ECE199JL: Introduction to Computer Engineering **Fall 2012**
Homework 7 **Due: at start of lecture on Friday 12 October**

Logic Structures, State, and Latches

Please do problems 3.27 and 3.30 from the textbook.

Here are four additional problems for this week:

1. Floating-Point Comparator
Design a logic circuit that operates on the bits of two IEEE 32-bit floating-point numbers A and B to determine whether $A > B$. (You may wish to consult the design in Notes Set 2.4 for ideas.) Sketch the logic diagram of your circuit as your answer, and explain the strategy and derivation of your circuit.

2. Bit-Slice Power-of-Two Checker
Design a bit-slice circuit that checks whether or not an unsigned integer is a power of 2, starting with the least significant bit. The following steps may help guide you to a good solution. (You will likely find it very helpful to review course notes sets 2.3 and 2.4.)

A. List the possible 'answers' (or information) that your bit slice may need to communicate to the next bit slice (and receive from the previous bit slice).

B. Choose a representation for these 'answers'.

C. Write out truth-tables for each output bit from your bit-slice in terms of all inputs to your bit-slice. (What are all of the appropriate inputs?)

D. Convert these truth-tables to Karnaugh maps and find simple logical expressions for all outputs.

E. Draw gate diagrams for each output to create your full design for one bit-slice.

F. Follow the same steps C, D, and E to design an "final decision" checker that maps from the outputs of the most-significant-bit (MSB) bit-slice to an answer: 1 = power of two, 0 = not a power of two.

3. Hardware Multiplier Design
Design a logic circuit that multiplies two 4-bit *unsigned* integers. You should use full adders as well as basic AND, OR, NOR gates as primitive circuit elements. If you wish, you can diagram a 4-bit adder unit out of full adders, and then use 4-bit adder blocks for the multiplier. You may also define and diagram other functional blocks, and then use them as part of your larger design. (Hints: think bit-slice! One good solution works just as you would do multi-digit multiplication by hand ... shift and add. Finally, think about how to easily implement the simple operation of a one-bit multiply $a_i b_j$. Note: the standard solution being hinted at is called an "array multiplier", but spend significant time trying to figure it out before looking it up.)

4. Software unsigned integer multiply
Back in the day (that is, when Professor Jones was a student) few microprocessors had built-in multiply hardware, because even a modest 16-bit by 16-bit hardware array multiplier requires several thousand transistors, which was a LOT back then. Even today, many low-power, inexpensive microcontrollers (including some members of Texas Instruments' MSP430 family, which Prof. Jones's graduate students use in their ultra-low-power system design research) lack hardware multipliers. If an integer multiply is required, it must be implemented in software using a series of shifts and additions, similar to how you would perform the multiplication by hand. The C code `multiply.c`, soon to be found on the course website or in the directory `/class/ece199/hw7`, implements *part* of the functionality required to perform a 16-bit by 16-bit unsigned integer multiplication via shift-and-add. Complete this program so that it works correctly. Test it by entering several values for which you know the correct answer. Turn in your completed code as your answer.