SANTA CLARA UNIVERSITY	ELEN 127 Fall 2018	Jim Lewis

Laboratory #2: BCD Adder in Verilog

For lab section on October 5, 2018

I. OBJECTIVES

A first foray into Verilog-based design, implementing a non-trivial combinational circuit.

PROBLEM STATEMENT

Binary Coded Decimal (BCD) is a numbering format where a 4-bit binary quantity is used to express a single decimal digit. So each 4-bit BCD value can only express the numbers 0 thru 9.

You will create an adder that takes two 4-bit BCD numbers and generates the sum. But the sum must be expressed in BCD. The adder will also take a carry-in and generate a carry-out. You will use a "normal" 4-bit adder to generate an initial 4-bit sum plus carryout, but then you will need to implement a conversion circuit to convert this into two BCD values (For example, a sum of 1010 would need to convert to the BCD value of 0001 0000.)

You will use this to connect two of your 4-bit adders to create an 8-bit adder.

II. PRE-LAB

(i) Specify the logic for a 7-segment decoder that takes a 4-bit BCD input and generates the correct 7 control signals to display the decimal digit on a 7-segment display. (Note that this decoder is likely different than the decoder you made for lab 1. This one is "normal".) You'll want to implement this first thing when you get to lab.

(ii) Draw a block diagram for how you might approach the conversion of the 4-bit sum (i.e., the output of the "normal" 4-bit adder) into two BCD values. Consider the concept of BCD "overflow", i.e., if you get a sum larger than 9. If you don't have overflow, then the 4-bit sum can pass thru unchanged. But if you have overflow, you will need to pass thru a different 4-bit value, which is some function of the 4-bit sum. We'll call this the remainder.

(iii) Specify the logic for detecting BCD overflow. I.e., something that takes the outputs of the "normal" and asserts an output if the value cannot be expressed as a single BCD digit.

(iv) Specify the logic for the modified 4-bit value (the remainder) that the conversion circuit should generate if BCD overflow is detected.

LABORATORY PROCEDURE

You will again be paired with a partner. You will divide and conquer on the implementation, each of you working on parts that need to be implemented, and then finally combining the pieces together into one top-level design that can be downloaded into the eval board.

1. Implement your 7-segment decoder:

   (i) Write the Verilog module code for your 7-segment display (part (i) of the pre-lab).
   (ii) Simulate it to make sure it appears to be working properly.

2. Implement the BCD overflow detection :

   (i) Write a Verilog module that implements the logic you specified in part (iii) of the pre-lab.

   (ii) Simulate it to ensure it works.

3. Implement the module to generate a remainder in the case of overflow:

   (i) Write a Verilog module that generates the remainder to be used in case of BCD overflow.

   (ii) Verify it.

4. Implement a 4-bit BCD adder:

   (i) Implement a "normal" 4-bit adder and then implement the block diagram you created in part (ii) of your pre-lab. This should yield a module that take two 4-bit inputs and outputs a 5-bit BCD value.

   (ii) Verify it.

5. Create a top-level module:

   (i) Instantiate your 4-bit BCD adder and two of your 7-segment decoders. One 7-segment decoder will take the low four bits of your adder output, and the other will take the remaining bit. Connect the outputs of the decoders to two 7-segment displays, and use 8 input switches to provide two 4-bit values to feed into the adder.

   (v) Compile, download, and test. When you think you have it, demonstrate to the TA.

6. (Bonus): Implement a two-digit adder

   (i) You should have the hang of this by now. Show the TA when you have it.

III. REPORT

For your lab report, include the code for your 7-segment decoder and for the BCD adder. Also include a diagram of how the entities were instantiated and connected to create the 2-digit adder (if you managed to do this). In addition, include answers to the following questions.

- What problems did you encounter while testing your steps yourself?
- Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these testcases escaped your own testing.