How Computers Work

Problem Set 3

Problem 1:

Finite State Machine Design

Suppose we wish to measure the traffic at a point on the MBTA track, counting the axle crossings in each direction. We set up a two light beams just above the rails and place two photocells A and B some inches apart along the track. When the beam shines on a photocell it produces a 0, and when the beam is interrupted, it produces a 1. Thus, when an axle crosses straight through from left to right, we read the following signals from A and B:

```
00 -> 10 -> 11 -> 01 -> 00
```

(and the reverse for the opposite direction). Unfortunately, trains my stop with axles part way through, and due to driver's sometimes reversing the trains, an axle may turn back after going part way through.

We wish to connect a synchronous finite-state machine taking its two inputs from A and B and producing two outputs X and Y such that:

- normally, both outputs are 0;
- when an axle has crossed completely from left to right, X becomes 1 for exactly one clock period;
- when an axle has crossed completely from right to left, Y becomes 1 for exactly one clock period.

(The pulses generated by X and Y could drive two counters, for example.) Assume that the only things that interrupt the light beams are axles, and that the clock is fast enough that we do not miss any transitions. We decide to design a Moore machine.

A. Draw a state-transition diagram for this FSM. Clearly label the inputs and outputs.

B. Construct the state-transition table for your FSM.

C. Show an implementation for this FSM using D flip-flops and a ROM.

D. Suppose we need to notify the maintenance department periodically about wear and tear on the track, and further that X and Y drive another FSM with one output that goes from 0 to 1 after 100,000 axle crossings (total, both directions). How many states must that FSM have? How many flip-flops would be needed to implement that FSM?

E. Consider the line above:

> *Assume that the only things that interrupt the light beams are axles, and that the clock is fast enough that we do not miss any transitions.*

What's wrong with this assumption? What else do you need to assume if your FSM is to work properly?

F. Let's fix it by throwing one flip-flop on each of the inputs. What do you have to worry about now?

Problem 2:

Another finite-state machine.

Design a finite-state machine that takes two unsigned binary numbers A and B in serial form, most significant bit first, and produces two binary outputs GE and LT. If A is greater than or equal to B, then A appears on the GE line and B appears on the the LT line. If B is greater than or equal to A, then B appears on the GE line and A appears on the the LT line.

Problem 3:

Weighing in

Ben Bitdiddle wasn't paying attention in lecture, and proudly shows you a device that can tell which of two weights is heavier than the other. When you look inside, you notice two digital scales wired to some circuitry. Ben notices your interest and shows you how it works: when you put the weights on the scales, the digital display lights up with the weight. Ben has connected each of the little LEDs from the scales to a bit of logic that determines which number is bigger. If the numbers are the same, then the device picks the weight on the left scale.

Ben points to the definition of a "perfect" arbiter (from the lecture, slightly edited):

> For some finite W_{margin} and T_{pd},
> produce the correct answer if $|W_a - W_b| > W_{margin}$,
> or any answer if $|W_a - W_b| <= W_{margin}$,
> but produce *some* stable answer within time T_{pd}.

Ben notes: "My T_{pd} is just the time of the digital scales, plus a bit for my logic, and my W_{margin} is just the accuracy of the digital scales. Therefore, this is a "perfect" arbiter! Ha ha! I'm gonna make a million bucks".

You notice Alyssa P. Hacker over in the corner, shaking her head and trying to suppress a smirk. What has she noticed?

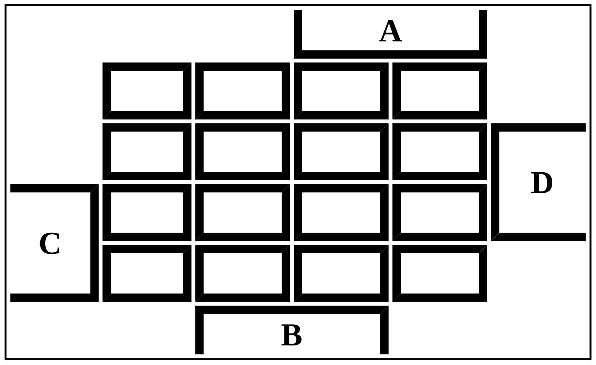
Problem 4:

Hazards

For the following logical expression:

$\overline{B}.\overline{D} + \overline{C}.D + B.D + A.B.C$

(i) draw the Karnaugh map, in the form:



(ii) Mark the areas represented by the terms in the expression above.

(iii) Implement the expression directly in Sum-of-Products, using invertors, AND gates and OR gates.

(iv) Indicate any potential hazards on the Karnaugh map (assuming only one-bit-at-a-time transitions); show on the Karnaugh map how they may be eliminated; and revise the gate-level logic diagram accordingly.