ECE199JL: Introduction to Computer Engineering **Fall 2012**
Homework 10 **Due: at start of lecture on Friday 2 November**

Logic Design with Components: Muxes, Decoders, and Memory

Please do problems 3.24, 3.26, 3.32, 3.34, and 3.35 from the textbook. When answering 3.32, be sure to understand the difference and explain in your own words rather than finding and copying the definitions from the text.

Here are four additional problems for this week:

1. Keyless Entry
Based on the keyless entry system design in Notes Set 3.1, derive Boolean logic expressions for the next-state values S_1^+ and S_0^+ as well as the output values D, R, and A.

2. Extending Keyless Entry
Starting from the *original* design for the keyless entry system in Notes Set 3.1 (not the extended variants that we developed in class), extend the design to time out unlock actions by the user. In other words, if a user unlocks one or all doors, then starts talking to a friend, and subsequently walks away without locking the car, the car doors should lock themselves. To implement this behavior, you must add a timeout of T cycles from both the DRIVER and UNLOCKED states back to the LOCKED state.

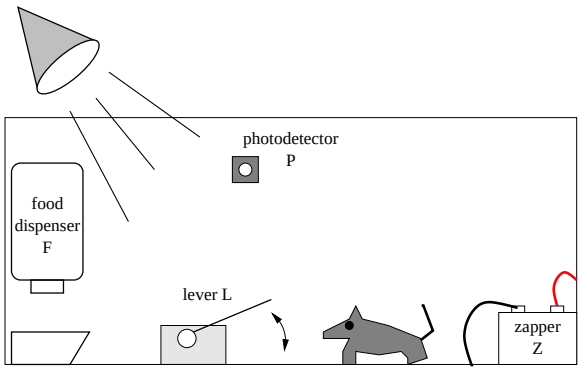
A. For the timeout, you should use a single binary counter. Explain the capabilities and behavior of the counter that you want to use. You need not draw the counter using gates, but you must explain its behavior, inputs, and outputs, and draw a block diagram with appropriate labels.

B. Explain how you will expand the DRIVER and UNLOCKED states to accommodate the timeout. Be clear about the source and input combination for any transition arcs entering, leaving, or passing between the new states (you will have to use ellipses—"..."—since we have not given you a value for T).

C. Implement and draw your design without changing the next-state logic corresponding to your answer to **Problem 1** above. In other words, the logic for S_1^+ and S_0^+ should be treated as black boxes, although you may need to insert muxes between these boxes and the flip-flop inputs.

D. Explain the purpose of any logic that you needed to add to the design in order to implement the extension (the counter, any muxes, logic into the select input to the mux and the data inputs to the muxes, and so forth).

3. Prof. Zapper's Rats
Pleased with your previous work and the outcome of the experiment, your part-time employer Professor Zapper from Psychology gives you a modest raise and asks you to design the logic for a slightly more complicated experiment to study the memory of rats.



The experimental set-up is shown to the right. In each "cycle," the experimenter turns the light on or off. Your system receives a signal from the photodetector P: when the light is on, $P = 1$. In the same "cycle" (long cycles), the rat responds by either depressing the lever L (in which case $L = 1$) or not depressing the lever ($L = 0$).

Professor Zapper wants the rats to follow a protocol. In the first and second cycles, the rat should match the lever with the light. In other words, the rat should press the lever when the light is on, and not press the lever when the light is off. In the third cycle, the rat should do the opposite: press the lever when the light is off, and don't press the lever when the light is on.

If the rat succeeds in this endeavor, it should receive food (set $F = 1$ for a cycle). If it fails, it should immediately be zapped for a cycle (set $Z = 1$ for a cycle). After the penalty/reward cycle, start over.

A. Based on this description, design a finite state machine to implement the desired experimental protocol. In particular, draw a complete state diagram labeled with outputs, internal state bits (you need to choose a representation), and input bit combinations on each transition arc. *Hint: you may want to simplify the inputs before your state machine sees them. If you do, explain how.*

B. Find simple logical expressions for next-state variables as well as functions mapping your internal state values to the outputs F and Z.

C. Draw a logic schematic circuit diagram for your system.

4. Arrays and Addresses in C

An array is a group of values with a common type indexed numerically. For example, when we write the bit values in an N-bit number A as $A_{N-1}A_{N-2}\ldots A_0$, you can think of these values as an array of N bits indexed from 0 to $(N-1)$.

Download the C program `layout.c` from the course web site on to a lab machine and compile it. This program will help you to understand how C arrays are laid out in a computer's memory. Memory on the lab machines is byte-addressable. In other words, the addressability of the memory is eight bits. Storing a `char` thus requires one memory address, but storing an `int` (32-bit 2's complement) requires four addresses.

A. Look at the code. Note that `first` is an array of type `int`, while `second` is an array of type `char`. Each has ten elements, but the elements occupy different amounts of memory. Run the program a couple of times and write down the starting address for both `first` and `second`. On most modern systems, these will be different each time you run the program; they are randomized to improve security.

B. Use the program to calculate an equation for the address at which the N^{th} element of an array is stored based on the following variables: the start of the array, the size of each element in bytes, and the index N. Note that the program uses hexadecimal to output memory addresses.

C. Notice that the calculated difference between two memory addresses (the last two columns in the table) does not produce the same result as you might obtain by performing the subtraction by hand. Explain what the software is doing to obtain the answers that you see in the last two columns.

D. The last lines of output from the program show the addresses of two non-existent elements of one array. They are an example of the computer doing what it was told, even though what it was told makes no sense. Do the addresses match your equation from **Part C**?