropriate bit patterns for blink\_display
* **blink\_display:** Handles all LED display for the stopwatch
  + ***Input:*** digit1\_display through digit4\_display, adj, sel, display\_clk, blink\_clk
  + ***Output:*** seq[7:0], an[3:0]
  + ***Implementation:***
    - Using display\_clk (666 Hz), cycle through each digit of the seven-segment display andset seq[7:0] to the appropriate digit\_display bit pattern
    - If adj is on, then blink the appropriate digits (as determined by sel) at the rate provided by blink\_clk (1.5 Hz)

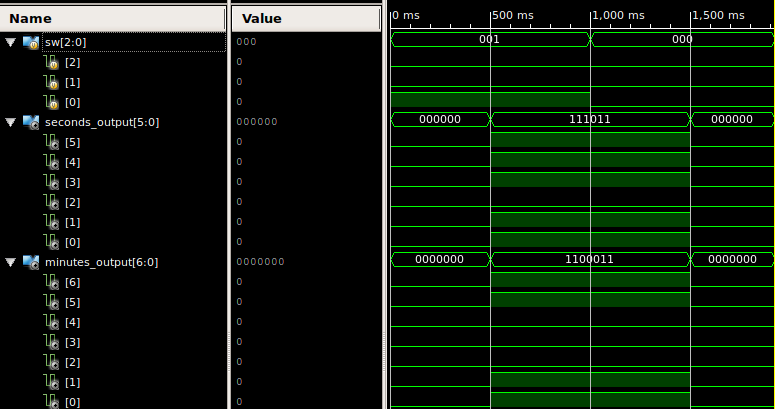
**Simulation**

We did a few simulations to test the functionality of our stopwatch. First, we show our simulation test of our “pause” implementation, which stops the clocks of the stopwatch from counting and allows it to resume counting at any time, even in the middle of a second. Our “pause” implementation takes the button btnS as input, which one can see is pushed at 1.6 seconds, then again later. Notice how after the first time btnS is pushed, our 1 Hz clock, used for regular counting, is frozen at 1 until the next time btnS is pushed. The same is true of our 2 Hz clock (frozen at 0), which is used for counting twice as fast when adjusting the time.

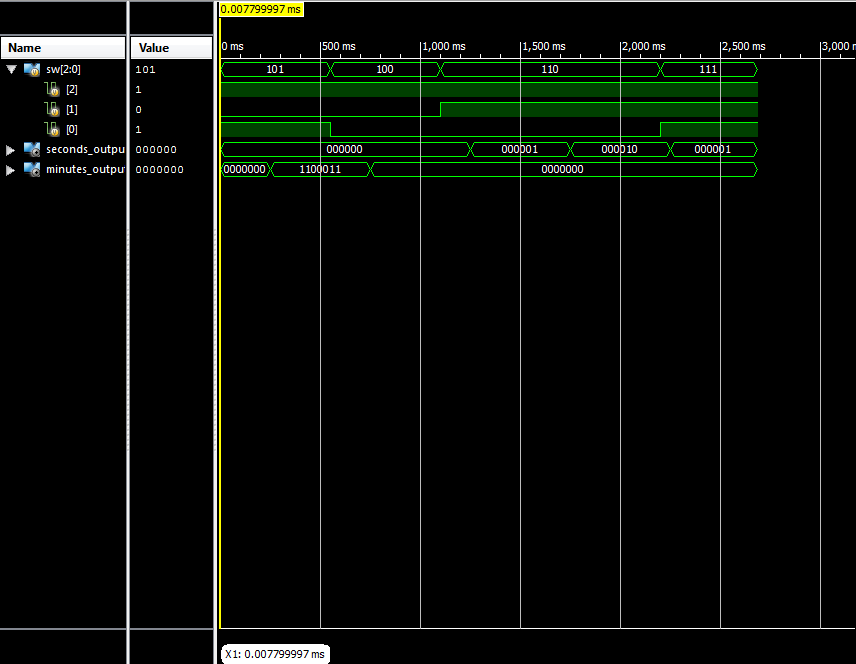


Also shown in these waveforms is our clock counters, which takes the main hardware clock that operates at 1MHz in order to implement our 4 clocks of 1 Hz, 1.5 Hz, 2 Hz, and 666Hz.

Our next simulation test shows our implementation of second and minute counters, as well as the functionality of our counter carry and overflow, both when counting up and when counting down.

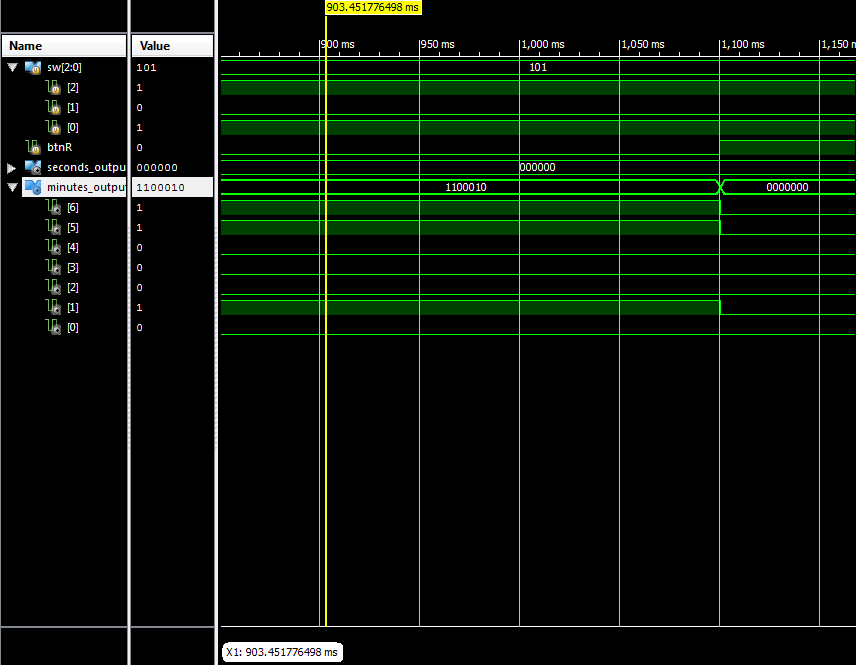
Note that sw[0] determines whether time is counted up (when it is 0) or counted down (when it is 1). When the counters start counting down, the counter for minutes changes from 7'b0 (0 seconds) to 7'b1100011 (99 seconds) and the counter for seconds changes from 6'b0 (0 seconds) to 6'b111011 (59 seconds). This shows that when our second counter is at 0 and the stopwatch counts down, the minutes also go down (in this case, wraps around). Similarly, after the sw[0] switch is flipped to 0, telling the stopwatch to count up, it is shown that our seconds counter, at 59 seconds, increments the minutes (again, wrapping up to 0) when going up to 0 seconds.

The next waveforms show the implementation of time adjustment in our stopwatch.



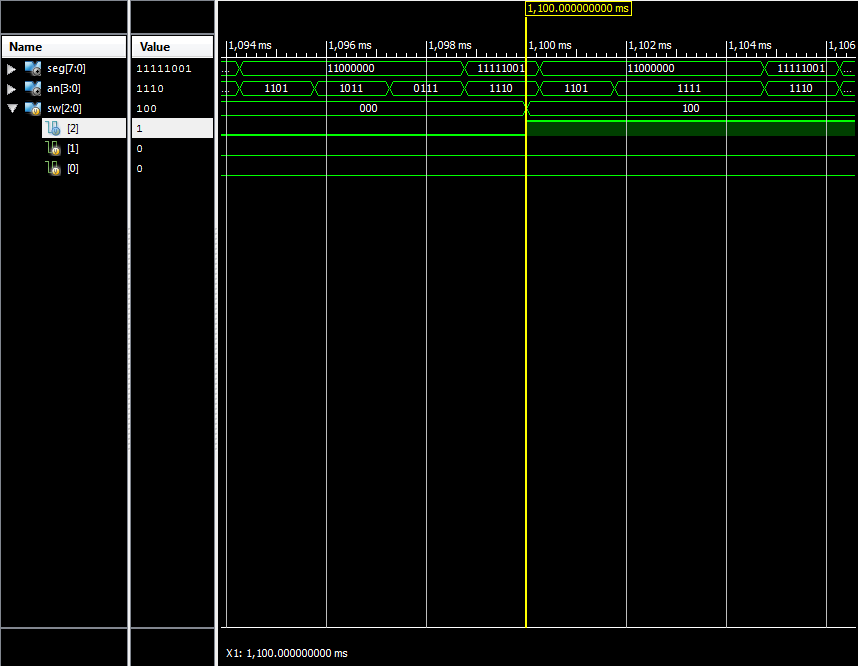
One can see that when the input sw is 101, the stopwatch is adjusting the minutes, at twice the speed as normal. When sw is changed to 100, the direction that the minutes adjusts is reversed. After, sw becomes 110 and the seconds become what the stopwatch adjusts. When sw is 111, we change the direction that the seconds are changed.

We also performed a simulation test to observe our implementation of resetting our stopwatch counter, shown here:



It is apparent that when when our input for “reset”, btnR, is pushed, the counter for minutes resets itself, going back to 0.

Finally, we show our simulation test that shows our implementation of our stopwatch blinking when adjusting the time.

Before choosing to adjust the time of the stopwatch, it is apparent that an is cycling through each one of the 7-segment displays, changing its output among 1110, 1101, 1011, and 0111, where 0 is the low-bit one-hot input specifying which one of the displays to output on the display. Once sw is switched to 100, we are choosing to adjust the minutes of the stopwatch. One can see that the an output starts cycling the numbers 1101, 1110, and 1111, showing that it outputs the seconds digits for the stopwatch, but does not output anything (1111) when it normally outputs the minutes digits. Our stopwatch switches between this implementation and the normal implementation at a 1.5 Hz rate, so every 1.5 seconds, minutes display does not display anything for 1.5 seconds, resulting in a blinking appearance.

*Note:* In our simulations, the seconds display of our stopwatch starts counting at 0.5 second rather than 1 second and starts adjusting at 0.25 seconds rather than 0.5 seconds. This is because simulation of the stopwatch takes a long time, so to speed it up for simulation purposes, we had our counters going off the positive edge of our clocks rather than our negative edges (as normal), so all our simulations have a phase offset of half a period.

**Conclusion**

This lab was primarily an exploration of sequential logic (with more than one clock speed) in verilog, with heavy modularity. Our main difficulties lied in the time\_counter module, where we tried several ways to implement counting and second-to-minute overflow. To handle adjust, we eventually used a multiplexer (refreshing much faster than time) to decide which clock (1 or 2 Hz) to feed into the time counters. For overflow, we originally used carry bits, but this caused a delay on minutes (00:59 -> 00:00 -> 01:01), so we instead had the minute counter directly check the current value of seconds and calculate whether to change.

This project seemed like a reasonable step up from combinational logic in the previous lab. One suggestion we have is to include more details about the display controls in the spec, such as how the segments are numbered. The reference manual cited is not particularly clear, and time should be spent figuring out refresh and blinking, not which wire lights up which segment.