# ASSIGNMENT 3: State Machines

In this assignment, you will design and implement a state machine that acts as the controller for the line-following robot presented in class.

**You will have to demonstrate a working solution to the professor or TA. Please check the class web site for office hours and plan ahead!**

The steps are:

1. Draw the State Transition Diagram showing each state, the outputs produced in each state, and the transition rules between states. This is already done (see below).

2. From the State Transition Diagram, create a State Transition Table. The Table maps from inputs (current state bits and external input bits) to outputs (next state bits and control output bits). This too is already done (see below).

3. From the State Transition Table, create Logic Equations that describe the Table's output bits as a function of its input bits.

4. Minimize these Logic Equations.

5. Implement the simplified equations in discrete logic, using the chips provided in your kit.

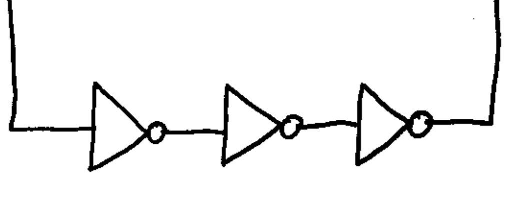
6. Build and test the resulting solution, using a latch chip to store the state bits and an external clock to run the machine.

7. When the state machine appears to be working, connect it to the robot and see if it can follow the line!

We will start this problem set by building an oscillator and dividing this frequency down to a point where it can be used in our state machine.

## PROBLEM 3-1. INVERTER CHAIN AS OSCILLATOR.

The following circuit will produce an oscillator—a circuit that continually flips back and forth between 0 and 1:



*a) Explain why this circuit oscillates.*

*b) Based on the characteristic propagation delay of a 74HC04 inverter , what frequency of oscillation would you expect?*

*c)What frequency would we see with 5 inverters in the chain?*

*d)What would happen if we used 4 inverters?*

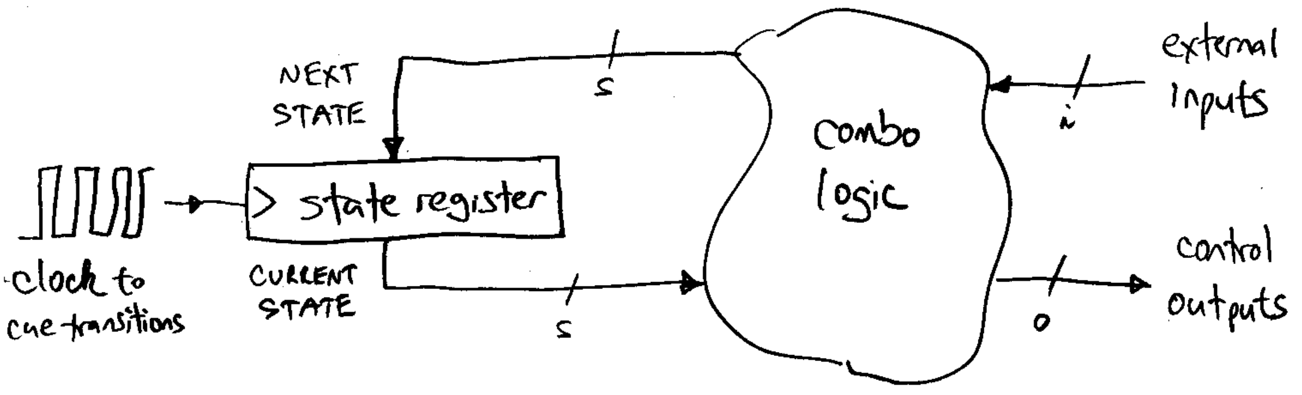
## PROBLEM 3-2. DIVIDING DOWN THE OSCILLATION.

Using 74HC04 inverters, build the oscillator shown above. Now, design a circuit using the 74HC393 dual counter to divide down the frequency of oscillation. Wire the 74HC393 into an 8-bit counter configuration (run the $Q_D$ output of one half into the clock input of the other half).

Run the inverter oscillator signal into the clock input of your 8-bit counter. Now probe the counter's Q outputs. You should see both the green and red LEDs on at the same time, indicating oscillation. **The probe circuit on your UML305DEV board has a maximum frequency limit of about 200 kHz, so you will only see this result from the most-divided-down outputs. You should see the oscillation on the highest (most-significant) output bit, which will have the lowest frequency.**

*Based on your calculation of the 3-inverter oscillator frequency, what frequency do you expect at the highest bit of your 8-bit counter?*

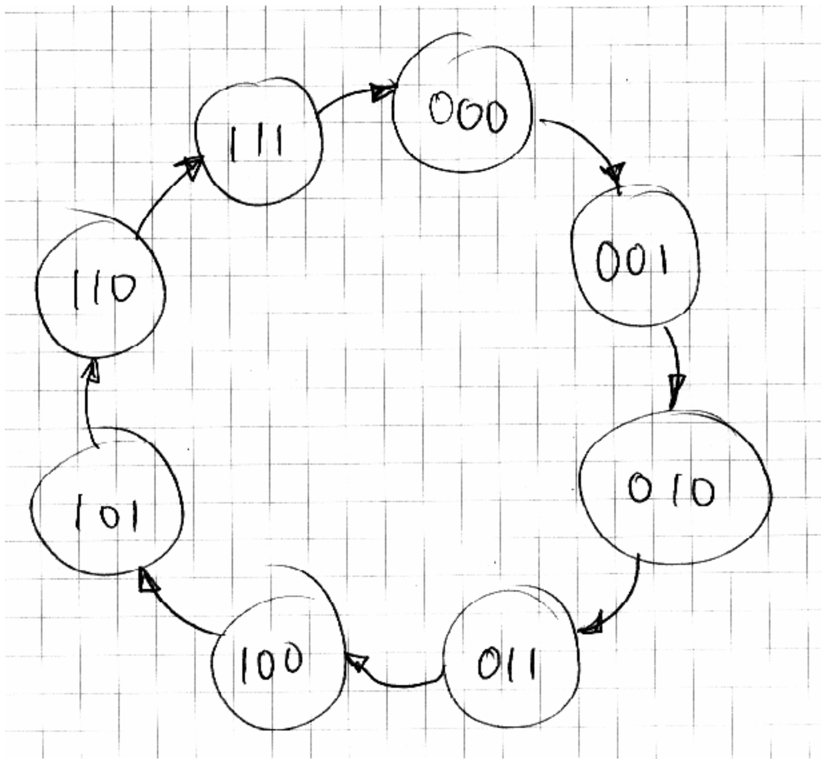
Now, let's review the conceptual implementation of the state machine:



The **state register** is a latch that holds bits that represent the current state. (There are *s* state bits.) **Combinational logic** creates a mapping from the current state plus external inputs to the new state, which is fed back into the state register. The combo logic also produces control outputs, as a function of the current state. Finally, an external clock signal continually strobes the state register's clock input. On each active edge of the clock, the state register captures the *next state* information from the combinational logic, thereby moving to a new state (or staying in the old one, depending on what signals are generated by the combo logic).

**PROBLEM 3-3. 3-Bit Counter as State Machine.**

It's also possible to build a counter from a state machine. Here's the state diagram of a 3-bit counter:



The following state table shows transitions from the "current state" ABC to the "new state" A'B'C':

| ABC | A'B'C' |
|---|---|
| 000 | 001 |
| 001 | 010 |
| 010 | 011 |
| 011 | 100 |
| 100 | 101 |
| 101 | 110 |
| 110 | 111 |
| 111 | 000 |

*For each new state A', B', and C', write the sum-of-products equation that represents it. For example:*

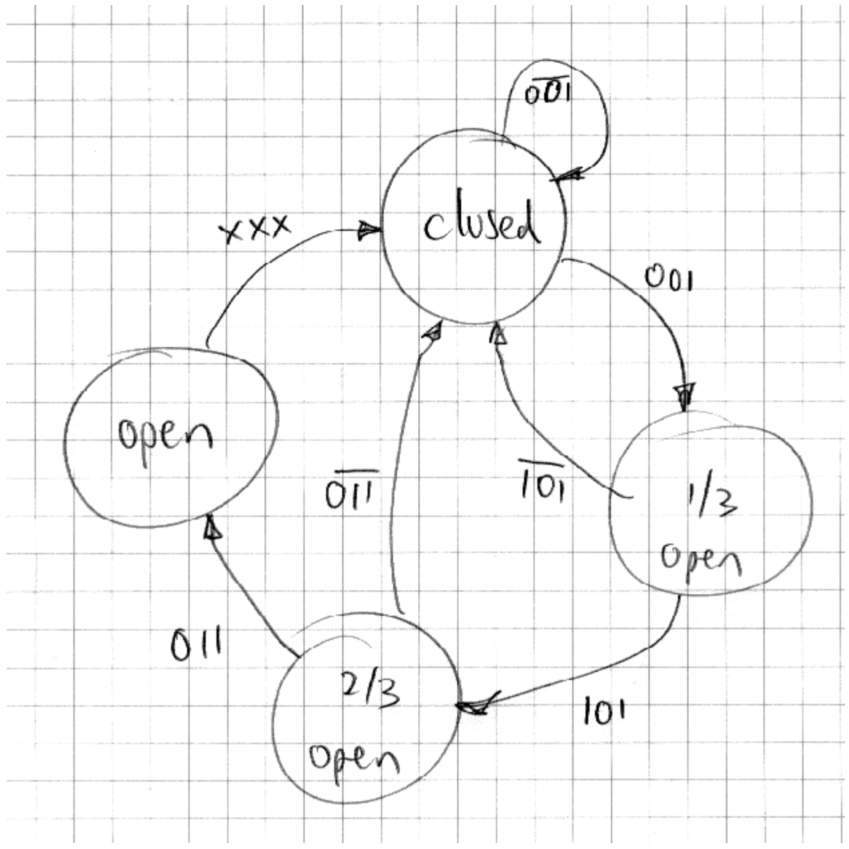
$A' = (\overline{A}\ B\ C) + (A\ \overline{B}\ \overline{C}) + (A\ \overline{B}\ C) + (A\ B\ \overline{C})$

*Write each equation (one each for A', B', and C'), and then reduce each to minimal terms. [Hint: the equation for A' above can still be minimized—collect the terms for A*(some expression of B and C).]*

*Which implementation do you think is better—chain of flip-flops, or state machine with combinational logic? Why?*

**PROBLEM 3-4: Combination Lock as State Machine.**

The figure below illustrates the state diagram for a three-number combination lock:



The lock has four states: closed, 1/3 open, 2/3 open, and open. It is opened using the correct series of three-bit numeric entries. When in the closed state, entering ‘001’ moves it to the 1/3 open state. From there, ‘101’ moves it to the 2/3 open state, but any other entry moves it back to closed. From 2/3 open, ‘011’ moves it to open (and any other entry moves back to closed). When in the open state, the lock should generate an output V which is true (which controls a solenoid that opens the lock). Then the lock automatically moves to the closed state.

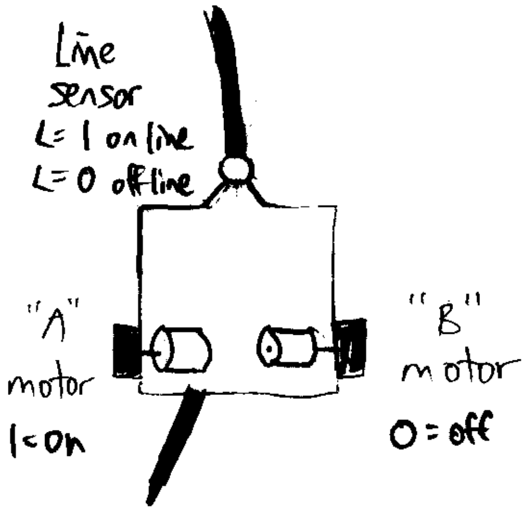
Your task is to create a circuit that implements this design. The steps are:

1. Assign two bits of state (A and B) to represent the four states of the lock.

2. Create a truth table that defines new state (A’ and B’) in terms of old state (A and B) and the input (X, Y, and Z, with X being the high bit). Also, make a truth table for output V (for vend) as a function of state bits A and B.

3. Write sum-of-products expressions for A’, B’, and V, and minimize them.

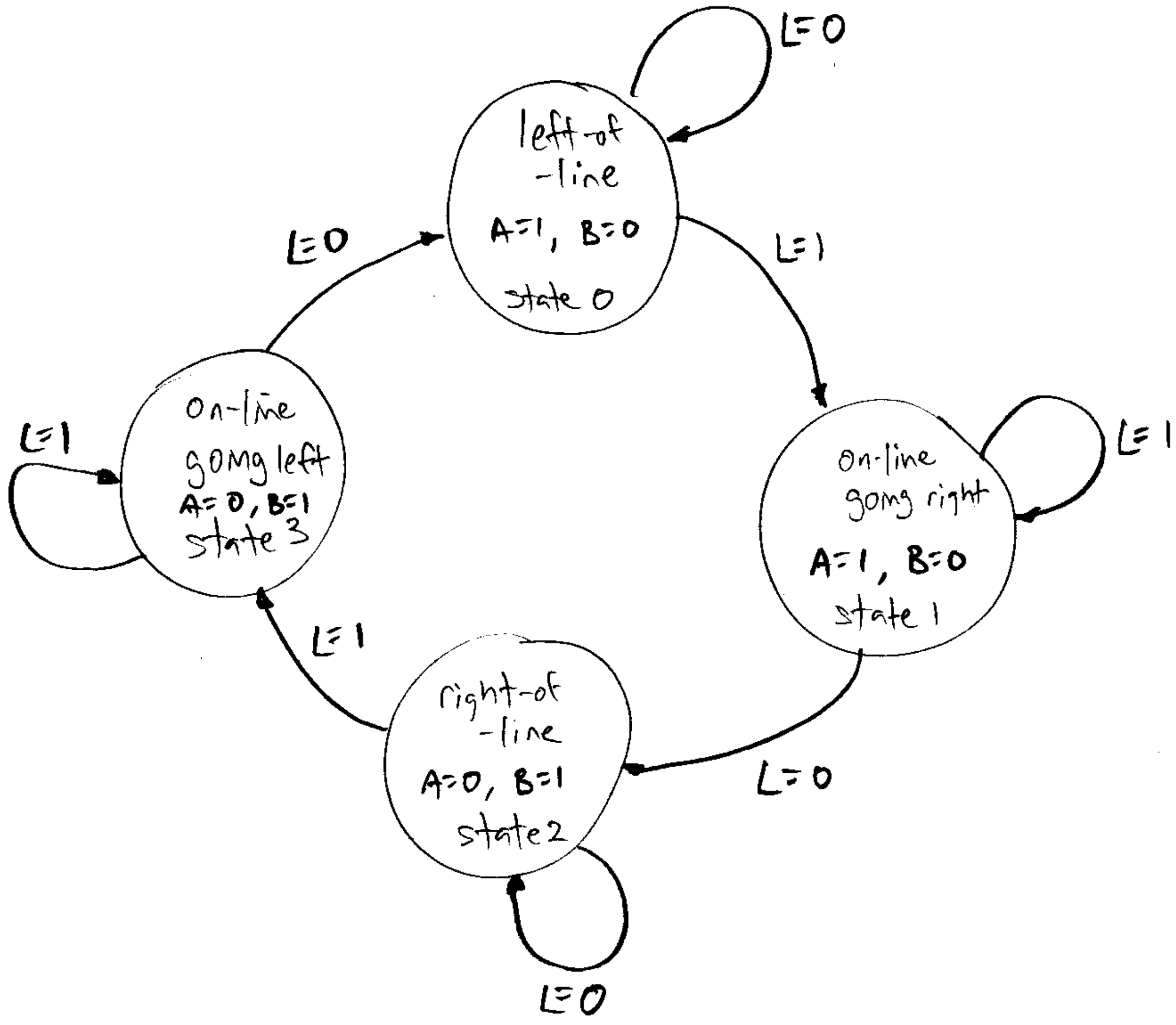
Let’s move on to the line-following robot problem. The robot we will use is depicted below:



The robot has two inputs, A and B, which control the left and right motors, respectively. If an input is high (1), then the motor is on. If the input is low (0), then the motor is off.

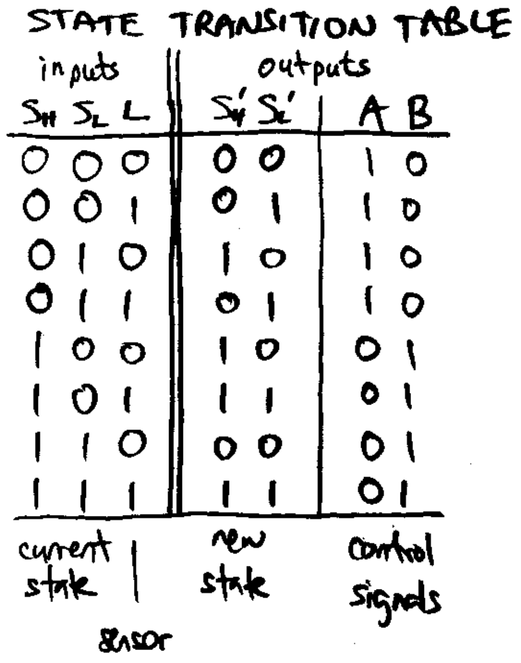
The robot produces one output, L, the line sensor. This is true (1) when the robot sees the line and low (0) when it does not.

As presented in class, a solution to getting the robot to follow the line is based on the following state machine:



There are four states, representing the robot being to the left of the line (state 0), on the line moving right (state 1), to the right of the line (state 2), and on the line moving left (state 3). The machine generates two outputs, A and B, which control the left and right side motor, respectively. The machine accepts one input, L, which is true when the robot is on the line.

From the State Transition Diagram, we can produce the following State Transition Table. In the table, we have encoded the states using two bits, $S_H$ and $S_L$, which denote the high and low bits of a 2-bit representation of the state (numbered 0 – 3).



STATE TRANSITION TABLE

| inputs | | | outputs | | | |
|---|---|---|---|---|---|---|
| $S_H$ | $S_L$ | L | $S_H'$ | $S_L'$ | A | B |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| current state | | sensor | new state | | control signals | |

The Transition Table shows that we are creating a mapping from three bits (the two state bits and the sensor input bit) to four bits (two next-state bits and two control output bits).

**PROBLEM 3-5. Using the sum-of-products technique, write an expression for each of the outputs ($S_H$', $S_L$', A, and B) as a function of the inputs ($S_H$, $S_L$, and L).**

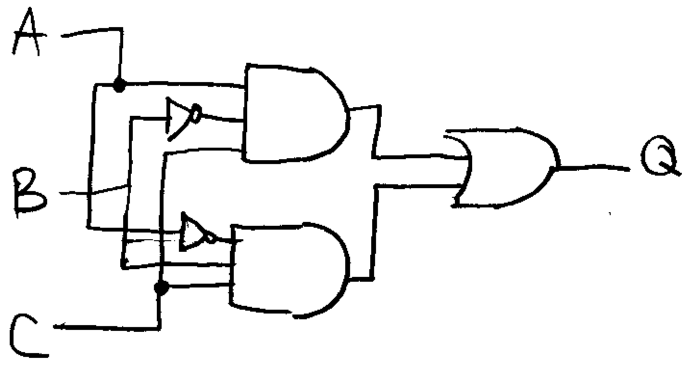
**PROBLEM 3-6. Using algebraic theorems, minimize each of these expressions.**

**PROBLEM 3-7. Using DeMorgan's Law, draw a logic diagram to implement each of these expressions. Remember that a sum-of-products can be replaced by a NAND-of-NANDs (see below). Make sure to only use gates that you have in your kit.**

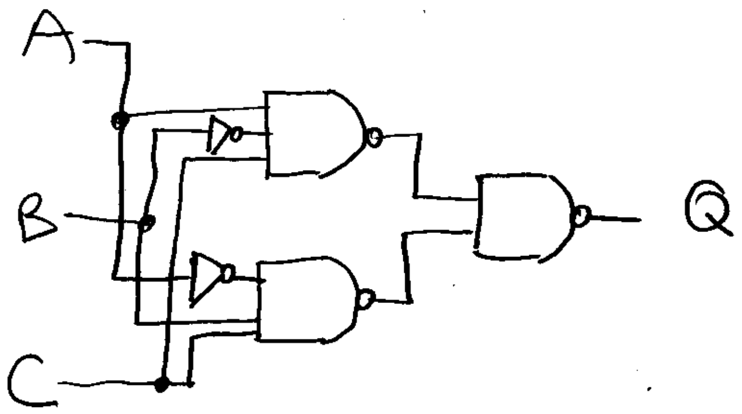
For example, suppose you were trying to implement the equation:

$Q = A\bar{B}C + \bar{A}BC$

This could be done with AND gates for the "ABC"-type expressions, followed by an OR gate to bring these together:



Or, by DeMorgan's theorem, you could NAND together the "ABC"-type expressions, and then NAND together the results:



**PROBLEM 3-8. Based on your logic diagram, draw a complete circuit for the state machine. Use the 74HC574 latch chip to hold the two state bits.**

**Get your design checked over by a TA or the professor before moving ahead, unless you are extremely confident.**

**PROBLEM 3-9. Implement the full state machine, using the divided oscillator of 3-2, a 74HC574 latch, and the combinational logic implementation of 3-5.**

Some construction hints:

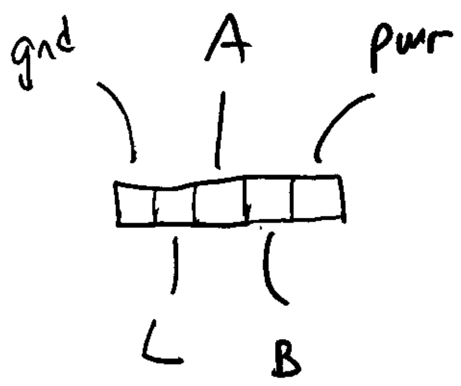
• plug a capacitor across the power and ground wires of the 74HC393 counter. This will smooth out the clock signal.

• wire status LED to the current state bits and motor outputs A and B, so you can see what's going on.

• test your design by wiring the line-sensor input L to a pushbutton. Then you should be able to see the state transitions by pressing and releasing the button.

**PROBLEM 3-10. Bring your implementation to the lab, connect it to the robot, and see if it works.**

There will be a ribbon cable to jumper from your board to the robot. You should make a panel of 5 connections, providing (in order) the signals: ground (0v), L, A, B, and power (+5v) as shown:



Wire an area on your breadboard to provide these signals, so it's an easy operation to connect your state machine (the "brain") to the robot.