ECE 171
Fall 2014
Project 2 Description

Ordinary adders can be used to perform BCD arithmetic. However, since six of the possible sixteen four-bit codes are not used in BCD representations, whenever the result of an addition exceeds 9, those codes must be skipped. This is done by checking the result of each addition operation. If the result is greater than 9, then 6 is added to that position, with a carry to the next position.

For example, when we add 37 + 15, we first add 7 + 5. The result is greater than 9 so we add 6 to the result, obtaining 2 with a carry of 1. Adding 3 + 1 plus the carry yields 5. Thus the result is 52.

```
   0011  0111
 + 0001  0101
-------------
         1100  (result of adding 7 + 5 is > 9)
         0110  (add 6 to result)
-------------
      1
   0101  0010  (carry out is added to addition of digits to left)
```

Note that the adders must each accept two 4-bit BCD numbers and a carry in and produce a 4-bit result and a carry out. The result of the BCD addition for the second digit could also be greater than 9 in which case 6 must be added to its result as well.

When you add two 4-bit numbers (and a carry in) you get a 4-bit sum and a carry out. When considering whether the result of an addition is >9 you'll need to consider all five output bits (the sum and the carry out). For example, if we had added 9 + 9, we have 1001 + 1001 = 0010 with a carry out and we'd need to compare 10010 to see if it's >9. Note that you can use the Verilog concatenation operator to form a 5-bit value from a 1-bit and 4-bit value. So, for example, if Cout is the carry out and Sum is a 4-bit value, you can use {Cout, Sum} to form a 5-bit value where Cout is the MSB. You'll find that useful for forming the input to your module that compares a 5-bit value to 9.

When you add 6 to a result that addition could cause a carry out as well. So, the easiest way to generate the carry in to the next column of digits (without having to do a five-bit addition) is to generate the carry to the next column if either of the two additions in the current column (the first one, and the second one which may have added 6) yielded a carry out. So above, 0111 + 0101 didn't generate a carry but the add of 6 to the result (1100 + 0110) did. If we had instead added 9 + 9 (1001 + 1001 = 10010) we would have had a carry at the first add and would use that to drive the carry in of the next digit.

For this project, you must design and model in Verilog a hierarchical behavioral dataflow description of a two-digit BCD adder. That is, it will accept two two-digit BCD numbers and add them. I will provide you with a working 4-bit adder module defined as below. You must create a module that accepts a 5-bit value and outputs 1 if it is greater than 9. Then, using instances of this module and the 4-bit adder module, you must create a one-digit BCD adder (that adds two one digit numbers). You can then create the two-digit BCD adder from these this module.

Model each module so that it has a 10ns propagation delay.

You must declare your two-digit BCDAdder as follows. This adds the two-digit BCD number X1 X0 to the two-digit BCD number Y1 Y0 (along with carry in Cin) and produces a two-digit BCD number Sum1 Sum0 and a carry out Cout.

```
module BCDAdder(X1, X0, Y1, Y0, Cin, Sum1, Sum0, Cout);
input [3:0] X1, X0, Y1, Y0;
input Cin;
output [3:0] Sum1, Sum0;
output Cout;
```

As you'll see from the code posted on the course web site, the 4-bit adder I'm providing for your use is defined as follows:

```
module Adder4Bit(X, Y, Cin, Sum, Cout);
input [3:0] X,Y;
input Cin;
output [3:0] Sum;
output Cout;
```

I recommend you create a different project for each testbench and add to the project only the user source files for the testbench and all modules it instantiates (recursively).

You will turn in

- A behavioral dataflow Verilog module that determines if a 5-bit value is >9.
- A testbench to verify that this compare module works correctly
- A behavioral model that adds two one-digit BCD numbers and a carry in, producing a BCD digit sum and a carry out (using the 4-bit adder provided and the compare module you created).
- A testbench to verify that this BCD digit adder module works correctly
- A behavioral dataflow model that uses two instances of your BCD digit adder to create a 2-digit BCD adder
- All design documentation (black boxes, truth tables, K-maps, etc)