Karthik Ramesh

104687637

Chungchhay Kuoch

004843308

Lab 3: Stopwatch

Introduction

The purpose of this lab report was to create a stopwatch wherein we used Verilog modules that connected together to make the whole stopwatch. In this stopwatch, we have functionalities, such as adjusting the minutes and seconds part of the watch, which we used two switches, one each, for adjusting and selecting. We also implemented different clock modules for keeping track of the different clocks that are required in this lab, such as 1hz incrementation for the normal incrementing, 2hz incrementation when we are adjusting the minutes/seconds, the master clock, the clock for showing all the four numbers at the same time in a cycle to make it seem like they appear at the same time for the naked eye, and the clock for how fast we wanted the numbers to blink. In addition to that, we implemented pause and reset buttons, and debouncers for each of the buttons.

Design Implementation

Top Module

We designed many modules, which all do different jobs that come together in a top module. In this module, we named this the main stopwatch module. We implemented our counter in this, too. This counter is a 3600 counter, so we implemented it differently from the way the TA suggested. We did out 2hz and 1hz clock in this module. If we noticed that the pause or reset button was pressed, we did the corresponding responses in this module.

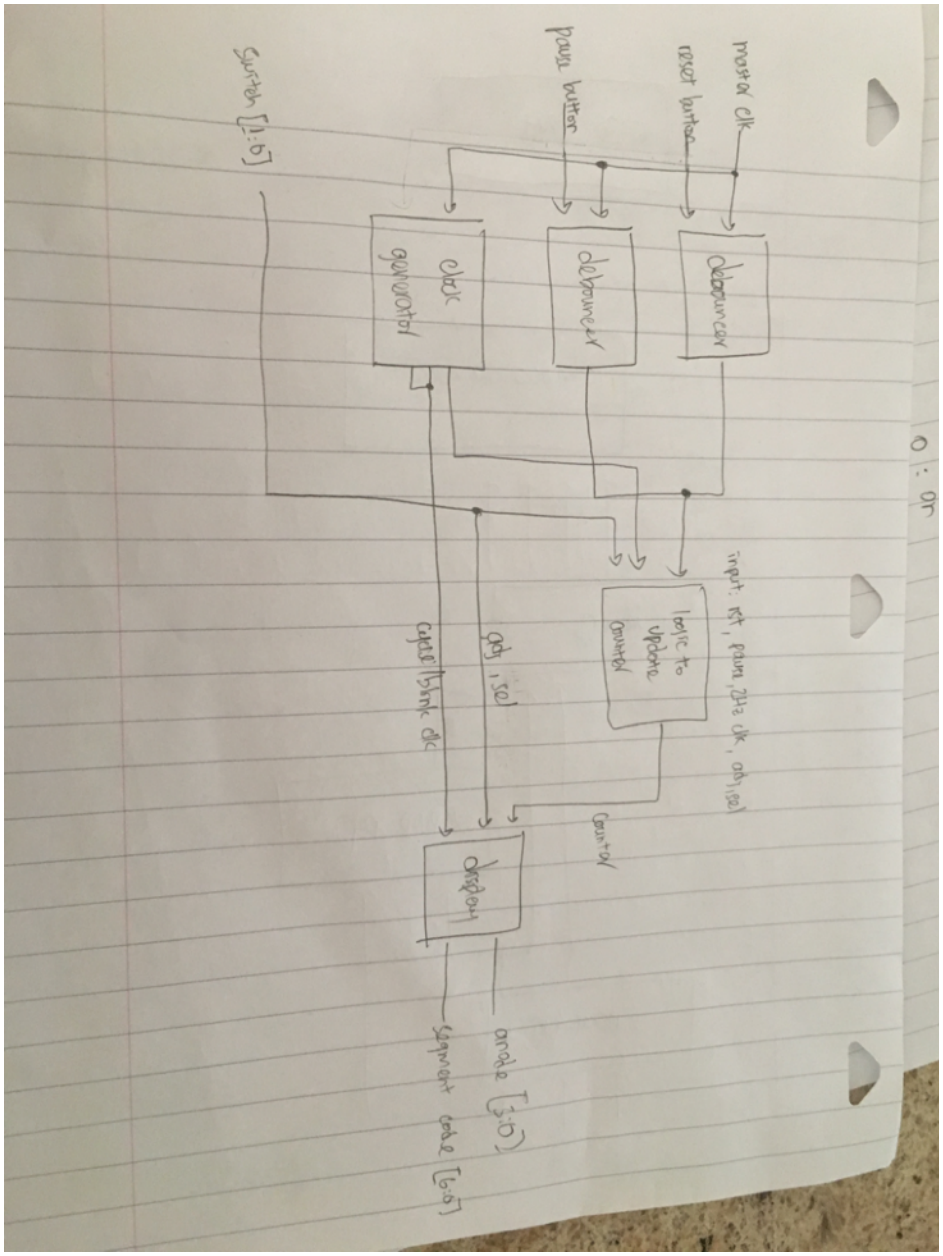

figure 1 with top module circuit design. The basic of this design is to have a counter that is incremented every one second and we perform the modulo operations and divisions to generate various numbers.

Clock

In this module, we make all of our different clocks: 2hz, cycling clock, and blinking clock, feeding them into another module that actually creates corresponding clock cycles for each of them. Then, we proceed to use them in the top module, and display modules to periodically update the display counter, the display itself, and the periodic blinking for when the adjust switch is turned on. The 2hz clock was used for adjusting the clock in adjust mode. (The 1hz clock was never tru-

ly implemented, but we did a pseudo-1hz clock in that we updated the clock “normally” at 1hz by updating it every other cycle in the 2hz cycle, which created the impression that we are running at 1hz.) The cycling clock was for updating the clock so fast that the 4 separate numbers we presented on the board will appear as if they are all happening at once.

Seven Segment Display

This took in the counter and turned on and off the corresponding parts of the seven segment display to feed into the show display module that called this function four times, one for each of the four different segments.

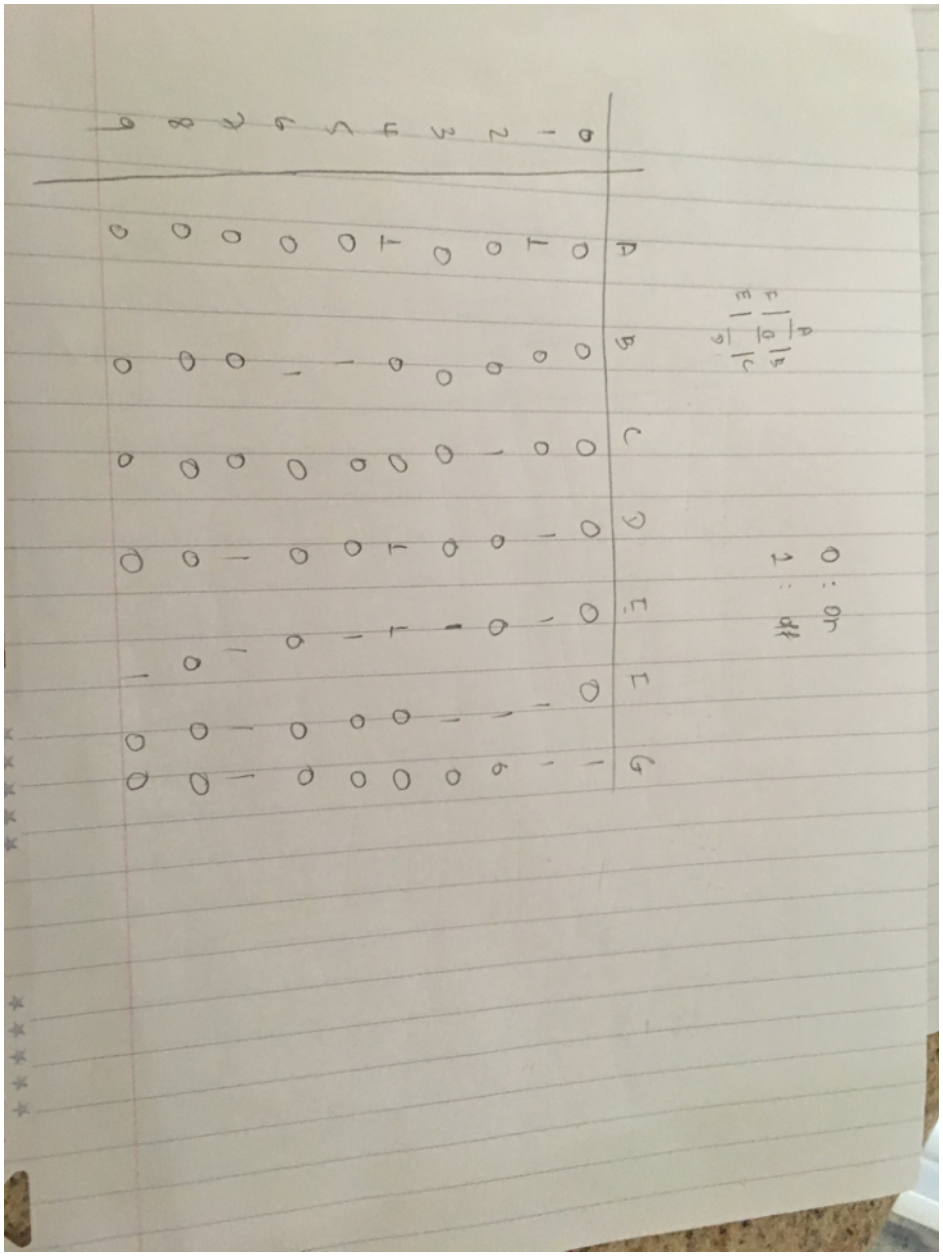

figure 2 with seven segment display. 0 represents ON while 1 represents OFF. The alphabets represent each segment display and each number has their own segments to display.

Debouncer

In this module, we made sure that when the button is pressed, it only registers when the button is at a stable state, due to the tension springs, and other parts that might affect what will happen along the way to pressing the button. We used this on both the pause and reset buttons.

Show Display

In this module, we converted interpreted 1's as OFF, and 0's as ON, for the seven segment display, depending on what number was supposed to show up on the screen, which came from the counter from the top module. This was connected to the nexys3 ucf file. This file also had to use the cyclic fast clock for showing the numbers as if they are all appearing at the same time.

Insert figure 4 with how show display would work in circuit design

Testbench

Figure 5: From this figure, you can see the output for the input number 11, which should translate to the seven segment display: '00:11'. When the clk_cycle reaches the 1110, and 1101 parts, the segment is updated to show up on display—you can see this where the segDis switches only at the edge for the clk_cycle when clk_cycle is 1110 and 1101. This shows the output of both our display and top modules. They both output the same things.

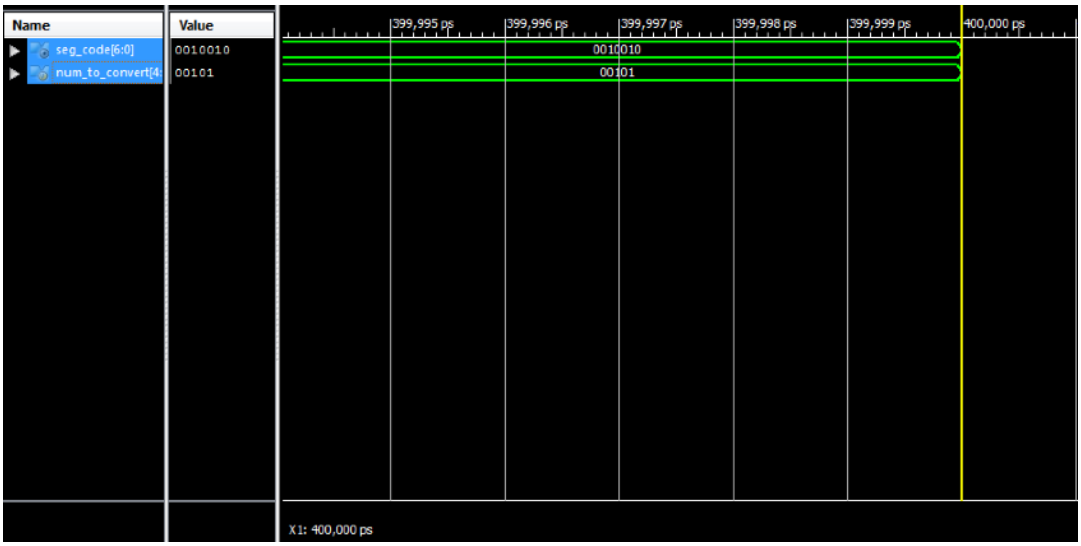

Figure 6: In this figure, you can see that the seg code part of the lab is owrking properly, as for the input of '5', the seg code translation should have 5 lines, and 2 blank spaces, and that shows up here, because the 0's mean lines, and the 1's mean blank spaces. Of course the arrangement matters, too, which cannot be easily seen, but trust that it works.

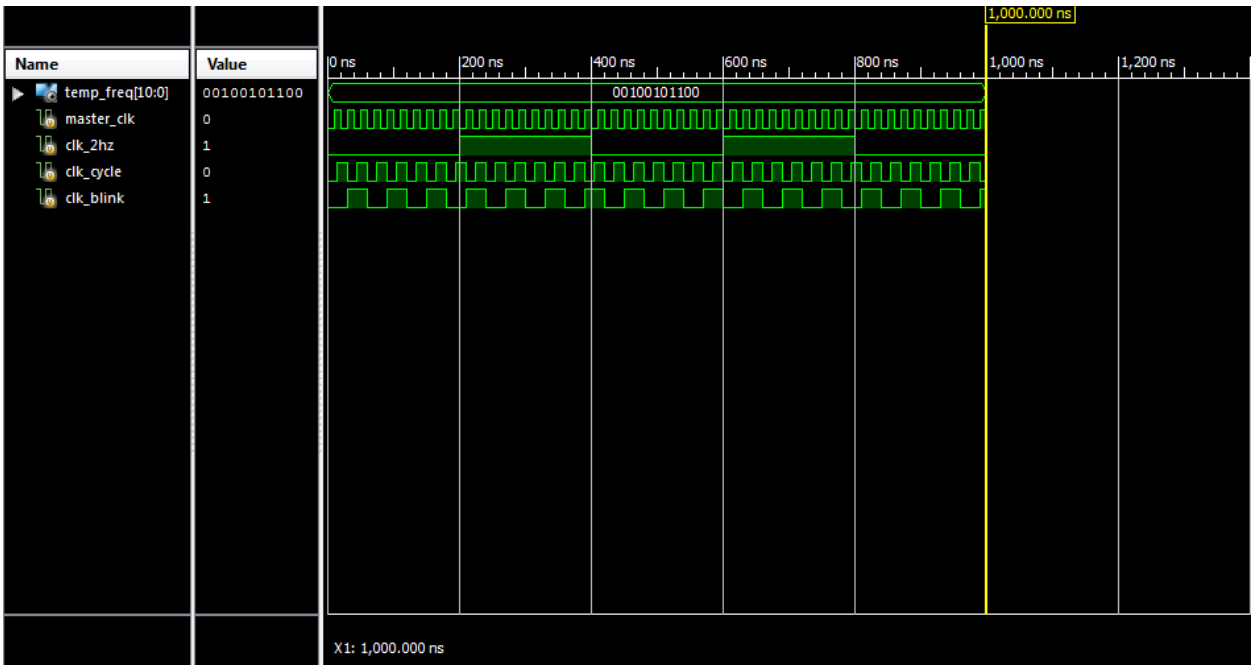

Figure 7: From this figure, you can see that I assigned arbitrary values for the different clock dividers, and they all appear on the testbench. (The temp_freq shown above is a random thing not related to the results shown with the different clocks below).

Conclusion

This was the toughest lab we have done so far. There were a lot more modules, and the integration of modules within modules inside modules was a new thing we undertook in this lab. Additionally, we gained experience on creating testbenches, as well as learning some of the quirks of the Xilinx ISE compiler, such as making sure to have an if-then-else syntax, instead of just if statements like in C or C++, because then, the compiler sometimes might throw errors because it is automatically expecting it (note: you, the TA, were there for this, as you had to see that we had to include a blank else statement for one of our if statements to simply make the compilation error go away, and then it worked properly).