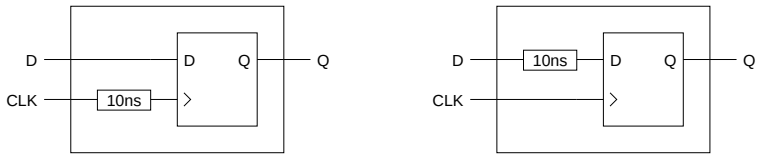
EE 121 June 4, 2002
Digital Design Laboratory Handout #34

Midterm Examination #2 Solutions

Open book, open notes. Time limit: 75 minutes

1. (20 points) *Setup and hold times.* The D flip-flops below have setup time $t_s = 18\,\text{ns}$ and hold time $t_h = 4\,\text{ns}$.



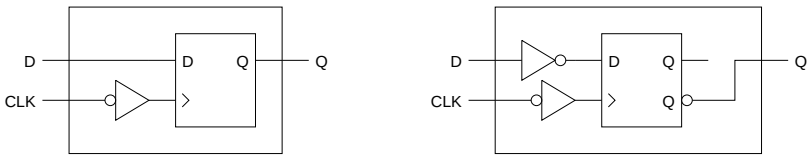
   a. Suppose the clock is delayed by exactly 10 ns. (See the left device in the figure above.) What are the setup and hold times for this modified flip-flop?

   When the clock is delayed, the data may be delayed by the same amount without violating the setup time. But the hold time is increased because the clock does not arrive until later.

   Setup time: $t_s = 18 - 10 = 8\,\text{ns}$ Hold time: $t_h = 4 + 10 = 14\,\text{ns}$

   b. Suppose the data input is delayed by exactly 10 ns. (See the right device in the figure above.) What are the setup and hold times for this modified flip-flop?

   Setup time: $t_s = 18 + 10 = 28\,\text{ns}$ Hold time: $t_h = 4 - 10 = -6\,\text{ns}$



   c. A negative-edge-triggered flip-flop can be built from a positive-edge-triggered flip-flop by inverting the clock input. (See the left device in the figure above.)

   Propagation delays in nanoseconds for the inverter are given in the following table.

	min	max
t_{LH}	12	22
t_{HL}	8	16

   Find the setup and hold times for the modified flip-flop.

   We must use minimum rising clock delay for setup time, maximum delay for hold time.

   Setup time: $t_s = 18 - 12 = 6\,\text{ns}$ Hold time: $t_h = 4 + 22 = 26\,\text{ns}$

   d. As you have discovered in parts (a) and (c), delaying the clock increases the hold time. To reduce the hold time, we might add delay to the data input, as shown in the right

device in the figure above. (Note that the output of the modified flip-flop must now be taken from the internal flip-flop's complemented output.) Find the setup and hold times for this negative-edge-triggered flip-flop.

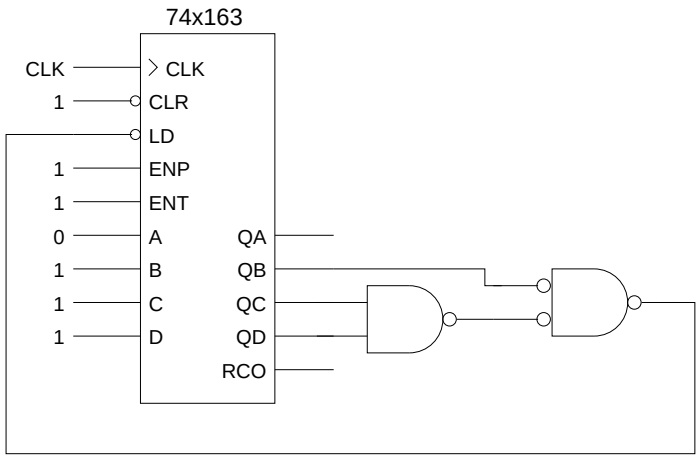
The data window has increased from 22 ns to 46 ns—not a good way to build a flip-flop!

Setup time: $t_{\mathrm{s}} = 18 - 12 + \max(22, 16) = 28\,\mathrm{ns}$

Hold time: $t_{\mathrm{h}} = 4 + 22 - \min(12, 8) = 18\,\mathrm{ns}$

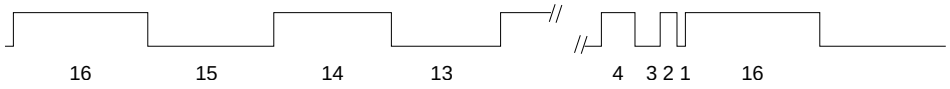
2. (20 points) *Fun with counters.*

   a. A *superstitious counter* is a free-running 4-bit counter that skips the value 13. Build a superstitious counter using one 74x163 4-bit counter and two 2-input gates.



   The combinational logic, which is equivalent to NAND3A, a 3-input NAND gate with one active low input, detects when the count is 12 (or 13). When the count reaches 12, the value 14 is loaded on the next clock, skipping 13. Otherwise, the counter runs freely.

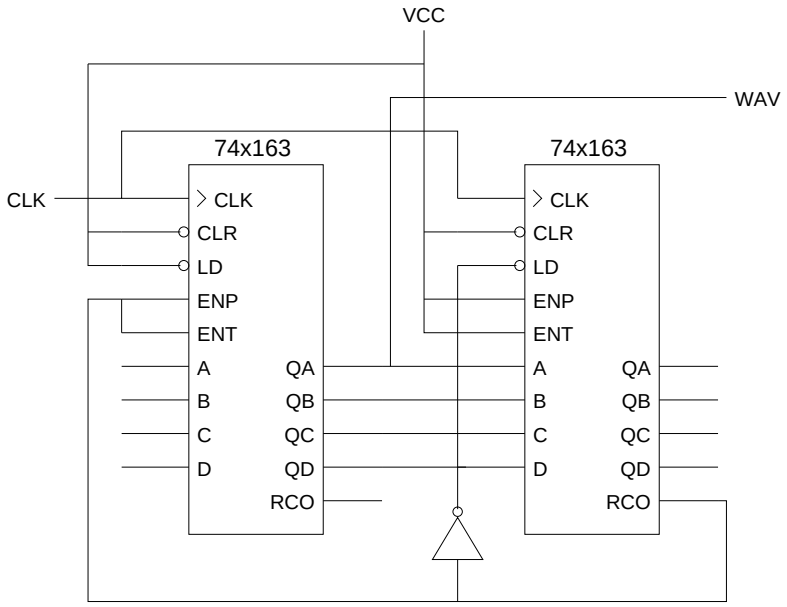
   b. The output of a *chirp counter* is the following waveform.



   The chirp output is high for 16 clocks, low for 15, high for 14, and so on until it is high of one clock duration. Then the cycle repeats. The overall period of this counter is

$$16 + 15 + \cdots + 2 + 1 = (17 \cdot 16)/2 = 136\,.$$

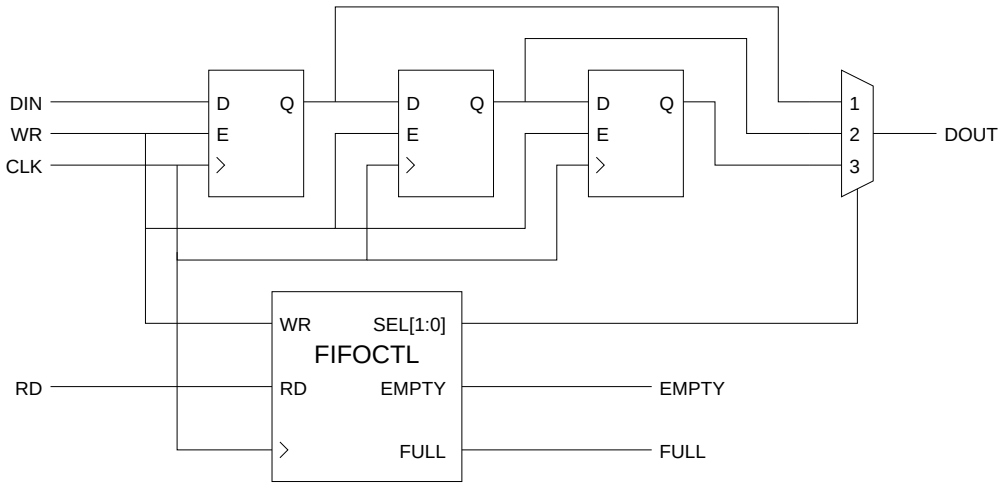
   Use two 4-bit counters and assorted gate(s) to build a chirp counter.



The left counter loads the right counter with a new start value each time the right counter reaches the maximum value of 15 and RCO goes high. At the same time, the left counter is incremented. Suppose that the value to be loaded into the right counter is 0. When the right counter reaches 15, RCO goes high, and at the next rising clock edge, the left counter value increments to 1, so the output WAV is high. The right counter increments at the next clock. After 16 clock cycles, RCO causes 1 to be loaded into the right counter and the output WAV toggles low. After 15 clocks, 2 is loaded into the right counter, the left counter increments so WAV toggles high, and so on.

3. (30 points) *FIFO.* A 3-deep FIFO stores up to three words of data that are retrieved ("read") in a first-in first-out manner. The state of the FIFO control unit is the two-bit binary number in the range $\{0, \ldots, 3\}$ that tells how many words are stored in the FIFO.

   The FIFO state machine has two control inputs, RD ("read") and WR ("write"), and two outputs, FULL and EMPTY. The FIFO controller also has internal signals, SEL0 and SEL1, that select which stored data value to supply to the output.



The control inputs are examined at each rising edge of the system clock. When WR alone is asserted, the FIFO state is incremented by 1, whereas when RD alone is asserted, the FIFO state is decremented by 1. If both RD and WR are asserted when the FIFO is nonempty, the FIFO state remains unchanged, since a new word is written into the FIFO while the oldest word is read and removed. Finally, when neither RD nor WR is active, the FIFO remains unchanged.

The FIFO silently ignores attempts to write when it is full or to read when it is empty. The Moore outputs FULL and EMPTY report when the FIFO is full or empty, respectively.

a. Fill in the following transition/output table for the FIFO state machine.

Q1 Q0	RD, WR				FULL	EMPTY
	00	01	10	11		
0 0	0 0	0 1	0 0	0 0	0	1
0 1	0 1	1 0	0 0	0 1	0	0
1 0	1 0	1 1	0 1	1 0	0	0
1 1	1 1	1 1	1 0	1 1	1	0
	Q1*,Q0*					

b. Find *simplified* transition/output equations for the FIFO state machine.

The canonical sums for Q0 and Q1 each have 8 minterms. Using Karnaugh maps, we obtain simpler sum-of-products representations. Other simplified equations might make use of intermediate expressions such as $\mathsf{RD} \oplus \mathsf{WR}$.

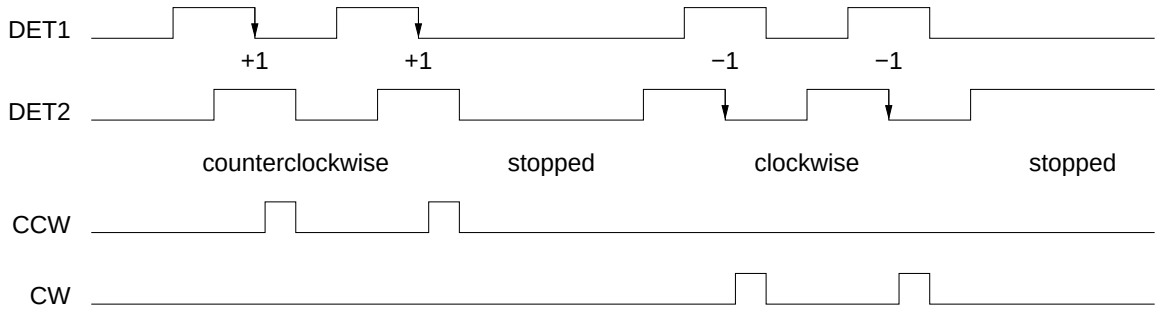
$$\mathsf{Q1^*} = \mathsf{Q1} \cdot \mathsf{Q0} + \mathsf{Q1} \cdot \mathsf{RD}' + \mathsf{Q1} \cdot \mathsf{WR} + \mathsf{Q0} \cdot \mathsf{RD}' \cdot \mathsf{WR}$$

$$\mathsf{Q0^*} = \mathsf{Q0}' \cdot \mathsf{RD}' \cdot \mathsf{WR} + \mathsf{Q1}' \cdot \mathsf{Q0} \cdot \mathsf{WR} + \mathsf{Q1} \cdot \mathsf{Q0} \cdot \mathsf{RD}' + \mathsf{Q1} \cdot \mathsf{Q0} \cdot \mathsf{WR} + \mathsf{Q1} \cdot \mathsf{Q0}' \cdot \mathsf{RD} \cdot \mathsf{WR}'$$

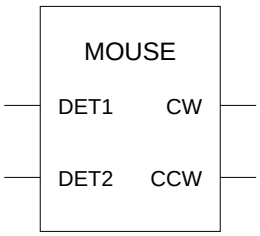
$$\mathsf{FULL} = \mathsf{Q1} \cdot \mathsf{Q0}$$

$$\mathsf{EMPTY} = \mathsf{Q1}' \cdot \mathsf{Q0}'$$

4. (30 points) *Mouse encoder.* The optical encoder wheel in a mechanical ball mouse produces detector pulses that indicate the speed and direction of motion.



Draw the state diagram of a clocked synchronous state machine whose inputs are the detector pulses and whose outputs are signals that can be used by an up/down counter.



This state machine has two Moore outputs, CW and CCW. CW should be active for one clock period when the encoder wheel has completed a clockwise step (indicated by the down arrows in the timing diagram), and CCW should be active for one clock period when the disk has completed a counterclockwise step.

Your state machine must accommodate input signals that are *not* debounced; you may assume that the output of each detector is stable before the other detector output changes.

This problem is open ended; there are quite a few situations to consider. More credit will be given for more complete solutions.

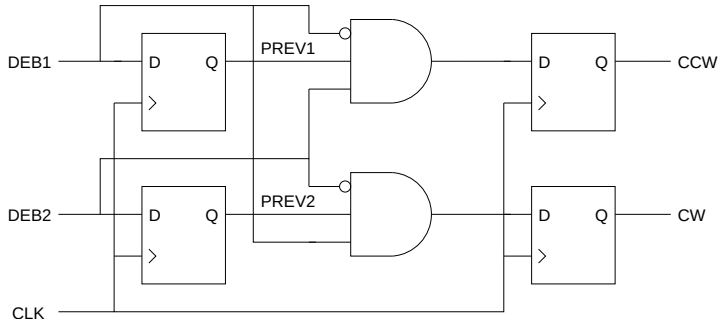
A single state diagram is not the proper way to represent the optical encoder processing machine, for two reasons:

- The state machine can be more simply realized by breaking it into two or more smaller machines.
- Much of the state consists of past values of the inputs and outputs, which can be stored in external flip-flops. In fact, the Moore outputs are really pipelined Mealy outputs, which store the values of Mealy outputs for a complete clock period.

First we design a state machine assuming that the inputs are debounced. The software for version of this state machine uses four variables: PREV1 and PREV2 are the values of the detector outputs during the previous clock cycle, and CW and CCW are the outputs that are pulsed for one clock period when a falling edge is detected on one detector output while the other detector output is high. The debounced inputs are DEB1 and DEB2.

```
for (;;) {
  if (DEB2 == 1 && DEB1 == 0 && PREV1 == 1)
    CCW = 1;
  if (DEB1 == 1 && DEB2 == 0 && PREV2 == 1)
    CW = 1;
  PREV1 = DEB1; PREV2 = DEB2;
}
```

The hardware realization uses four D flip-flops for state variables and Moore outputs.



The detector signals can be used to debounce each other. When one signal changes, further changes of that signal can be ignored until the other signal changes. In order to detect changes, the previous value of the detector signals are remembered. We need one more variable: the value of the signal that changed most recently; call this variable CHANGED.

```
for (;;) {
  if (DET1 != PREV1 == 1 && CHANGED == 2) {
    DEB1 = DET1; CHANGED = 1;
  }
  if (DET2 != PREV1 == 2 && CHANGED == 1) {
    DEB2 = DET2; CHANGED = 2;
  }
  PREV1 = DET1; PREV2 = DET2;
}
```

In the hardware realization shown below, the flip-flop that stores CHANGED represents detector 1 by value 0. Clock connections are omitted to simplify the diagram.