Physics 364, Fall 2014, Lab #23 **Name:** ____________________

(sequential logic: flip-flops)

Wednesday, November 19 (section 401); Thursday, November 20 (section 402)

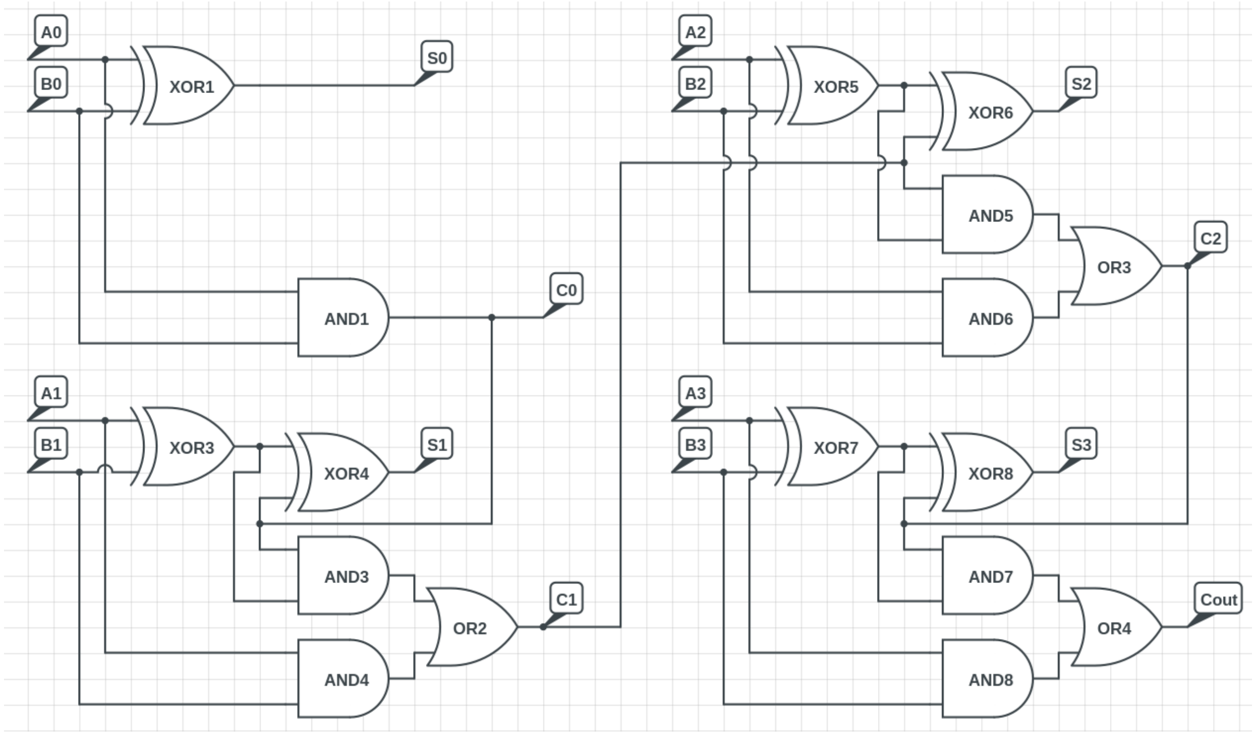
Course materials and schedule are at http://positron.hep.upenn.edu/p364

Today, we continue the digital segment of the course . . .

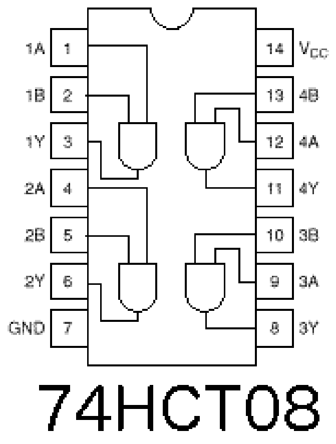
Part 1 **Start Time:** ______________

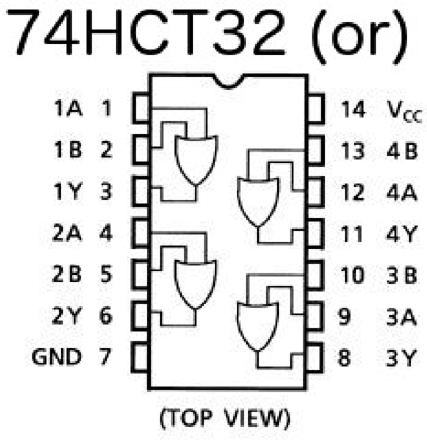
<u>4-bit adder</u> (time estimate: 60 minutes)

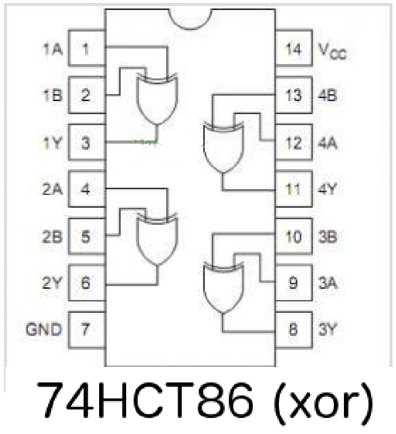
This circuit is purely "combinational" logic, but it may help us to see the motivation for sequential logic. It is the same 4-bit adder that we saw in last weekend's reading, except that we omit the "carry in" bit to reduce the number of wires you need to run.



Build the four-bit adder, shown above, using 7 AND gates (this requires a total of two 74HCT08 "quad two-input AND gate" chips, each of which contains four separate AND gates), 7 XOR gates (requires two 74HCT86 "quad two-input XOR gate"), 3 OR gates (requires just one 74HCT32 "quad two-input OR gate"), a DIP-switch array (contains 8 switches) to use for inputs, and 5 LEDs (with series resistors) for outputs. More details, along with pinouts for the ICs (integrated circuits), are shown below and on the next page.

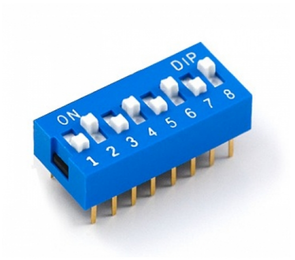


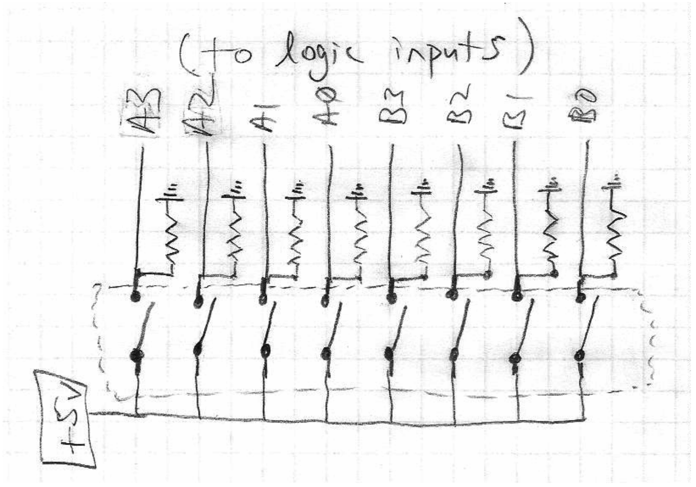


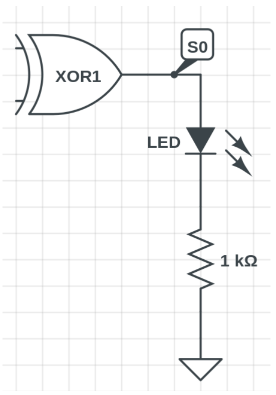


On each logic IC, connect pin 14 (top-right) to $V_{CC} = +5$ V, and connect pin 7 (bottom-left) to ground.

Your adder requires two four-bit binary numbers as input — a total of 8 bits. To provide these inputs, use an 8-position DIP switch. Each switch is an open circuit when in the OFF/down position and connects the two corresponding pins when in the ON/up position. Connect each pin of the bottom side (the side that **doesn't** say "ON") to +5 V, and connect each pin of the top side (the "ON" side) through a separate 1 kΩ series resistor to ground. Check with a scope or a voltmeter that indeed each pin on the "ON" side of the DIP switch is set to 0 V when the switch is in the "OFF" position and is set to +5 V when the switch is in the "ON" position. These 8 switch outputs (labeled $A_3 \ldots A_0$ and $B_3 \ldots B_0$ in the diagram below) will be the 8 inputs of your adder.







Your adder also needs a way to display its output. For each of the adder's five outputs (S_0, S_1, S_2, S_3, and C_{out}), connect an LED with a 1 kΩ series resistor to ground, as illustrated in the above-right diagram for S_0. The output will be easier to interpret if you make the order of LEDs, from left to right, be C_{out}, S_3, S_2, S_1, S_0.

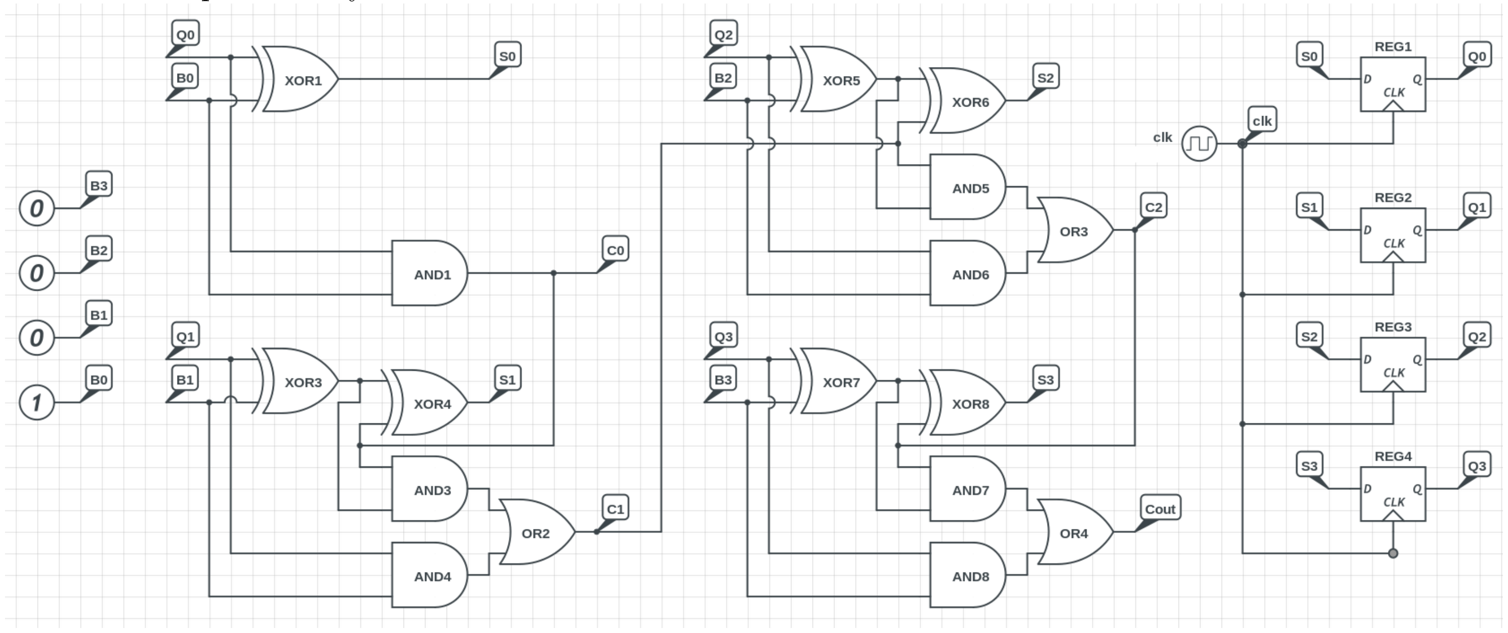
Check that your adder indeed adds the two four-bit numbers keyed in using your DIP switches. List a few examples that you checked.

Don't take your adder apart! You will use it in the next part.

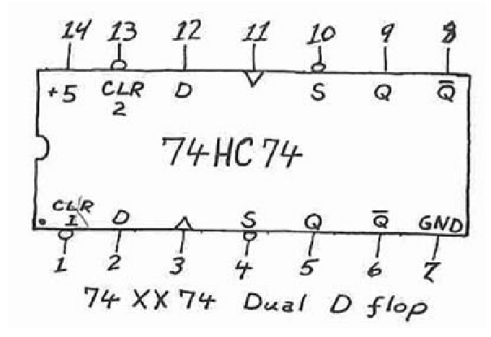
Part 2 **Start Time:** ______________

4-bit counter (time estimate: 45 minutes)

Now we'll convert your 4-bit adder into a 4-bit counter, by adding four D-type flip-flops to "remember" the current value of the counter. Your existing adder will add 1 to the count once per clock cycle.



To accomplish this, you first need to find two **74HC74** "dual D-type flip-flop" ICs. Each of these ICs contains two flip-flops. First connect pin 7 to ground and pin 14 to +5 V. Each of the IC's two flip-flops has a **D** input, a **clock** input, a **Q** output, and a $\overline{\mathbf{Q}}$ output (opposite of **Q**). Each flip-flop also has (active-low) **set*** and **clear*** inputs, which we will not use, so **we need to wire them to** +5 V **(their inactive state).** Don't forget to do this!



function	flip-flop 1 pin	flip-flop 2 pin
D	2	12
clock	3	11
set*	4	10
clear*	1	13
Q	5	9
Q*	6	8

Wire the four S_3, S_2, S_1, S_0 sum bits to the four flip-flop **D** inputs. Wire the four corresponding flip-flop **Q** outputs (not the $\overline{\mathbf{Q}}$ outputs) to the A_3, A_2, A_1, A_0 adder inputs (at the logic gates), and make sure the four corresponding DIP switches are in the OFF position. (Leave the B_3, B_2, B_1, B_0 adder inputs unchanged.) Wire up another four LEDs (preferably of a different color from the five you already have), with 1 kΩ series resistors, to display Q_3, Q_2, Q_1, Q_0, the four flip-flop **Q** outputs (not the $\overline{\mathbf{Q}}$ outputs). **Don't forget to wire 74HC74 pins 1,4,10,13 to** +5 V **to disable the unwanted set*** **and clear*** **inputs.**

Now set up your function generator to produce a 1 Hz square wave, 5 V_{pp}, with a +2.5 V DC offset, so that its LOW state is 0 V and its HIGH state is +5 V. Make sure the FG ground is wired to the breadboard ground. After checking with the scope that the square wave from the FG has the correct LOW and HIGH voltages (so that you don't cook the flip-flops), send this square wave to the four flip-flops' **clock** inputs (pins 3 and 11 of the two 74HC74 ICs).

Set the adder's $b_3b_2b_1b_0$ inputs to 0001 so that the counter (hopefully!) increments by 1 each clock cycle. Watch the circuit count by watching the LEDs!

Clock it at 1 Hz. Watch it count by watching the LEDs. Since we just made up this lab for this year, let us know if getting to this stage involves anything tricky that we should warn everyone else about.

Why do the old $S_3S_2S_1S_0$ LEDs stay one count ahead of the new $Q_3Q_2Q_1Q_0$ LEDs?

How would you make your counter count up by 2's or by 3's instead? Try it.

What trick can you use to make your counter count *down* by 1 each clock cycle? As a hint, see if you can remember how you would represent the number "-1" as a 4-bit **two's complement** binary number.

Set the $b_3b_2b_1b_0$ inputs back to 0001 so that the clock counts up by 1 each clock cycle. Now clock it at about 1 MHz (but not so fast that it counts incorrectly) and watch the flip-flops' $Q_3Q_2Q_1Q_0$ outputs with the scope. They should all change together and should cleanly count out 0000, 0001, 0010, 0011, ... If instead you look with the scope at the adder's $S_3S_2S_1S_0$ outputs, you should see that the S_3 bit changes later than the S_2 bit, etc. If you look at the C_{out} bit, it should change even later. You may also see some "glitches" on C_{out}, S_3 and S_2 that last only a small fraction of a clock period. What could cause these glitches?

Take a photo of your counter circuit — particularly if it looks like a rat's nest of wires. The tedious nature of building circuits like this out of many individual logic gates is a key motivation for the study of Field Programmable Gate Arrays that will occupy the final four labs of the semester.

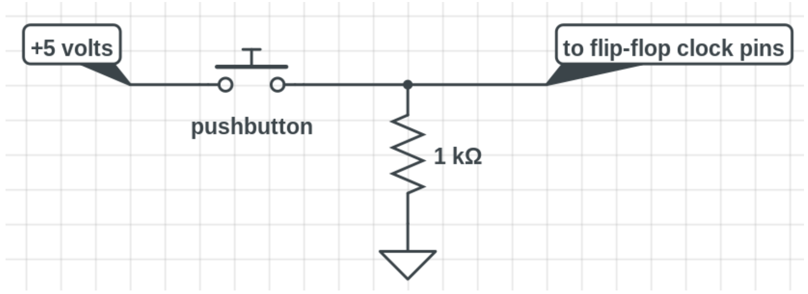
Don't take your counter apart! You will use it in the next part.

Part 3 **Start Time:** ______________

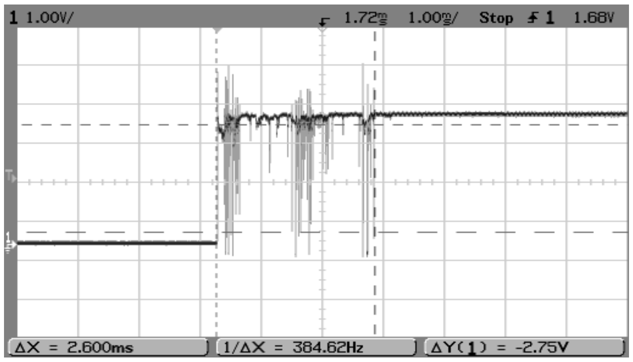
switch "debouncing" (time estimate: 45 minutes)

Don't take your counter apart! You will use it again here!

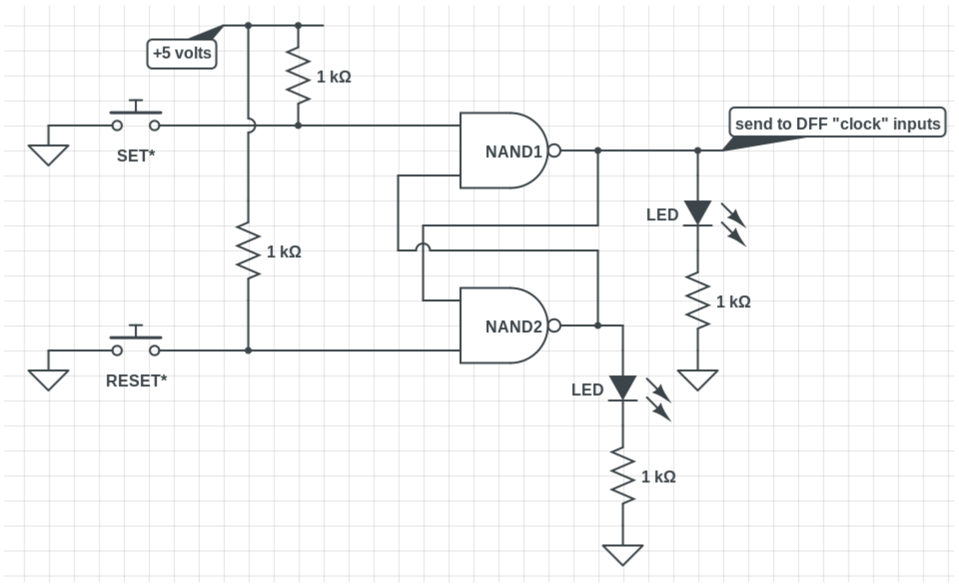
First use a simple push-button + resistor as a button-actuated clock pulse, to replace the clock supplied by the function generator.

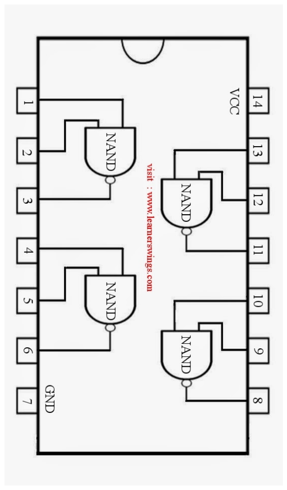


See if you can make your counter count up one tick per press of the pushbutton. We expect that some (if not most) of the time, you will find that your counter counts up by more than just a single tick each time you press the button. To try to figure out why, look at your button's "output" signal with a scope. When you press or unpress the button, you should see the switch output "bounce" wildly back and forth before settling down. By the way, if you get a really good scope trace of a switch bouncing many times, please save it for us. This photo from Wikipedia is the best we have so far.



One commonly used method to work around the "bouncing switch" problem is to use an SR latch (which is one of the simplest examples of a flip-flop) and a pair of switches to provide a push-button clock signal. This SR latch uses a pair of cross-coupled NAND gates; it is similar to the version we studied in the notes (and in Eggleston), which uses two NOR gates.[1] Build the latch shown below, using two of the four NAND gates found in a single 74HC00 IC. You'll need two more pushbuttons and two more LEDs, too. You press the **SET*** button to make the clock go HIGH, and then press the **RESET*** button to make the clock go LOW. This eliminates the bouncing, because once you've pressed **SET***, the latch stays in the HIGH state (no matter what you do to the **SET*** button) until you press **RESET***, and vice versa. You can verify this set/reset behavior with the LEDs, if you didn't already do so in the optional part of Lab 19.



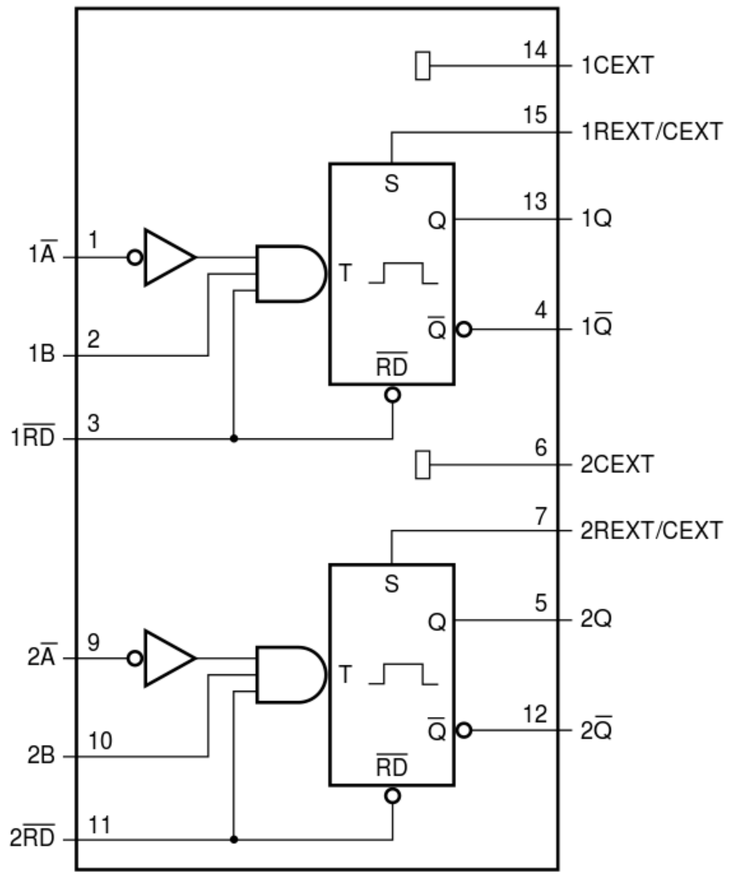


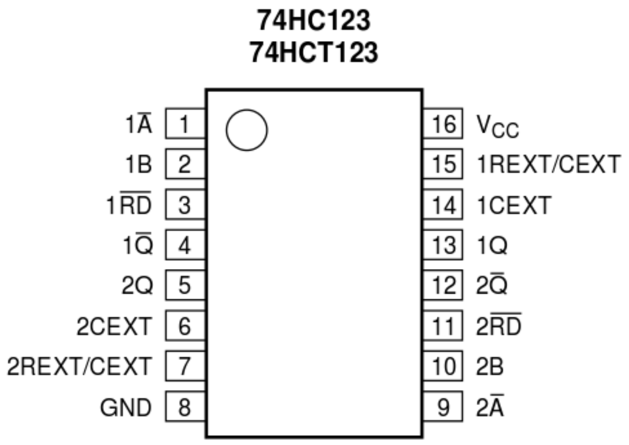
Now if you use the above circuit's output (instead of the single pushbutton's output from the previous page) as the clock input for your four DFFs, you should see your counter always count by one step at a time.

[1]The NOR version is preferable, because its two inputs are active-high **SET** and **RESET**, whereas the NAND version has active-low inputs called **SET*** and **RESET***. Active-high logic is more intuitive than active-low. But we currently have only NAND gates in stock, so we'll use the NAND version.

Optional: It's a little bit annoying to have to press two separate buttons each time you want a single clock pulse.[2] There is a device known as a **monostable** (often colloquially called a "**one-shot**") that can also solve this problem. A one-shot outputs a pulse of constant width, once it is triggered by an input pulse. The width of the output pulse is programmable via an RC time constant. Additional triggers are ignored during the time when the output is HIGH, so as long as the output pulse is wider than the time during which the switch is bouncing, the switch bounce is eliminated. If you want to give a 74HCT123 a try, wire pin 8 (lower right) to ground, pin 16 (upper left) to +5 V (for power). Then wire **RD*** (pin 3) to HIGH, wire **A*** (pin 1) to LOW, and wire up a single push-button such that pressing the button gives you a LOW-to-HIGH transition on **B** (pin 2). You will also need a 1 μF capacitor between pins 14 and 15 and a 10 kΩ resistor from pin 15 to +5 V. You should see a 10 ms-wide pulse emerge from the **Q** output (pin 13) in response to pushing the button. This could, in turn, be used to replace the clock signal for your counter.

By the way, a technical term for a flip-flop is "bistable," meaning it has two stable states. The one-shot has only one stable state, hence the name "monostable."



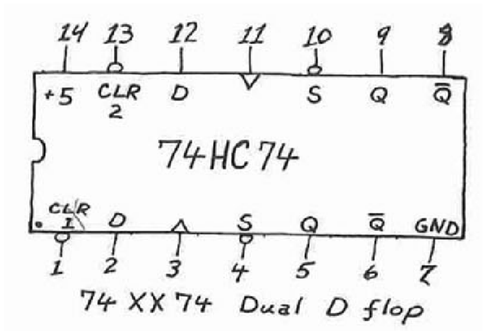


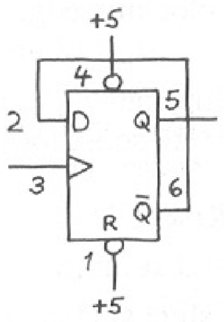
[2] One simple workaround for this is to use an SPDT (single-pole double-throw) pushbutton switch instead of two separate pushbuttons. `en.wikipedia.org/wiki/Switch#Contact_terminology`

Part 4 **Start Time:** ______________

<u>**one more counter**</u> (time estimate: 30 minutes)

If you take a single D-type flip-flop (you can re-use one of the 74HC74 chips you already have on your breadboard) and wire its $\overline{\mathbf{Q}}$ output (that's Q-bar, not Q) back to its **D** input, you will find that the **Q** output oscillates with one-half the frequency of the **clock** input. Wire up your flip-flop as shown in the right figure below. (Make sure power and ground are still there, too.) Send a clock into pin 3 in any way you wish: you could use your push-button clock from Part 3, or you could use the square wave from the function generator (be careful that you have LOW=0 V, HIGH=+5 V). In the first case, you probably want to observe the **Q** output with the usual LED and resistor. In the second case, perhaps you prefer the oscilloscope.

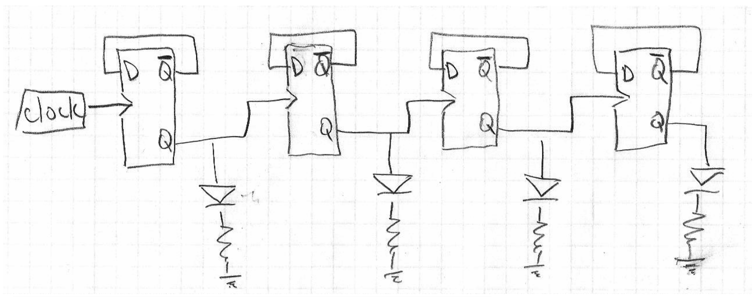


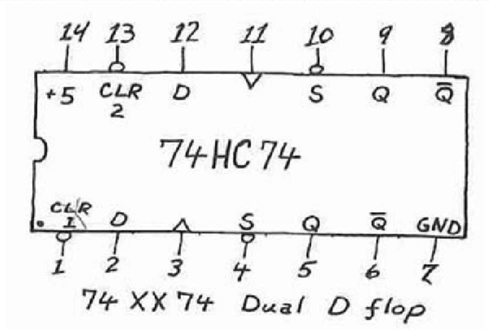


Because this trick works to divide the clock frequency in half, a flip-flop configured in this way is sometimes called a "divide by two" circuit. Can you explain why this circuit's output has half the frequency of the incoming clock?

It's easy to make a **ripple counter** out of several D-type flip-flops. Wire up the four flip-flops (in two 74HC74 ICs) that you already have on your breadboard, to make a 4-bit ripple counter, as shown below. Each DFF is wired up as a divide-by-two, by sending its Q^* output back to its D input. The next trick is to use the Q output of the first DFF as the clock for the second DFF, and so on, which gives clock/2, then clock/4, then clock/8, then clock/16.

Try it! Make sure that you still have your V_{CC} and ground connections, and make sure that all of the flip-flops' set* and clear* inputs are tied to V_{CC}.





function	flip-flop 1 pin	flip-flop 2 pin
D	2	12
clock	3	11
$\overline{\text{set}}$	4	10
$\overline{\text{clear}}$	1	13
Q	5	9
$\overline{Q}$	6	8

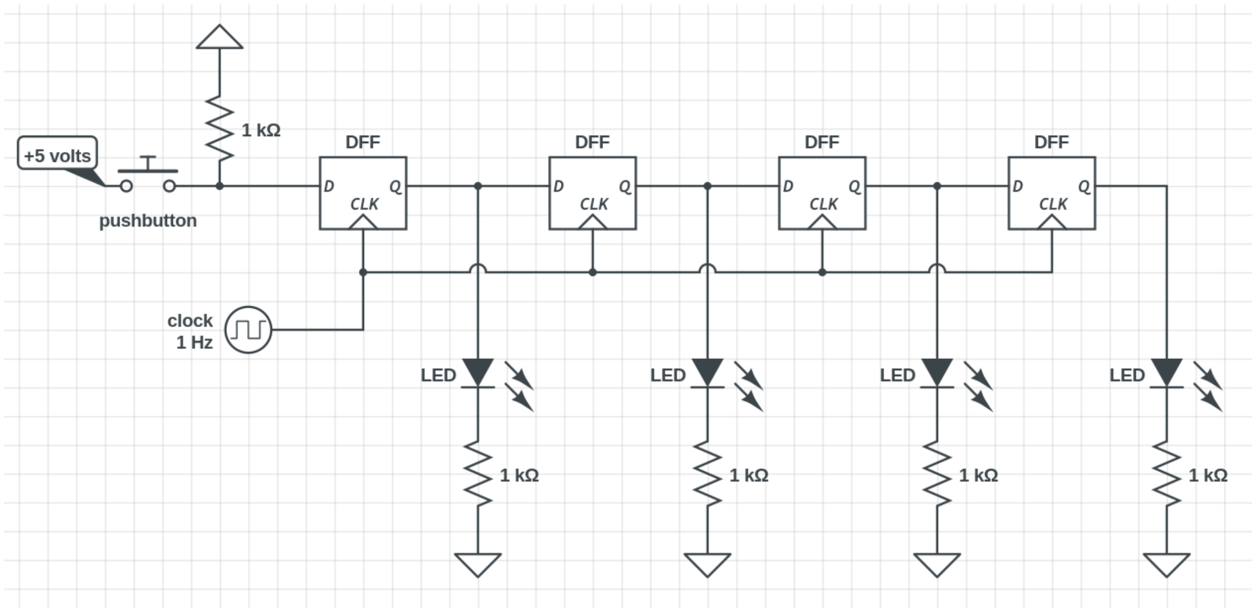
The main drawback of a ripple counter is that the output bits don't all update at the same time — this destroys one of the key benefits of synchronous logic. But you see this circuit from time to time because it's such an easy way to make a counter. Clock your counter from the function generator's square wave and see if you can see with the oscilloscope that the successive bits of the counter are delayed, with respect to the clock, by several tens of nanoseconds, with delays accumulating at each successive divide-by-two.

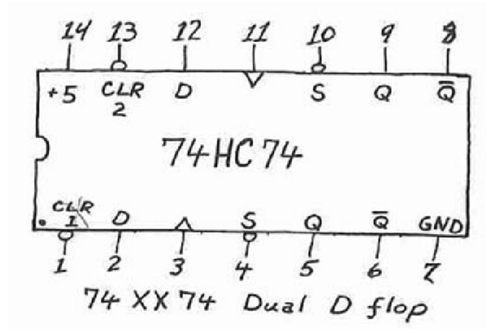
Optional: There are also integrated circuits that count for you, e.g. the 74HC193 “4-bit synchronous up/down counter.” If you have a lot of spare time today, you could try to figure out how to make this chip count. We have a drawer of them in back. (Regrettably, I did not have time to spell out how to do it, and I suspect that this lab is already long enough without this part.) http://www.nxp.com/documents/data_sheet/74HC_HCT193.pdf

Part 5 (optional!) **Start Time:** ______________

shift register (time estimate: 30 minutes)

If you have extra time, try making a 4-bit (or 8-bit, if you're feeling ambitious) shift register, using two (or four) 74HC74 "dual D-type flip-flop" ICs. Make sure that you have your V_{CC} and ground connections, and make sure that all of the flip-flops' set* and clear* inputs are tied to V_{CC}. Use a 1 Hz square wave (alternating between 0 V and +5 V) from the function generator to clock all of the flip-flops. By momentarily pushing the button (for about one second), you can inject a blip that propagates from left to right through the circuit, illuminating each LED in turn. The contents get "shifted" one position to the right with each clock cycle. A shift register, by the way, is one possible mechanism for using one Arduino output to control many separate signals, as long as those signals don't need to update too often.





function	flip-flop 1 pin	flip-flop 2 pin
D	2	12
clock	3	11
$\overline{\text{set}}$	4	10
$\overline{\text{clear}}$	1	13
Q	5	9
$\overline{Q}$	6	8