

Physics 219 – Fall, 2005

LabNotes 12 – Sequential Logic

Table of Contents

Wednesday, December 7	1
Memory	1
A Puzzler.....	1
D-Type Flip-Flops	3
Toggle Connection	5
<i>Student Manual</i> section 14-2.....	6
Shift-Register.....	7
One Shot.....	8
<i>Student Manual</i> lab section 14-6c.....	9

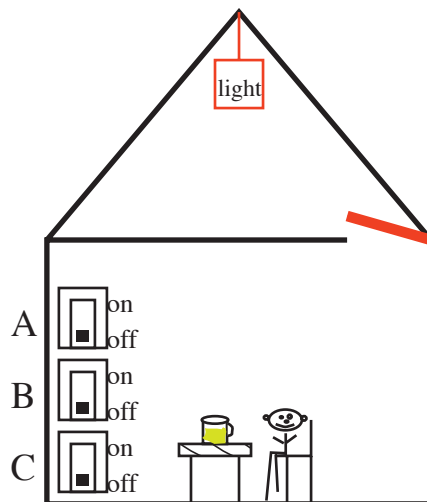
Wednesday, December 7

Memory

Recall that with combinational logic the outputs at a moment in time depend only on the inputs at that time. On the other hand with sequential logic the outputs depend not only on present inputs, but also on past history. There is *memory*.

A Puzzler

Here is a “puzzler” captures the essential point of sequential logic; the ability to remember past “state”):

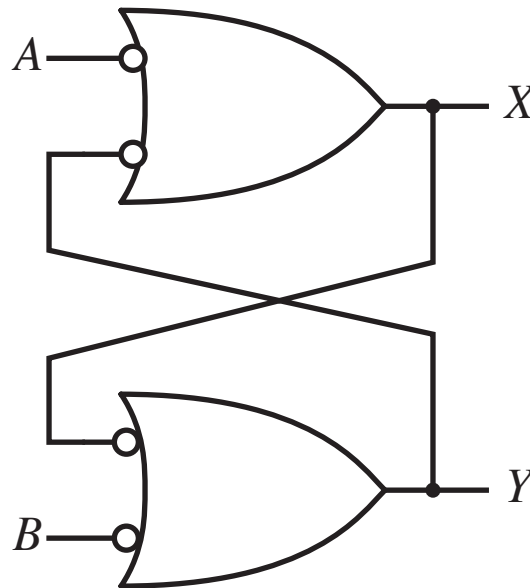


There is a light bulb in an attic, and a person downstairs with three switches (A, B,

and C) and a beer (or perhaps, for Americans, a cup of coffee). The configuration is such that she can't check on the light without visiting the attic.) How can the person figure out which switch controls the light by making only one trip to the attic?

Whereas the basic building block of combination logic are the simple logic gates such as AND, OR, NAND, INVERTER, *etc.*, the basic building block of sequential logic is the **flip-flop**.

A primitive flip-flop can be constructed using simple logic gates *with feedback*:



Suppose *A* and *B* are both HIGH. Then there are two stable¹ output states *XY*:

<i>A</i>	<i>B</i>	<i>X</i>	<i>Y</i>
1	1	1	0
1	1	0	1

You should also be able to convince yourselves that the states

<i>A</i>	<i>B</i>	<i>X</i>	<i>Y</i>
1	1	0	0
1	1	1	1

¹ By "stable" we mean that there are no "contradictions" in the circuit. Note that if *X*=1, *Y*=0 is a stable state then, from symmetry, *X*=0, *Y*=1 must also be stable.

are not stable.

Which of the two possible stable output states will this arrangement find itself in? It turns out that the answer depends on the past history of the arrangement: We can "write to memory" by momentarily bringing either A or B LOW.

If A goes LOW momentarily then the flip-flop will be forced into the output state with $X = \text{HIGH}$ and $Y = \text{LOW}$.

If B goes LOW momentarily then the flip-flop will be forced into the output state with $X = \text{LOW}$ and $Y = \text{HIGH}$.

Thus, if the flip-flop is found in the $X = \text{HIGH}$ and $Y = \text{LOW}$ output state it is a record of the fact that A was the input most recently LOW while if the flip-flop is found in the $X = \text{LOW}$ and $Y = \text{HIGH}$ output state it is a record of the fact that B was the input most recently LOW. This might not seem like much but in fact this simple building block, along with the simple logic gates discussed previously are all that is needed to build a stored program computer.

D-Type Flip-Flops

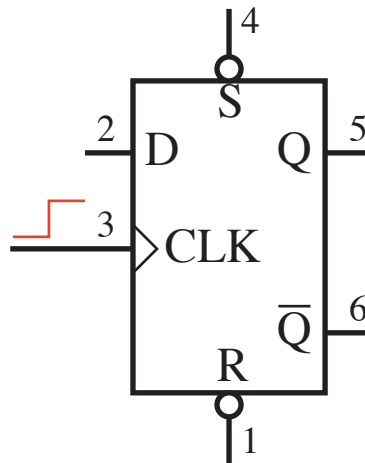
One of the most useful flip-flops are of the **clocked** variety. The output state of the flip-flop depends on the value of the input(s) at the time that a "clock" signal is received. The output is then said to be "latched" (into place) and will remain in this state even if the inputs change until the next clock signal.

Most commonly, clocked flip-flops are **edge-triggered**. In this case the output is latched at the time of a clock transition. If the latching occurs at a "rising" clock transition, the flip-flop is said to be **positive-edge-triggered**, while if the latching occurs at a falling clock transition, the flip-flop is **negative-edge-triggered**. Less often, one encounters **level-triggered** or transparent latches. For these latches the outputs will follow the inputs as long as an ENABLE line is active. (The hexadecimal LED displays that we use in our computers contain an example of this kind of "transparent" latch.)

The most common and useful of the edge-triggered flip-flops is the D-type flip-flop. It simply saves at its output Q the level that was present at its input D at the time it is triggered.

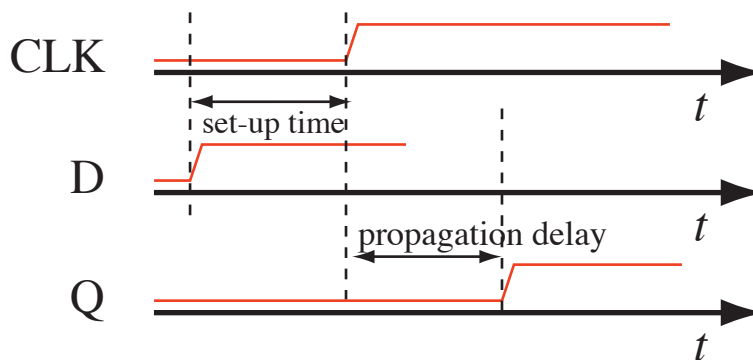
The 74HC74 package contains two CMOS positive-edge-triggered D-type flip-flops. The operation table for these flip-flops is:

CLK	D	Q	\overline{Q}
\uparrow	1	1	0
\uparrow	0	0	1



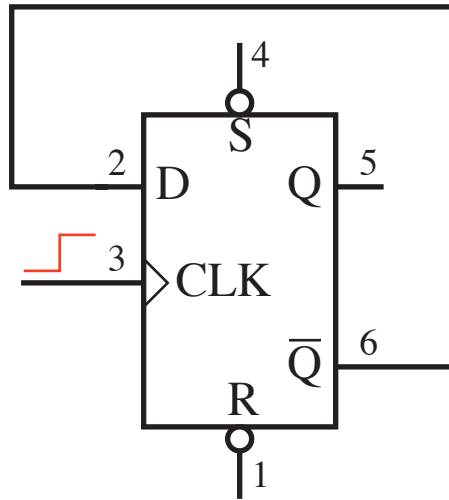
Thus "the D -input is latched at the output by a rising clock input." Thus, this flip-flop serves simply to save one bit of information.

The D input must be stable for a certain short amount of time, called the set-up time, prior to the clock transition. There is also a short propagation delay following the clock transition before the output is actually latched. The essential timing characteristics are shown below:

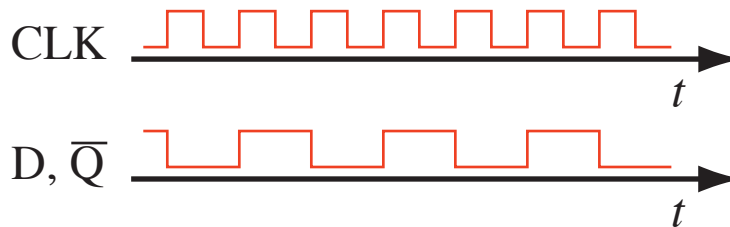


On page 595 of the *Student Manual* there is a the data sheet for a 74HC74 flip-flop. You should become familiar with this kind of data sheet. Note that this data sheet contains a "logic schematic" which shows how the flip-flop is implement using basic logic gates. While more complicated than the primitive flip flop discussed in section 14-1 above, note that at the core of the design is the kind of feedback-based arrangement used in the primitive flip flop. Can you figure out what the functions of the S (PRESET) and R (CLEAR) inputs are?

Toggle Connection



Suppose we wire up a *feedback loop* from the \bar{Q} output to the D input. Suppose further that, just prior to a clock transition $D = \bar{Q} = \text{HIGH}$ and $Q = \text{LOW}$. At the rising edge of the clock signal the $D = \text{HIGH}$ input is latched to the outputs. The outputs will change (Q will become HIGH and \bar{Q} will become LOW) after a propagation delay of a few nanoseconds. With the next rising edge, the outputs will "toggle" back to their original state. The timing diagram will look like:



Thus, we have built a "divide by 2" counter. This may at this point not seem all that impressive. But, as with most cases of the real magic happens when you glue large numbers of these "primitive" building blocks together in inspired ways. Like any good building set, sequential logic needs very few different kinds of parts to achieve many different ends.

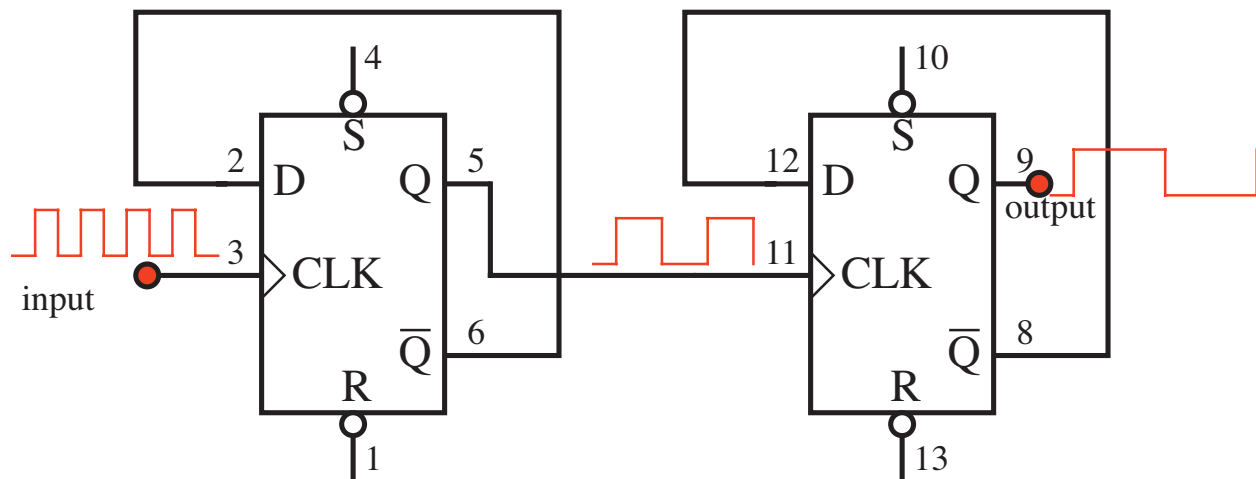
FUNCTION TABLE

INPUTS				OUTPUTS	
$\overline{\text{PRE}}$	$\overline{\text{CLR}}$	CLK	D	Q	\overline{Q}
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H [†]	H [†]
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q ₀	\overline{Q}_0

† This configuration is nonstable; that is, it does not persist when $\overline{\text{PRE}}$ or $\overline{\text{CLR}}$ returns to its inactive (high) level.

Student Manual section 14-2

- Get familiar with D-type flip-flops by completing section 14-2 in the *Student Manual*.
- “Cascade” two D-type flip-flops by wiring connecting the to get divide by four counter, as shown in the figure below. Use the “synch out” output from a function generator to provide the clock signal and use the two channels of your oscilloscope to verify that the frequency of the output signal is indeed a factor of four lower than the frequency of the input signal.

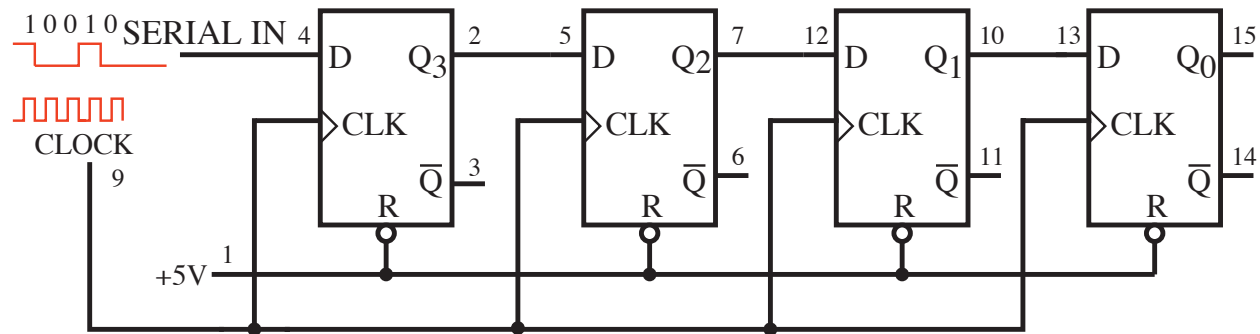


- You have built a **2-bit counter**. To watch it count it binary from 0 to 3, connect two LEDs to the Q outputs of the two flip-flops to act as indicators, then feed in a 1 Hz (or slower) clock signal. Pretty exciting, huh? This scheme can obviously be extended to make an n -bit counter.

Shift-Register

The shift register shown in the figure below can be used as a "4-bit serial-to-parallel converter". Serial data, made up of a sequence of 0's and 1's, are fed into the input. During each CLK cycle the data stream is shifted one step to the right on the "data lines". (Each Q output of a flip-flop represents one bit of data.) Eventually the data reaches the right-most bit Q_0 ; then it is shifted into oblivion with the next CLK cycle. In your home-made shift register, built with D-flops, whenever the level at SERIAL IN changes, the change propagates to the right at a rate of one flop/clock cycle. This manner of "clocking in serial data, one bit at a time" is essentially the same scheme employed in the I²C protocol that we encountered when we connected a serial eeprom to the LogoChip in the *Build Your Own Meter* lab.

In understanding this circuit, it's important to remember the propagation delay; the outputs of each flop don't change until a little bit after the CLK transition. Thus the inputs of each flop see the "old data". We can think of this arrangement as providing a **digitally timed delay**, which is a very useful thing in digital design. (In digital circuits, timing is *everything*.) The size of the delay can be adjusted by varying the clock frequency. It's fun to set the clock speed very low and connect the flop outputs to the bank of LED indicator lights found on your breadboards and watch the transition propagate! Also note that the changes in the flop outputs are **synchronized** with the clock, whereas the changes at IN are not. Being able to synchronize signals to a "system clock" is another important capability in digital design.



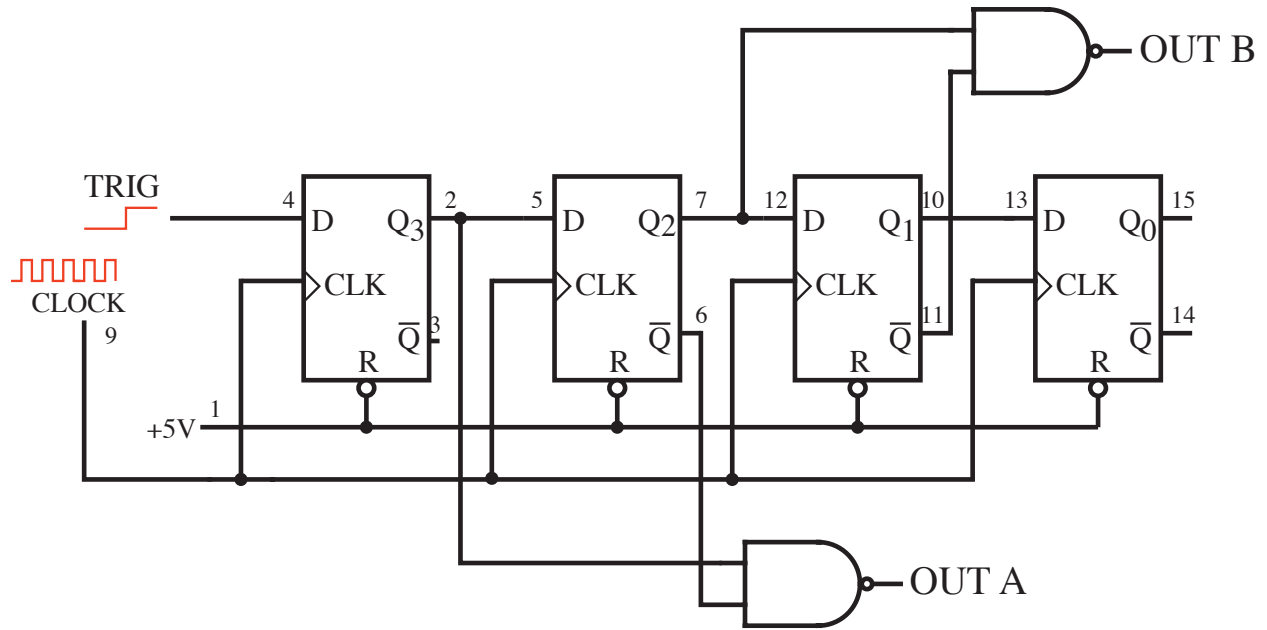
74HC175 Based Shift Register

- Wire up the “shift register” shown in the figure above. Connect LEDs to the Q outputs of the four flip-flops to act as indicator. Try inputting the serial data “by hand” using a slide switch for the data and a 0.1 Hz clock signal provided by a function generator to do the clocking. Can you load a particular number, say 1001, into the register.?

One Shot

Suppose you wanted to generate just a single short-lived pulse. Here’s a clever way to do it:

By wiring a NAND gate as to the Q output of one flip-flop and the \bar{Q} output of the next flip-flop, you've built a "one-shot". If TRIGGER IN makes a transition from LOW to HIGH, the inputs of the NAND gate are both HIGH only briefly as the transition edge propagates along the flops. Thus, for the circuit shown below, the first of a pair of short-lived "LOW pulses" (one cycle in duration) is emitted at OUT A of the leftmost NAND gate. A second similarly wired NAND gate, located one flop downstream, generates a second short-lived pulse at OUT B.



74HC175 Based Double Barreled One Shot

***Student Manual* lab section 14-6c**

- Build the “double-barreled one shot” describe in section 14-6c in the *Student Manual*. While this circuit is clever, as you may realize by now, nowadays a microcontroller provides an equally simple yet much more versatile way to generate pulse sequences.