ECE 4250/ 7250: VHDL and Programmable Logic Devices Laboratory

Lab #: 07
Lab Title: BCD Converter
Group #: 5
Names: John Kelly | Zach Rump

Teaching Assistant Use Only:

	Points Earned	Reasons for Deduction
Pre-lab:		
Post Lab report:		
Demonstration:		
Final Lab Grade:		

Comments to students:

Objective

The objective of Lab 07 is to implement BCD to Binary and Binary to BCD converters on Spartan-3 FPGA using the technique of separating data path and control section. The system will take a 16-bit input (loaded twice from the 8 switches) and display the conversion result as hex digits on the seven segment displays.

Implementation

The implementation is split into three categories: The code to drive the seven segment displays, the conversion implementations, and the test bench. The file Dec_7seg.vhd describes which signals need to be sent to the 7-segment display for a given hexadecimal digit. The AnodeControl.vhd file cycles through the different seven segments and enables them by toggling the Anode. The test bench, Lab7Bench.vhd is used in conjunction with the Lab7Bench.ucf file in order to interface with the hardware. The end result is that the Spartan-3 hardware drives test bench which drives the BCD2BIN and BIN2BCD components. Finally, the LEDDisplay.vhd module is used as a component in the test bench, and ties all the modules together.

BCD2BIN

For the first part, converting from BCD to binary, we read in the upper 8 bits first, stored it in a temp signal, then read in the lower 8 bits and stored them into another temp signal. Once we got to our third state (after both numbers have been pressed AND released), we concatenated the 8 bit vectors together into one 16 bit vector. The 16 bit vector was then parsed into four sub vectors, each getting 4 bits that represent the ones place, tens place, hundreds place, and thousands place on a BCD number (I.E. the lower 4 bits represent the 10s place, the upper most 4 bits represent the thousands place). Each of these four vectors was multiplied by its placement, such that the ones place was multiplied by one, the tens place by ten, hundreds place by hundred, and thousands place by a thousand. Each of these scaled vectors were then added into a final output vector such that the result formed the binary equivalent of the BCD vector. The whole process took three states, where the first state read the upper 8 bits, the second state read the lower 8 bits, and the last state computed the binary equivalent of the BCD number.

BIN2BCD

For the second part of the lab, we had to reverse the process in order to determine a BCD number from its binary equivalent. In order to do this, we broke the process into three states, similar to part one. The first state read in the upper 8 bits of a 16 bit number, the second state read in the lower 8 bits of a 16 bit number, and the final state computed the BCD conversion. More specifically, the third state concatenated the two 8 bit numbers together from state 1 and state 2 into a 16 bit std_logic_vector and then iterated through a loop for the length of the vector. Inside the loop, the standard shift and add three method was used to compute the final BCD equivalent. Since a 16 bit vector could produce a larger than 16 bit BCD vector, we created five four bit vectors in order to computer the result, but disregarded the upper most 4 bits so that the final result was only a 16 bit vector. Originally we had used the method of dividing the number by 10, multiplying by 6, and adding that to the originally number to compute the BCD equivalent, which worked just fine, but our final method corresponded more with the method approached in class.

Conclusion

The main challenge with this lab was understanding the algorithm to actually perform the conversion. After getting the simulation to work, most of the issues were discovered in attempting to synthesize the code. For example, in the BCD2BIN and BIN2BCD components we got warnings about latches, and signals used but not assigned. The warnings themselves didn't make the specific issue very clear, but researching the error message gave quick insight into what the actual issues were. For example, many of the latch warnings were caused by failure to check the clk'event in the clocked process. Similarly, we had initially intended to use a control signal called DONE to indicate the end of conversion. This showed up as a warning because we never actually used it for anything, and could remove it without issues.

This lab contributed to our VHDL knowledge by forcing us to consider how to design the HDL in order to make the implementation as straightforward and easy to understand as possible. Additionally, it provided us with practice splitting the design up into datapath and control section, and understanding how VHDL code can be modularized effectively. Lastly, this lab forced us to acknowledge the differences between simulated and synthesized VHDL, and helped us better understand best practices for writing synthesizable code.

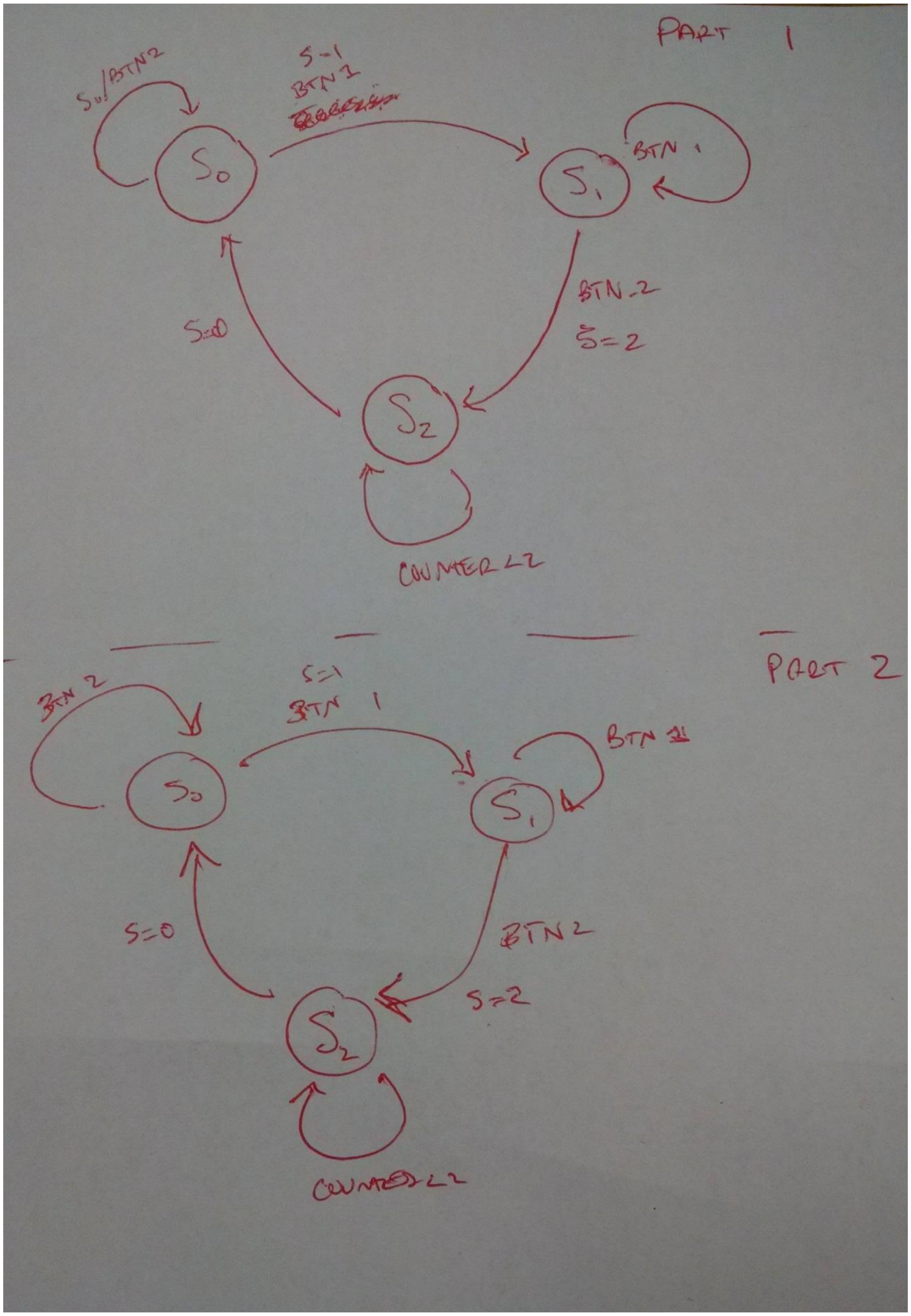

Figure 1. State diagram for parts one and two.