University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 150 R. H. Katz
Spring 2007

Problem Set # 10 (Assigned 10 April, Due 20 April)
This is the last one!

1. Traditional arithmetic circuits operate on their input bits all at the same time, calculating their output bits in parallel. Consider an alternative way to implement such operations using a so-called bit serial approach. The two numbers to be processed are presented to a Finite State Machine one bit at a time, with the lowest order bits presented first. The Finite State Machine produces at its output the lowest order bit of the result, then the next higher order result bit, and so on, until all of the input bits have been processed and all of the output result bits generated.
   (a) Design a (very) simple datapath for a bit-serial twos complement subtractor at the gate/component level (AND, OR, NOT, D-FFs, etc.) and identify the interface between your control state machine and your datapath. Remember that you have to deal with carry-in and carry-out (HINT: Remember that in twos complement, A minus B is the same as A plus ~B plus 1, where ~B is the ones complement of B).
   (b) Show your state diagram for a 4-bit bit-serial subtractor, where the outputs of the state machine are the control signals of the datapath you designed in (a).
   (c) Demonstrate how your subsystem works by showing step-by-step how it executes 0111 (7) minus 0011 (3) . The carry-in to the low order bit is initially zero.

2. A priority encoder circuit has n-inputs, numbered 0 to n-1, and n corresponding outputs, numbered in the same way. The output associated with the highest numbered input that is one will be set to 1, and all other outputs will be zero. Note that if no input is one, then all of the outputs will be zero. Implement a 4-input priority encoder circuit using logic constants 0, 1, inverters, and steering logic (pass transistors) only. (HINT: Think in terms of simple pass transistor circuit for stage i that directs a 0 to output i if a higher input is a one, otherwise passing input i directly through to output i).