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

EECS 150 R. H. Katz
Spring 2001

Problem Set # 8/9 (Assigned 22 March, Due 13 April)

1. Last semester's Midterm II introduced the following problem: design a FSM that continuously outputs a 1 as long as there has been a (non-zero) even number of ones in the input stream since the last reset. It stops outputting one when the machine has detected the sequence 0101 in the input stream. Implement this state machine with a more sophistication datapath: a shift register for the state register and a comparator to detect the critical sequences.
   (a) Design a datapath down to the block diagram level of its functional units to support this state machine. Define your register transfer operations, control signals, bus architecture, and timing behavior.
   (b) Design the state machine that implements the behavior of the specification above in terms of how it orchestrates the execution of the datapath you designed in part (a).
2. On your Midterm II Exam, Question 6, you were given a single read port/single write port register file as part of the problem. In that specification, the RD was asynchronous, and the WR was synchronous. Consider the following questions and extensions of this register file.
   (a) Using building blocks such as tri-state gates, decoders, multiplexers, and standard D flip-flops, show how to implement such a register file assuming it contains four words of four data bits each. Recall that the inputs are RD, WR, REG ADDRESS[1:0], DATA IN[3:0], and CLOCK, and the output is DATA OUT[3:0]. Design a single bit in detail, show how it is iterated to form a 4-bit word, and show how four words form the full fledged register file.
   (b) Analyze the timing of your design for 2(a). Make some assumptions about the propagation delays of your components and the set-up and hold constraints of your flip-flops (invent some numbers in nanoseconds, remembering constraints like FF prop delays exceed hold times—your textbook has some indicative delay numbers for older technology). Identify your critical propagation timing path for reading and writing, and determine the minimum clock period/maximum clock rate that your register file can support.
   (c) Modify your design in 2(a) so it can read two registers at the same time. Just design those parts that are different from what you did above. What changes are necessary to the inputs and outputs, and how does this affect your max clock rate and why?
3. A barrel shifter is circuit that allows its input to be shifted any number of positions. For example, a 4-bit rotating barrel shifter can shift its inputs I_3, I_2, I_1, I_0, zero, one, two, or three positions to the right as indicated by the shift control inputs S_1, S_0. The outputs become:

S1	S0	O3	O2	O1	O3
0	0	I3	I2	I1	I0
0	1	I0	I3	I2	I1
1	0	I1	I0	I3	I2
1	1	I2	I1	I0	I3

   (a) Using conventional multiplexers, decoders, and logic gates, show how to implement the barrel shifter as a purely combinational logic circuit.
   (b) Take a different approach by implementing the barrel shifter as a Finite State Machine plus simple datapath. Define the control signals between your controller and datapath, and show your detailed state machine for implementing the correct behavior of the barrel shifter as indicated above.
   (c) Compare and contrast your answers to 3(a) and 3(b). Which is faster? Which utilizes more hardware? Which is the better design and under what circumstances?

4. Design a 4-bit adder with carry-in and carry-out, and an extra input called Mode. When Mode = 0, the adder is a standard binary adder. When Mode = 1, it becomes a Binary Coded Decimal (BCD) adder. Show how to implement such an adder down to the gate level. You can use hierarchy to simplify your design, i.e., if every adder bit slice is the same, design it to the gate level once, and simply use a block diagram appropriately labeled to represent a reuse of the design.
5. The traditional binary adder operates on all of its input bits at the same time, calculating the sum output bits in parallel. Consider an alternative way to implement a binary adder using a so-called bit serial approach. The two numbers to be added 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 sum, then the next higher order sum bit, and so on, until all of the input bits have been processed and all of the output sum bits generated.
   (a) Design a simple datapath for the bit-serial adder down to the gate level, and identify the interface between your control finite state machine and the datapath. Consider how to deal with carry-in and carry-out in your design.
   (b) Show your state diagram for a 4-bit bit-serial adder, where the outputs of the state machine are the control signals of the datapath you designed in 5(a).
   (c) Demonstrate how your subsystem works by showing step-by-step how it executes the summation of 0110 and 1100. The carry-in to the low order bit is initially zero.