Aditya Gopalan
adgo4176@colorado.edu
109052749
10/7/2019

ECEN 2350 - Lab 1: Write-Up

1. Design Block #1:

In this lab, we were able to learn the basics and the function of Verilog and the way that it was able to function with the FPGA board. We were able to learn how the different functions and the basics of digital logic were used to make our board able to do basic calculations and display certain things that we wanted it to do. In this design block, our goal was to create the board display our birthdays with a given input of binary number and make the 7-segment display on the board itself to be able to display this. With the press of a button, the display should be able to switch the display to our partner's birth date.

We were able to create this code to make this happen by assigning 6 HEX variable to different bits on the board from bit0 to bit23 in out main designblock1.v file. In this file we assigned the parameter of our birthday in a 24 bit value for which we input into the bits that were mentioned above. Once that was done, we made sure that if no key was pressed on the board, it would display one of the partner's birth date, while if they key was pressed, then the other's birthday was displayed. After we had established that the birthday segment of the code was working with no errors, we needed to make a function called print7seg.v that allowed us to generate a binary value for each of the numbers that the 7-segment display would be displaying. We converted this 4 bit binary to that of an 8 bit number that we would use to let the board understand and upload to the board using the Quartus software. We then went back to the designblock1.v file as made sure that we assigned each segment of the display to a bit on the board and a corresponding HEX number that ranged from HEX0-HEX5. This would later help with the numbers that would be displayed on the board.

For our testbench, we needed to check that even though that the board was displaying the correct numbers for our birthday, that it was actually going with the computation of both of the designblock1.v files as well as the print7seg.v files correctly. To do this, we called all of the variables used in both of the previously mentioned files and tested each of the functions and combinations of the key presses and switch modulations to ensure that the code was working as calculations rather than simply stating what should be outputted. We used the $monitor initial block as that was able to call not just out LED and SW and KEY variables, but it was able to see what the binary values would be under different time every 10 milliseconds. This would be displayed in a txt file that would show what the binary value should be as well we what was actually outputted.

In this design block we were able to get the demonstration to work properly, though it did take a little long to do and a lot of errors in syntax to get it right.

Aditya Gopalan
adgo4176@colorado.edu
109052749
10/7/2019

2. Design Block #2:

In this design block we had to use 2's complement to essentially create a full adder to the FPGA board. With the flip of a switch at the bottom of the board we would be able to use binary to add and subtract numbers using the 2's complement method and get an output of another number on the 7-segment display. If this number, once added or subtracted was too big or too small, it would go into an overflow error which should be shown on the board as OF on the 7-segment display to signal that the adder has calculated an overflow error.

To do this, we had to start with initializing the variables again. The variables would be named the same as that of Design Block 1 as a lot of the code would be based on the same design with some changes. The 7-segment display function that was referenced in design block 1 would be the same code that we will reference again in this block so that would not change. However, we would change a lot of the main. In this case, we had to introduce a variable of the carry variable as well as the sum and negative functions as this was to allow the adder to add and subtract both positive and negative numbers and we were able to use a regular 2's complement method without any complicated manipulation and being able to carry the 1 over to get the right final answer. To do this code, we had to use the basic functions and knowledge of the 2's complement to switch all of the binary values to the opposite of the original if it was negative, then add one and then add the first number to get the final value. We were able to do this by using a carry and negative function as we were able to make the code do what we wanted when it came to the function properties.

We did, however, need to be careful about the overflow errors we according to the lab, any number that results in higher than +7 or lower than -8 would be an overflow error when doing a 4 bit 2's complement addition or subtraction. To ensure that we were able to correct for this overflow, we created a wire overflow to check to see if the number was outside the desired range. If it was, it would display a OF or would keep the answer display blank to signal that it was an overflow error. In the fulladder.v file that we created we made sure that we were able to add three inputs and produces two outputs. The first two inputs are (a) and (b) and the third input is an input carry as Cin. The output carry is designated as Cout and the normal output is designated as S which is SUM. We wrote it so that if two 1s sum = 0, if 3 1s sum = 1, if only 1 1 then sum = 1. Also, we would assign cout to be a value if and only if there are at least two ones in inputs.

We were able to get our demonstration to work for the most part as we got majority of the adder and subtraction problems to work, however, we have one or 2 solutions that do not work either because they are incorrect in the arithmetic or the

overflow error would not be called properly to ensure that the output was shown as overflow. If we had a little more time, we would have been able to figure it out.

3. **Design Block #3:**

In this design block, we had to use the FPGA board to essentially create a number comparer. This would be in the case that the switches on the board would be assigned a binary value of either positive or negative and when it was flipped, it would be shown on the 7-segment display. Once that was shown on the display itself, it would be then compared to one another and the LEDs above the first 3 switches would light up depending on what the operator was to determine what value of what number was going to be higher or equal.

This design block was relatively simple as we were just comparing 2 numbers. We assigned value of x to be bits 0-3 and value of y to be bits 4-7. We used multiple wire assigned variables as we were able to compare them to the final result. Since we aren't able to use regular if and else if statements to compare the numbers, We used LEDR and printnum to to check to see if the number was less or more than the other number. Once this was done, we a true or false case using the “?” operator to see if it was true (1) or false (0). Then once that was determined, we were able to output the result to the LEDs on top of the switches. We assigned the LED0 to be less than, LED1 to be greater than and LED3 to be equal to. Once these numbers were compared using the 4bit binary values we were able to light up the corresponding LED to determine which number was less than, greater than or equal to the other.

For this testbench, we were able to use the relatively same one as the first testbench, however, we needed to check again for the value since this was a simulation of what the board would show. We called the functions in the testbench every 30 milliseconds that allowed us to see the progression of the way that the board was able to calculate the comparison. Once that was done, we outputted the 10 bit binary value to the txt file and compared it to what the correct answer should be.

Our demonstration worked, for most of the test cases that we were giving it as well as on the board, however, we found it frustrating that some of the comparisons would not show the correct result. We would check our code for hours on end and was not able to find what the issue was. There were many times that the board would get the absolute value of the number and its comparison correct but some of the times when the negative was involved, it would be incorrect.

Overall in this lab, we learned a lot about the basics of the syntax and the structure of Verilog. Though it was challenging for a lot of the ways to code the FPGA board, it was helpful to see the way that logic was able to play a role in this code. The most frustrating part of the lab was the fact that some of the random solutions that the board would produce would be wrong. However, I feel that if I had a slight bit more time

Aditya Gopalan
adgo4176@colorado.edu
109052749
10/7/2019

to work on the lab, I would have been able to eventually figure out what the issue could have been and could have been able to solve it.