CS 152A Lab 3: Stopwatch

Gee Won (Jennifer) Jo, Andrew Lin, Daxuan (Albert) Shu
704171725, 204455444, 204853061
May 25th, 2017

Introduction

In Lab 3, we designed and implemented a stopwatch circuit on the Nexys™3 Spartan-6 FPGA Board. Using buttons and slider switches as inputs, we displayed the digits of the stopwatch through the on-board seven segment display. This involved not only incrementing counter for each minutes and seconds using the clock, but also handling adjust mode at different frequency, selecting the adjust mode, implementing pause and reset with a debouncer.

Design Description

Our design was separated into five distinct modules, each with their own purpose.

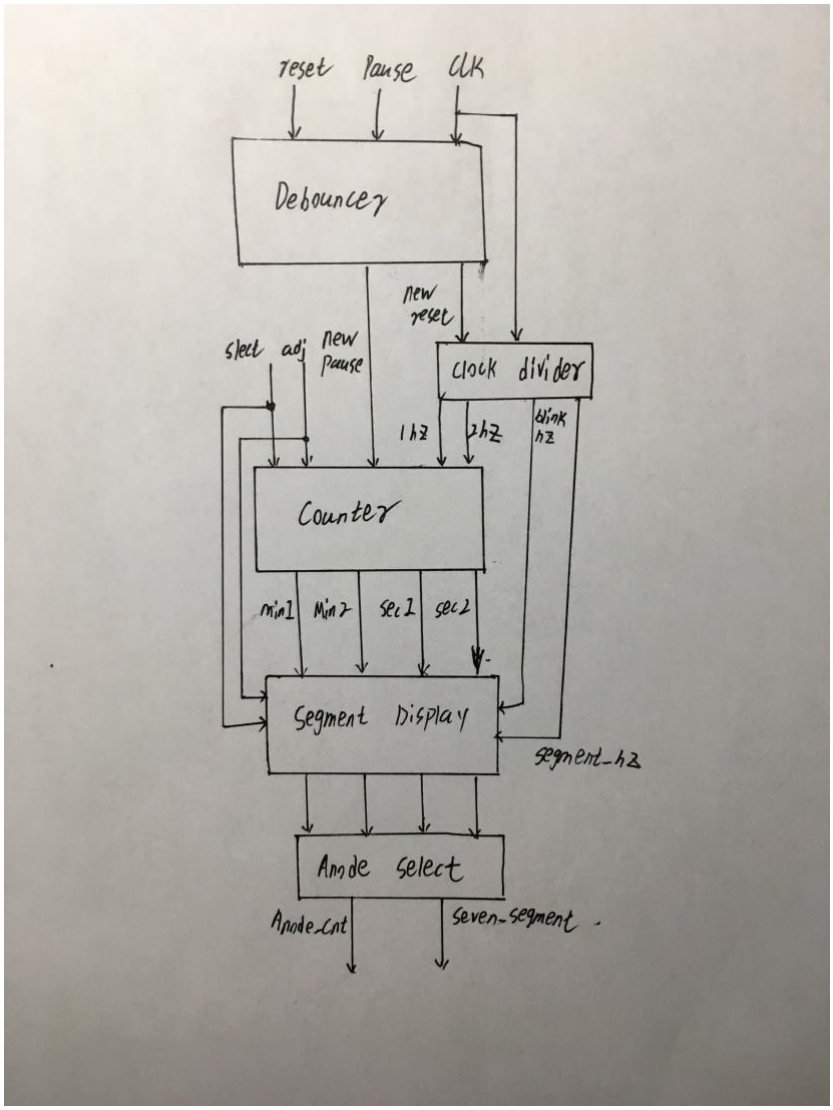


Figure 1. Overall design of the implementation of the stopwatch.

The “debouncer” module was used to essentially stabilize button input. On the Nexys3, the signals received from buttons are not uniform and will usually waver with many dips and spikes. The debouncer module takes the button input and, once the button input has been on for a sufficient period of time, it outputs that the button has been pressed. This leads to less unwanted and possibly buggy behavior from the button. For our lab, the debouncer was used for the pause button, and we used a 16 bit register array for the threshold. This equates to 2^16 = 65536 clock ticks, which is actually just about 0.0006 seconds, but is more than enough to stabilize the signal received from the button.

The “stopwatch” module takes the clock ticks from the Nexys3’s 100MHz clock and converts it to four output clocks with varied frequencies: a 1 Hertz clock, a 2 Hertz clock, a 500 Hz clock, and a 4 Hz clock. The 1 Hz clock is used to increment the stopwatch in its normal mode, the 2 Hz clock is used to increment the stopwatch by two twice every second in its adjust mode, and the 4 Hz clock is used to blink the display while in adjust mode. The 500 Hz clock is used as a multiplexer between each digit we display onto the seven segment display.

The stopwatch module also takes care of all of the logic for the pause state. When the stopwatch is paused, none of the numbers should increment, but the display must still work. Thus, when the stopwatch is paused, we just stop the 1, 2 and 4 Hz clocks from counting, effectively stopping the stopwatch from incrementing, but we continue to allow the 500 Hz clock to count in order to keep displaying the time on the seven segment display. None of the other modules need to know whether the clock is currently paused.

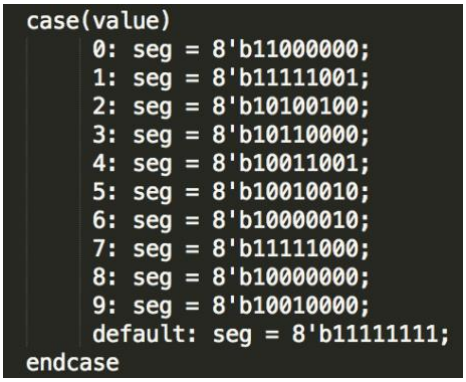
The “timer” module takes just one of the converted clock outputs from the stopwatch module and converts them into the current time to display, as in with minutes and seconds, rather than clock ticks. In the normal stopwatch mode, the timer just increments the seconds by one on each tick of the input clock. In the adjust mode, the timer increments the seconds by two on each tick of the input clock. Various if-statements increment the tens digit when the ones digit overflows, and the minutes whenever the seconds overflow.

The timer module also works very closely with the “clock_selector” module. If the stopwatch is in its normal state, the selector outputs the one Hz clock, which is then passed into the timer. If the stopwatch is adjusting, then the selector outputs the two Hz clock to be passed into the timer. The timer module also receives the current adjust and select states and can choose the right actions to execute on each clock tick, without worrying whether the clock is ticking at one or two Hz.

All of these modules come together in the “display” module. The display module calls each of the above modules to get, in chronological order, the current pause button state, the current clock ticks, the correct clock to use, and finally the current time. Then, on each tick of the fast, 500 Hz, clock, the display module cycles between the ten’s digit of the current time’s minutes, the one’s digit of the minutes, the ten’s digit of the seconds, and the one’s digit of the seconds. It turns the current digit’s place on in the anode output, which turns on only that digit’s place in the seven segment display, and also converts the current digit’s value into its seven segment representation, and outputs that to the seven segment, thus displaying the number.

Simulation Documentation

For simplicity's sake, we ran the stopwatch entirely on the Nexys3 board. We tested each "unit" of the stopwatch separately on the board before combining them into each module, and then linking the modules into our final stopwatch. For example, one of our most difficult challenges was having the seven segment display display our numbers correctly. First, the on and off for each segment was flipped compared to what we were expecting. Secondly, we also got the order mixed up. Originally, we thought that the bits were read from left to right, with index 0 (corresponding to the topmost segment) at the left. Then, we realized that the array indices followed normal arrays, and that index 0 was at the right of the bit string.



```
case(value)
    0: seg = 8'b11000000;
    1: seg = 8'b11111001;
    2: seg = 8'b10100100;
    3: seg = 8'b10110000;
    4: seg = 8'b10011001;
    5: seg = 8'b10010010;
    6: seg = 8'b10000010;
    7: seg = 8'b11111000;
    8: seg = 8'b10000000;
    9: seg = 8'b10010000;
    default: seg = 8'b11111111;
endcase
```

Figure 2: Converting Display Values to Seven Segment Display Bit Strings

In order to turn a specific segment on, we set that index to 0. Index 8 corresponds to the decimal point, which was always unused, and thus is always equal to 1.

Additionally, we originally did not know that we needed to use the anode output to control which digit was being displayed. This caused all of the values to appear wrong, with all of the digits overlapping each other.

Our main goals for our tests were to make sure that normal incrementation worked, all of the overflows worked regardless of the current modes, the buttons (Pause and Reset) worked, and finally that the adjust mode worked. These were also the major functionalities that we recorded for our video demo.

Conclusion

In this lab, we designed the stopwatch module with many components. We worked with clocks, synchronization in different modules, I/O from button, switches and seven segment timers. We also learned techniques like debouncing.

Modularizing our design for this project was very effective. Although we encountered some confusing bugs, our modularization helped us quickly locate and efficiently fix every one of them. For example, when our pause functionality did not work correctly, we investigated the bug in the stopwatch module, because that was where we implemented pausing. And when we encountered problems with displaying each digit, we knew we had to look for the bug in the display module.

The most problematic bug we encountered was the anode output for the seven segment display. When we read the project specs, we did not understand what the anode output was,

and we did not write the code to implement that. It took a lot of failed troubleshooting before we found out how to use the anode output from the TA. After we fixed that, our display finally worked.

Our group members worked together on all parts of the project, designing how each module would work, writing the code for each module, and figuring out how to debug each module.