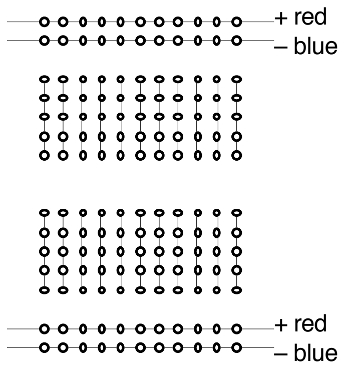
ASSIGNMENT 2b
due at the start of class, Wednesday Sept 25.

For each section of the assignment, the work that you are supposed to turn in is *indicated in italics* at the end of each problem or sub-problem. This result may be a drawing/schematic, a written answer, or an equation, or a combination of all three. You may prepare the result by hand or using a computer, but I want it turned in as hard copy.

2b-1: Basic Prototyping and the 74HC00 Quad NAND Gate.
In this exercise, you will learn basic prototyping skills using your UML305DEV board and the solderless breadboard.

The figure below illustrates a section of the solderless breadboard:

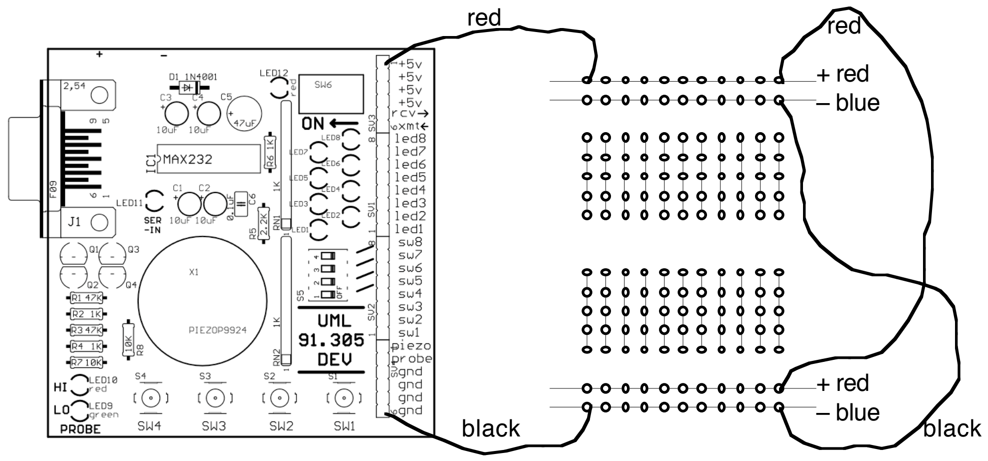


Each group of holes that is connected (in the drawing) with a thin line is connected electrically on the board. Thus, there are two sets of horizontal “busses,” along the top and bottom, and vertical groups of five holes. The horizontal busses are used for power and ground distribution, and the vertical holes are used to install and connect parts.

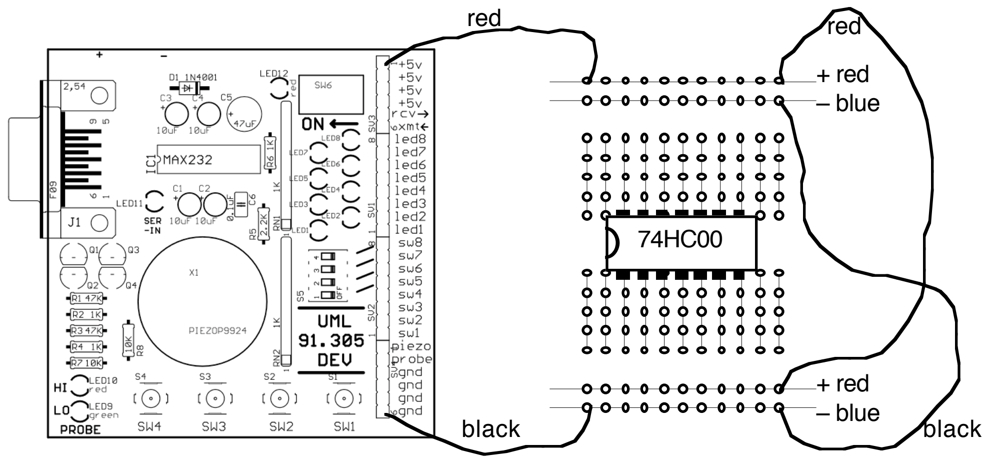
The busses are printed in red and blue, and labeled + and –. Start off by connecting power and ground from the UML305DEV board to one of your proto boards. Get a length of red wire, strip about 1/3 of an inch of insulation from both ends. Then run one end into one of the four +5v sockets on the UML305DEV board. Connect the other end to one of the red bus strips. Get a second red wire, and connect from one red bus to the other red bus.

Using a black wire, also connect ground from the UML305DEV board to one of the blue bus strips. Get a second black wire, and connect from the blue bus to the other blue bus.

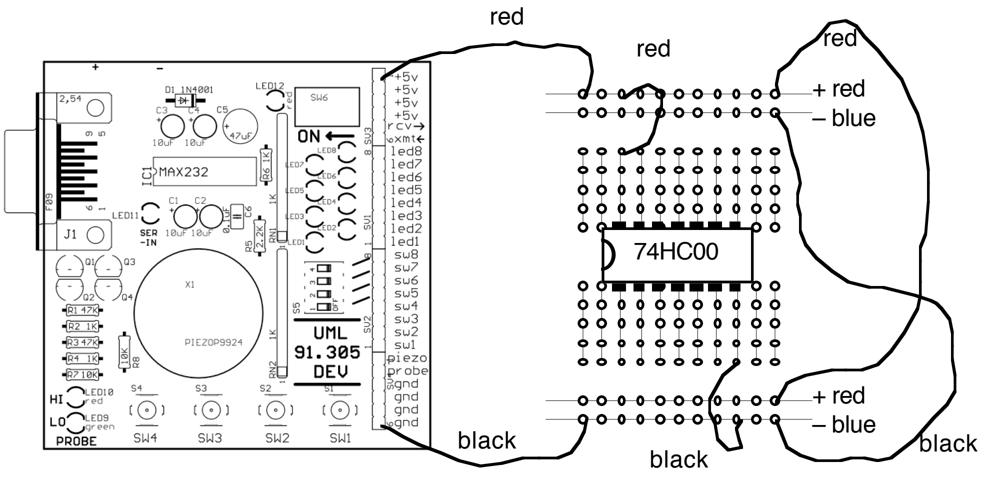
Your set up should now look like:



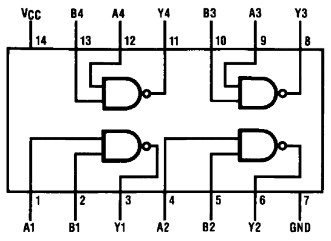
Now you're ready to install a chip and wire it up. Get the 74HC00 quad NAND gate chip, and plug it into the breadboard, straddling the vertical banks of pins. Arrange the chip so that the notch is to the left, putting pin 1 in the lower left corner:



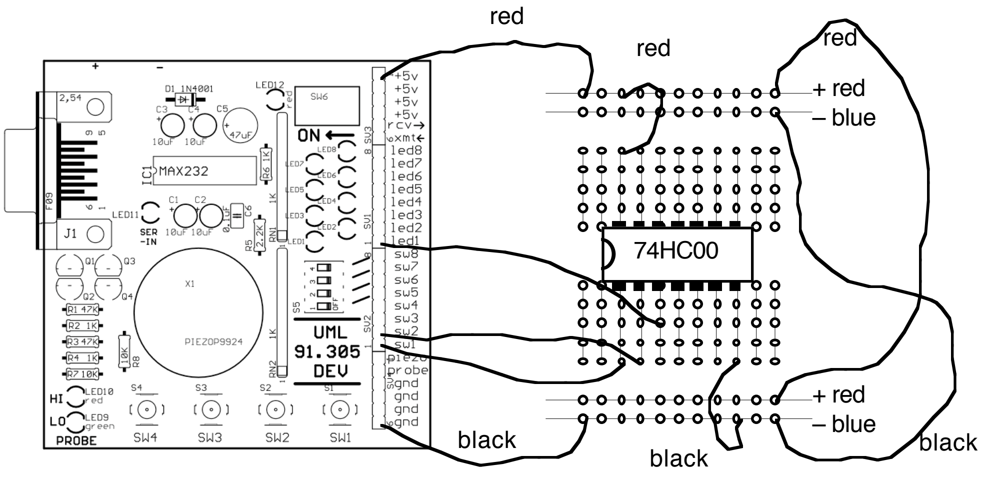
Now, connect power and ground to the chip by running wires to from the chip to the power busses. Pin 14 is the power pin; use a red wire to connect it to the red power bus. Pin 7 is the ground pin; use a black wire to connect it to the blue ground bus:



Now, let's connect to an actual gate from the chip. Looking at the data sheet, one can see that the gates are connected as illustrated:



We'll connect the #1 gate. The inputs A1 and B1 are at pins 1 and 2, and the output Y1 is at pin 3. Using white wires, connect the inputs to pushbuttons SW1 and SW2, and the output to LED1. The circuit should now look like:



At this point, turn on the UML305DEV board. Make sure the red power LED next to the power switch is on.

The pushbuttons generate logic low (0v) signals when they are not pressed. So, the input to the NAND gate is 0 0, and its output should be logic high (5v). Thus, LED1 should now be lit.

Test the NAND function: when both inputs are true (buttons pressed), the output goes low (LED off).

To turn in: nothing to turn in. This exercise is here for your benefit if you have not used solderless breadboards before.

2b-2: Counters.

With the assistance of the data sheet, hook up the 74HC393 dual 4-bit binary counter. Use a pushbutton to generate the clock input; when it's working, you should see it increment one count per button press.

Draw a circuit diagram of the chip and how you have attached it so that it will count.
Does it count on the rising edge or falling edge of the clock signal?

Using the chips in your kit, design a circuit that will generate a carry from the first counter stage to act as the clock input of the second, creating a full 8-bit counter. Do the counters increment on the rising edge or the falling edge of the clock signal? Make sure to think this through, so that the second counter increments on the same edge as the first.

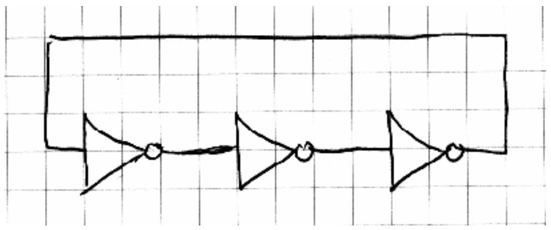
Draw a circuit diagram of your 8-bit counter.

Typically, counters are built with chains of flip-flops. Using your 74HC73 dual JK flip-flop chip, build a two-bit counter.

Draw a circuit diagram of your 2-bit counter.

2b-3: Feedback Loops with Inverters.

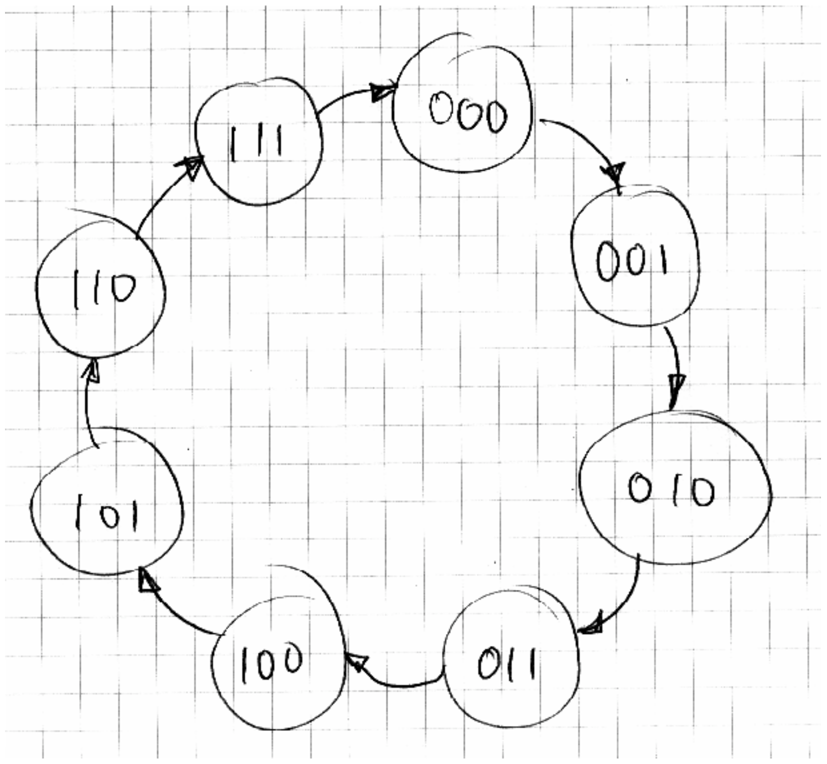
The following circuit creates an oscillator:



Explain why this circuit oscillates.
What frequency would you expect it to produce?

2b-4: 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:

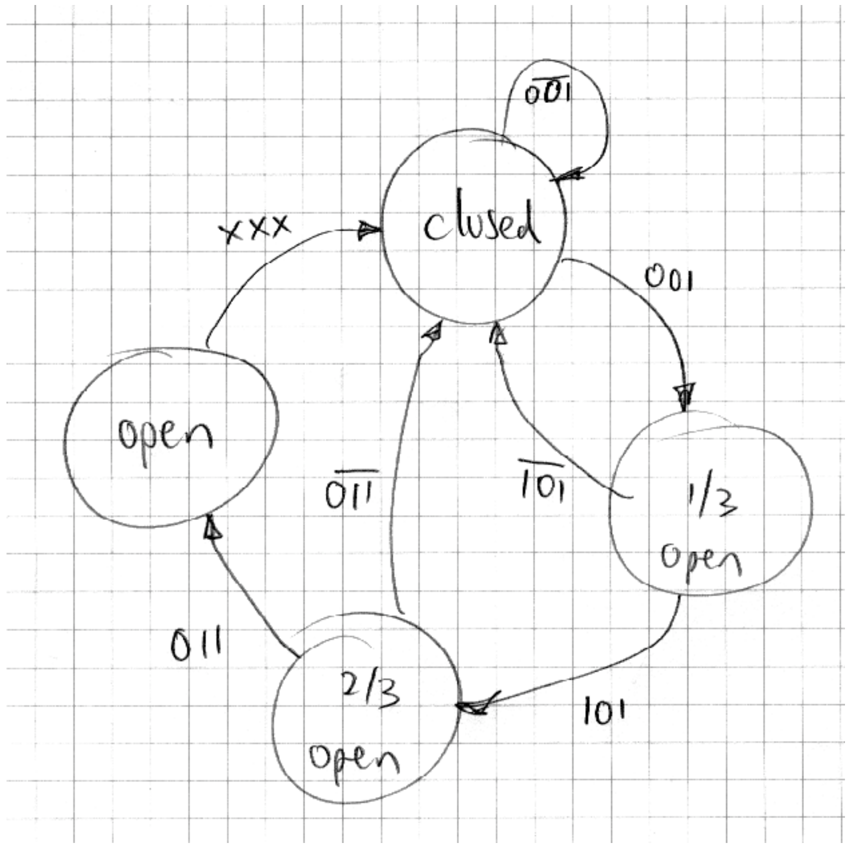
$A' = (\overline{A}\ B\ C) + (A\ \overline{B}\ \overline{C}) + (A\ \overline{B}\ C) + (A\ B\ \overline{C})$

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?

2b-5: 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.

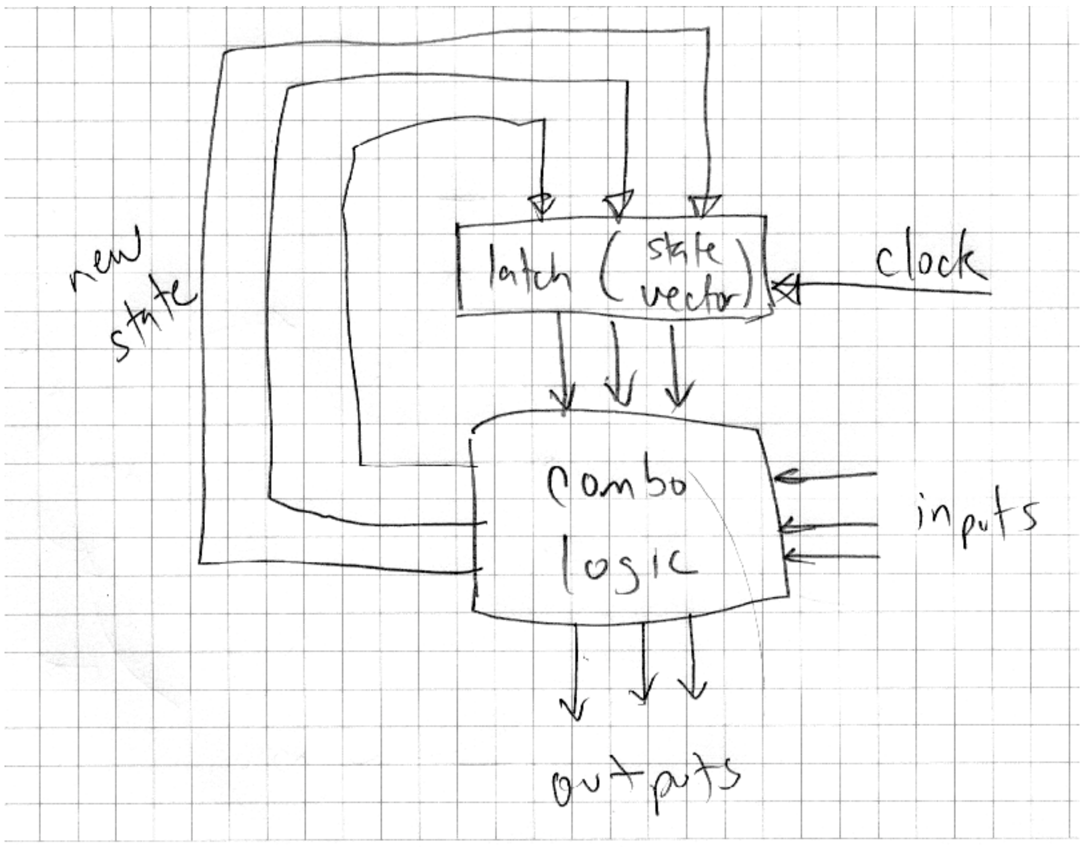
Your task is to create a circuit that implements this design. The steps are:

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.

4. Draw a circuit diagram for your implementation, using combinational logic and a 74HC574 latch. Remember the canonical form for a state machine:



5. Implement your design. Does it work? If there are any problems, solve them and show your process.

Turn in written notes, tables, expressions, and diagrams from steps 1 through 5 of this problem.

2b-6: Other State Machines (for Honors Students).

Think of two other devices or systems that can be implemented in terms of state machines like the ones here. Draw the state transition diagrams for them, and explain your design.