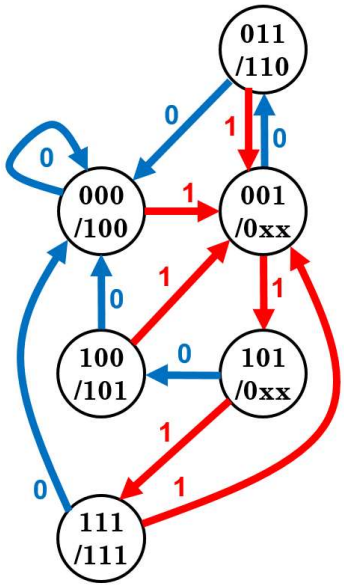
ECE 120: Introduction to Computing Spring 2019 ZJUI

due at start of section on Friday 26 April 2019

Homework 9: Building and Extending Finite State Machines

1. Decompression

Prof. Lumetta has designed an FSM to aid in a decompression task. Starting from the state transition diagram shown below, with states labeled with state representation and outputs as $S_2S_1S_0$/ABC and transition arcs labeled with input G, you must develop an implementation by following the steps below.



a. Draw a K-map for each output (A, B, and C) and for each next-state variable (S_2^+, S_1^+, and S_0^+).

b. Use the K-maps from **part (a)** to find expressions for each output and next-state variable with minimal area. (**Minimize each variable independently—do not try to share gates for implementations.**) You must consider both minimal SOP and minimal POS solutions, but you should only circle the better of the two choices (SOP or POS) on your K-maps and write the corresponding expressions when handing in your homework.

c. Draw a circuit implementing the FSM. Put three flip-flips in the middle of your circuit, labeled S_2, S_1, and S_0. Draw next-state logic to the left of the flip-flops. Each next-state variable should be produced by separate gates. Label the inputs to these gates with the current values available from the flip flops (S_2, S_1, and S_0, and their complements), or with G or G'. Draw the output logic to the right of the flip-flops, again using separate gates for each output variable. Inputs to the output logic should come directly from the flip-flops. **Draw your circuit carefully and legibly.**

2. Counter Design

In this problem, you will design a counter using two approaches, then compare the size of the two designs in terms of area. The counter must produce the 4-bit output $Q=Q_3Q_2Q_1Q_0$ in the sequence 0000, 0001, 0011, 0010, 0100, 1100, 1000, and then back to 0000. This sequence can be used to produce a drink with four layers in which the first two and the last two are mingled, but the middle two are separated. As with all counters, the sequence should repeat infinitely. The counter's internal state bits should be denoted by S_i.

a. How many bits are needed to represent the state of your counter?
b. Using the minimum number of flip-flops and the patterns 0 through 6 (as unsigned) to represent the states starting with output 0000, write a truth table including each output variable ($Q_3Q_2Q_1Q_0$) and each next state variable (S_i^+) in terms of the current counter state. Note that some rows of your truth tables will produce x's (don't cares).
c. Copy the truth table from **part (b)** into separate K-maps (one for each Q_i and one for each next-state variable).
d. Use the K-maps from **part (c)** to find expressions for each output and next-state variable with minimal area. You must consider both minimal SOP and minimal POS solutions, but you need only circle the better of the two choices (SOP or POS) on your K-maps and write the corresponding expressions when handing in your homework.
e. Now use the expressions from **part (d)** to calculate the value of the area heuristic for each of the outputs and for each next-state variable. Notice that all state variables are available from the flip-flops in both complemented and uncomplemented form, so you should not count inverters associated with literals (for example, both S_0 and S_0' cost 1 area).
f. Sum the values from **part (e)**, then add 18 for each flip-flop needed by your design, to find the total area for the design.
g. Repeat **parts (b)** through **(f)** using the 4-bit output values to represent the six states of the counter. In this case, the output logic will consist only of wires.
h. Compare the area of the two designs. Which one is better?
i. Finally, for each of the two counter designs, and for each value of state bits not included in the desired counter loop, find the next state. Do all of these states eventually converge to the desired 7-state loop? (See Sections 3.2.4 and 3.2.5 of the notes if you are not clear on why this question is asked of you.)

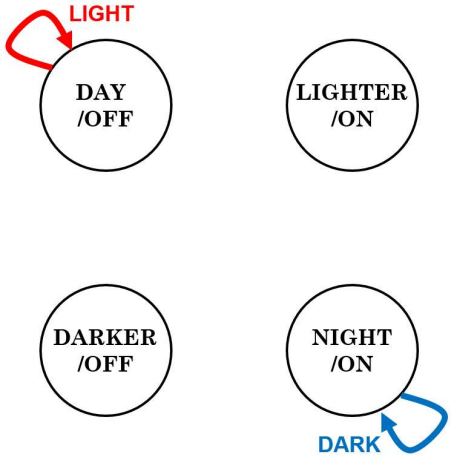
Note that your designs may give you some opportunity for reuse of logic gates. For example, one gate might produce a signal needed both for an output and for a next-state variable. Technically, you should only count such a gate's area once, but you need not be so careful in this problem. Just make your decisions about how to count and your results clear and easy to read and follow.

3. Headlight Control

Your aid is needed in developing the next-generation automatic headlight control system for Fjord Motors. You must design an FSM that monitors the current ambient light level, provided to your FSM as a 2-bit unsigned number L, and automatically turns the headlights on/off through an output H (H=1 means the headlights are on).

Your design should turn the headlights off whenever the light level L has been 2 or more for two consecutive cycles, and should turn the headlights on whenever the light level L has been 1 or less for two consecutive cycles (remember than L is a 2-bit unsigned number, hence ranges from 0 to 3).

a. Complete the abstract state transition diagram below by adding transitions labeled as DARK and LIGHT from each state. These labels correspond to $L < 2$ and $L \geq 2$, respectively.



b. Now choose a representation for the states. The "DAY" state should use $S_1S_0 = 00$. If possible, choose bit patterns such that all transitions change only one state bit rather than changing both state bits.

c. Write a truth table for the output H in terms of the current state S_1S_0, and a next-state table for the next state bits S_1^+ and S_0^+ in terms of the ambient light level $L = L_1L_0$ and the current state S_1S_0.

d. Copy the truth table from **part (c)** into separate K-maps (one for H and one for each next-state variable).

e. Use the K-maps from **part (d)** to find expressions for each output and next-state variable with minimal area. (**Minimize each variable independently—do not try to share gates for implementations.**) You must consider both minimal SOP and minimal POS solutions, but you should only circle the better of the two choices (SOP or POS) on your K-maps and write the corresponding expressions when handing in your homework.

f. For the luxury model of Fjord's new vehicle, the driver has a control knob to adjust the automatic headlights' light-level threshold. This knob produces an 8-bit unsigned number T. The light sensor L has also been upgraded to produce an 8-bit unsigned value (instead of a 2-bit value). The values L and T are fed into an 8-bit unsigned comparator that produces a signal X whenever $L < T$. In other words, X=1 when $L < T$, and X=0 when $L \geq T$. Explain how to integrate your FSM design with the comparator for the luxury version of the vehicle. *Hint: You should not need to change your design's structure.*

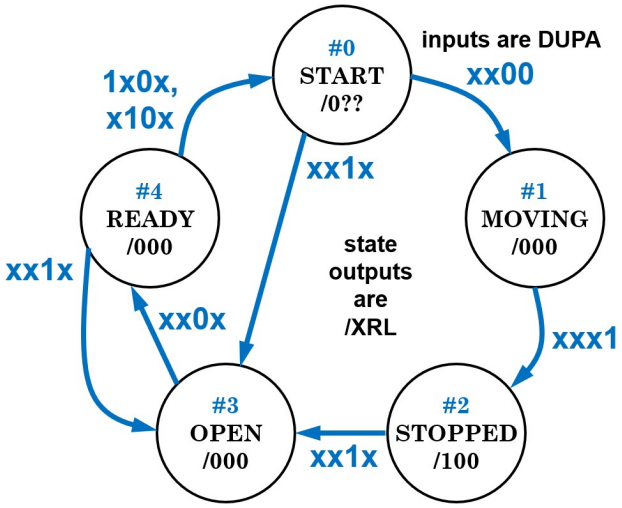
4. The Elevator Problem

Given your midterm solutions, Prof. Lumetta is now ready to have you design the FSM to control an elevator. Here's how it will work:

Inputs (all active high):
D – the elevator needs to go down (a button is pressed for a floor below the current floor)
U – the elevator needs to go up (a button is pressed for a floor above the current floor)
P – the elevator door is open
A – the elevator is stable at a floor (rather than moving between floors)

Outputs (all active high):
X – try to open the elevator door (it may open/close for other reasons)
R – make the elevator move upward
L – make the elevator move downward

The state machine should operate as shown in the figure to the right. Self-loops are not shown in the figure: instead, every input combination for which a state has no arc leaving the state is a self-loop.



For the states shown, your state machine should use a one-hot encoding with bit numbers given by the numbers above the state names. For example, S_2 should be 1 iff the elevator is in the STOPPED state, and S_4 should be 1 iff the elevator is in the READY state.

The elevator also needs one more bit of state, which you should call T. The T bit is 1 when the elevator has been most recently moved upward, and 0 when the elevator has most recently moved downward. The T bit can only change when moving from READY to START, and should only change when the elevator does not need to move in the same direction as it did previously. In other words, if T=1, the elevator last moved upwards, and it should continue to move upward if moving upward is necessary (if U=1). Only when U=0 (and D=1) should your FSM start moving the elevator downward (by changing T to 0 and moving to START).

Note that, for the START state, the outputs R and L are not given in the state diagram. They depend on T, of course: the FSM should make the elevator move upward when T=1 and downward when T=0.

What's your job?

Write the output equations and next-state equations in minimal SOP form. That's all.

5. Alice and Bob, the Security Experts

Update your Subversion repository to obtain the **`hw9`** subdirectory, in which you will find the FSM-based game **`alice-and-bob.c`**. Compile it and play it to get a feeling for how it works.

a. Each of the rooms in the shared apartment in the game has a unique value for the state variable **`room_number`**. Draw a state transition diagram for the physical configuration of the shared apartment. In particular, create one node for each possible value of the state variable **`room_number`**, and label each node with the corresponding room number. Draw transition arcs labeled with choice numbers to connect rooms to one another. Note that for this part, **you should only draw transition arcs for transitions that change the value of `room_number`**. You may either read the code or play the game to determine the labels used in the FSM implemented by the game. Note that these labels do not depend on either of the other state variables defined by the program.

b. The diagram that you drew for **part (a)** captures the high-level configuration of the FSM states. Now you must draw a more detailed version. Start by making three additional copies of your diagram from **part (a)**. These three copies correspond to the following combinations of variables (mark the three copies as FIRST, SECOND, and THIRD, as indicated below):

- FIRST: `0 == have_password` and `0 == hacked_router`
- SECOND: `1 == have_password` and `0 == hacked_router`
- THIRD: `1 == have_password` and `1 == hacked_router`

Also add a single additional state corresponding to the condition `1 == game_won`. Mark this state with the label **END**.

Now add transition arcs as appropriate between the three copies, using the choices required in the game to follow these transitions.

Finally, clearly mark the starting state of the game's FSM with the label **START**.

Then go watch a movie.