Francis (Haocheng) Feng - 205 339 293
Megan Pham - 505 313 300
CS M152A Lab 2
Dipti Sahu

Lab 3 Report

Introduction

A field-programmable gate array, or FPGA, is a semiconductor device that contains logic blocks and interconnects, allowing people to design and test their applications. We will finally be using an FPGA board to create a stopwatch, counting minutes and seconds, complete with a seven-segment display. In addition, we will be able to adjust, reset, and pause the time using the buttons and sliders that are connected to our code.

Module Design

In order to create our stopwatch, we utilized seven modules: clks, clock_type, digits, debouncer, seven_segment, show_segments, and the main stopwatch. Each module corresponds to one of the requirements in the lab instructions.

In the first submodule, **clks**, we created the four different clocks we needed to use, 2Hz, 1Hz, a fast clock, and a blinking clock. To calculate our clocks from 100 Mhz, we used the following logic: say we want to build a 1HZ from 100 MHz. Every time the clock hits 50,000,000, we will toggle a flip-flop. Toggling at 2Hz (100 million/50 million =2) gives us a 1Hz clock. For the 2Hz clock, we toggled at 25 million counts. The fast clock rate we chose was 500Hz, so we toggled the pin at 100,000 counts. Last, the blinking clock rate we chose was 1.8 Hz, and since it does not divide perfectly, we rounded to 27777777 counts (toggle a flip-flop at 3.6Hz gives us a 1.8Hz clock at 50 percent duty cycle).

Our next module was **clock_type**, which would allow us to choose the regular clock and the adjust mode clock, so if the adj flag was true, then we would set the main clock to 2Hz, and if not, we would keep it at 1HZ.

We would then call this submodule in **digits**, a module that allowed us to convert our counts into seconds and minutes. We attached wires to each position of the clock, the ones place for seconds, the tens place for seconds, the ones place for minutes, and the tens place for minutes. If the reset button was pushed, we set all those positions back to 0. However, if adjust and pause is not turned on, then we can allow our clock to begin incrementing by 1 each second. However, once we hit 59 seconds, we reset seconds

back to 00. We then check for minutes and reset that to 00 as well if it is set at 59. If it's just the ones place set at 9, then we can increment the tens place of minutes up by 1. Last, if it's just the ones place of seconds set at 9, then we can increment the tens place of seconds up by 1. Now, we will check for the adjust button and pause button. If the adjust flag is 1 and select is set to 0, then we need to keep the seconds still and only increase the minutes, setting 59 or _9 back to 00 or _0. The same thing happens when adjust flag is 1 but the select flag is set to 1. We will freeze our minutes and keep incrementing our seconds. If the pause button is clicked, then we do nothing and the minutes and seconds will stop incrementing.

In order to combat the oscillating behavior generated by hardware inconsistencies, we created a module called **debouncer**. This module serves to filter out the noise created by pressing a button and only registering one valid push. By using a flip flop as a "buffer", we can use the output of the flip flop as the input for the button. This method will allow for a consistent output to all other components in the stopwatch.

So far, our modules have been working with the decimal form of the minutes and seconds displays. However, in order to display the time correctly on the FPGA board, we must convert the time from decimal form into a seven-segment encoding. The module **seven_segments** takes in a decimal as input and churns out an 8-bit representation of the seven-segment display for one of the time slots. If the digit is invalid, any number not in the range from 0 to 9, then all 8 bits will be set to 1 which translates to an empty display.

The purpose of the **show_segments** module is to correctly display the seven-segment display on the FPGA board given the inputs from the other modules. At the top level, the module uses a register to keep track of which of the four positions on the display that the module is updating. The register counts from 0 to 3 and repeats as long as the program is being executed. For each position, the appropriate anode is updated. Additionally, the module must be able to handle two different states, adjust and regular. During adjust mode, the display has been programmed to blink at 2 Hz. During regular mode, the display will blink at a normal 1 Hz. At each tick, the display will be updated with the most current times in an 8-bit format.

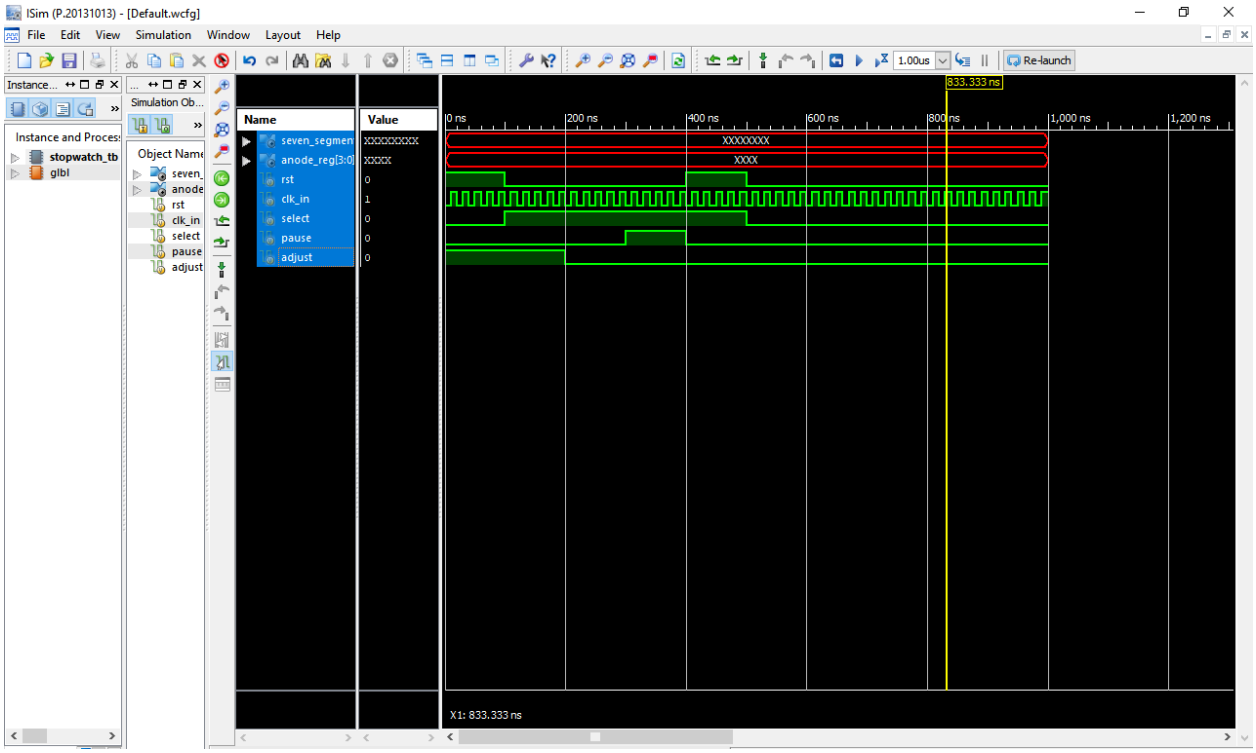
The final module is the main **stopwatch** module that will call all of the other modules and connect them at the top level. This module first calls the debouncer module to correctly configure the buttons. Then the clocks must be set up so that they can be used later by other modules to clock the displays. Next, the digits module is called to initialize and update the current time in decimal form. The output of the previous module will then be passed on to the seven_segments module to convert the time from decimal form to

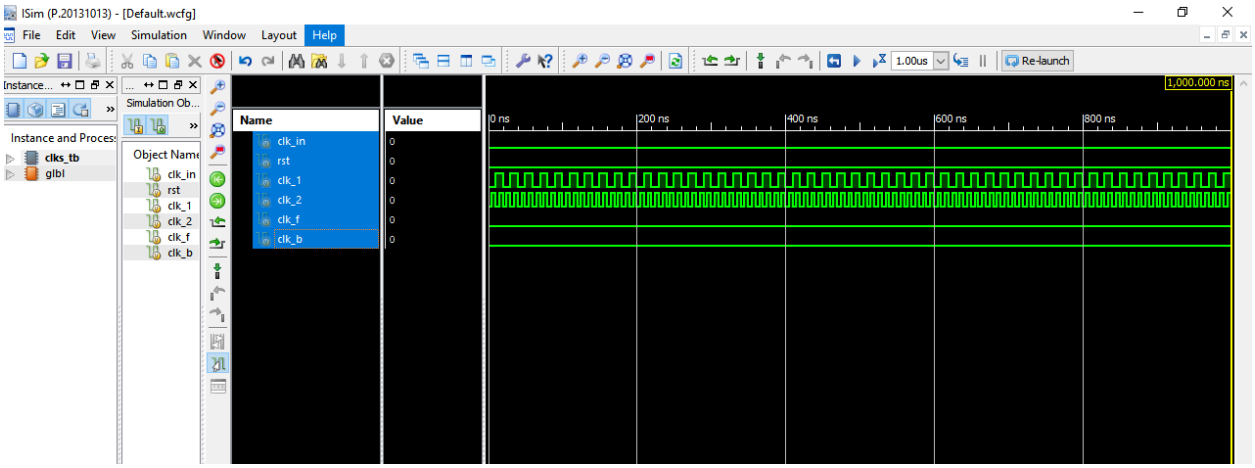
an 8-bit representation. The four 8 bit representation output will be used as input for the show_segments module which will help to display the appropriate seven segments in all four positions.

Testbench Design

In our testbench code, we made sure to test every module. As modules were intertwined with each other, it made sense to primarily test the regular clock, blinking clock, and the different sliders and buttons to ensure that each module was running correctly. To make our testbench, we made sure to toggle each switch/button (pause, adjust, and reset).

Simulation Waveforms





ISE Design Overview Summary Report

stopwatch Project Status (02/16/2022 - 12:13:30)			
Project File:	lab2_p2g4.xise	**Parser Errors:**	No Errors
Module Name:	stopwatch	**Implementation State:**	Translated
Target Device:	xc6slx16-3csg324	• **Errors:**	
Product Version:	ISE 14.7	• **Warnings:**	
Design Goal:	Balanced	• **Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	
Environment:	System Settings	• **Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	72	18224	0%
Number of Slice LUTs	168	9112	1%
Number of fully used LUT-FF pairs	72	168	42%
Number of bonded IOBs	6	232	2%
Number of BUFG/BUFGCTRLs	1	16	6%

Detailed Reports						[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**	
Synthesis Report	Current	Wed Feb 23 12:12:04 2022				
Translation Report	Current	Wed Feb 23 12:12:17 2022				
Map Report						
Place and Route Report						
Power Report						
Post-PAR Static Timing Report						
Bitgen Report						

Secondary Reports			[-]
Report Name	**Status**	**Generated**	
ISIM Simulator Log	Current	Wed Feb 23 12:59:11 2022	
Post-Synthesis Simulation Model Report	Current	Wed Feb 23 12:12:13 2022	
WebTalk Report	Current	Wed Feb 23 12:43:29 2022	
WebTalk Log File	Current	Wed Feb 23 12:43:30 2022	

Date Generated: 02/23/2022 - 13:00:33

Conclusion

Overall, we created a fully functional stopwatch using Verilog and the FPGA board. We were able to implement four different clocks as well as buttons and sliders to adjust the time. One difficulty we encountered was using the FPGA boards as this was the first time we came into lab due to COVID-19 restrictions and as a result, as first time using the hardware. However, after some testing and practice, it became a lot easier to use.