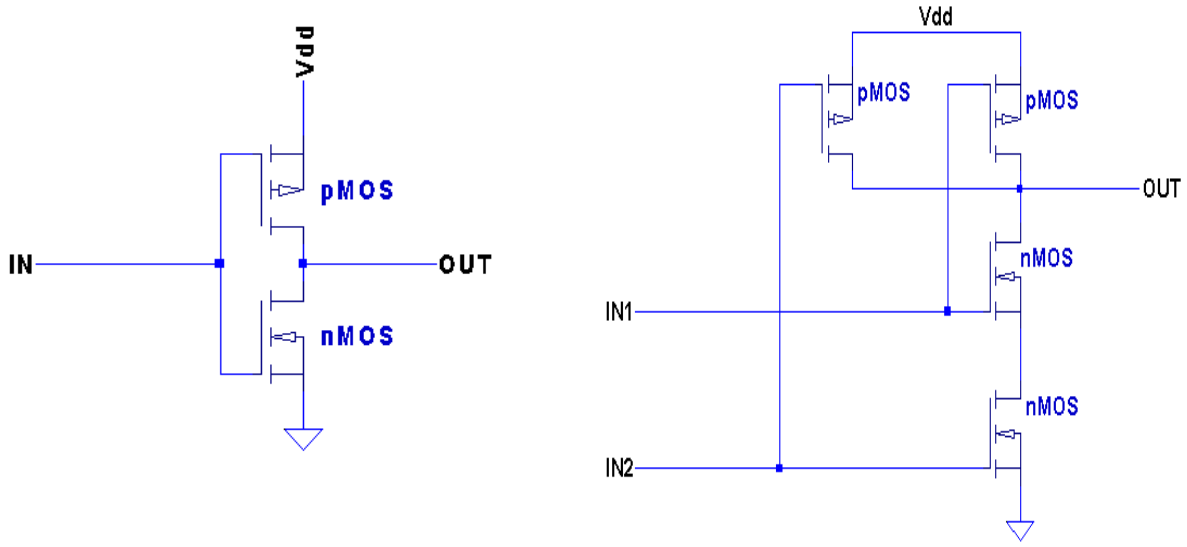
Physics 364 – fall 2010 – Lab #8 – due by lecture, Monday 2010-11-08

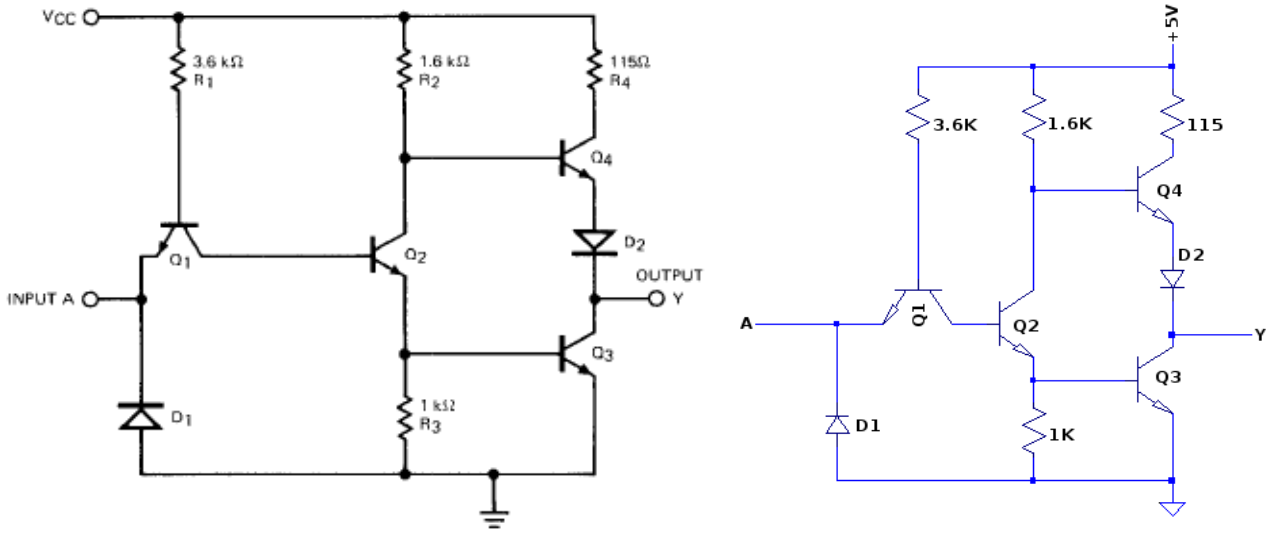
Lab #8 begins the digital part of this course. You saw a hint of digital electronics last week when you built or simulated a CMOS inverter and CMOS NAND gate (repeated here for illustration). Nearly all real-world logic nowadays is CMOS-based. But it turns out, to my surprise, that all discrete logic components currently available in the lab are of the older TTL style (using BJTs), so we will spend this week working with TTL gates.

Partly this will give you a chance to learn a bit about the innards of logic gates; partly it will give you time to get used to new digital concepts; and partly the tedium of working with discrete digital components will illustrate for you what a terrific invention programmable logic is. Starting next week (a week earlier than I had initially planned), we'll start building digital logic inside FPGAs.

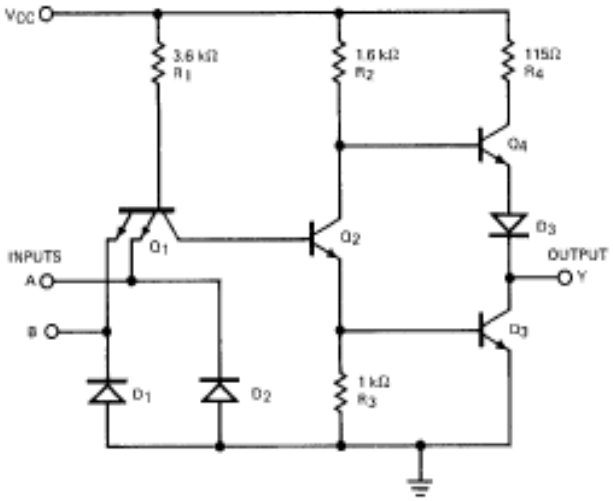


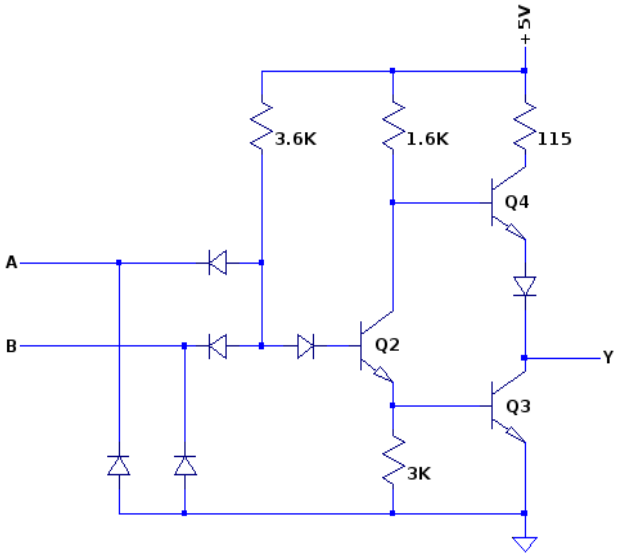
<u>Part 1: inside a TTL inverter and NAND gate</u>

(a) I show below the schematic diagram (from the 1972 Fairchild Semiconductor TTL data book) for 1/6 of a 7404 inverter, alongside my LTspice model. Open the model (on the positron/p364 web site) in LTspice and get a feeling for how the circuit works. (You might want to leave this part of the lab for last, to save lab time. I put it at the beginning to make the connection with last week's lab.)



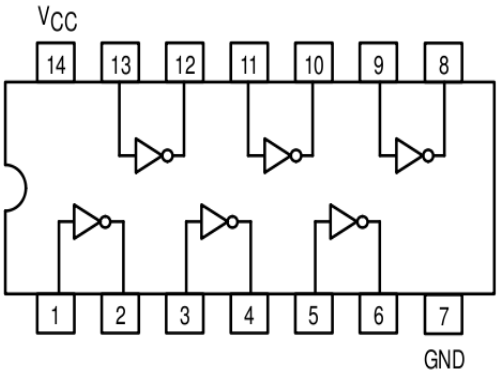
(b) I show below the 1972 Fairchild schematic for 1/4 of a 7400 NAND gate, alongside my LTspice model. Since LTspice has no model for a dual-emitter NPN transistor, I replaced it with three diodes. Open this model up (preferably at the end of the lab) and get a sense of how it works. Also compare the maximum current that this gate's output can sink in its LOW output state with the maximum current that it can source in its HIGH output state. Which of the two states would be more effective at lighting an LED? (This question will be more clear after you look at Part 2b.)





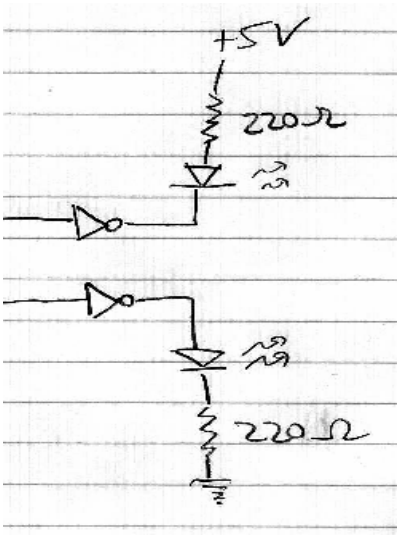
Part 2: real-life 74LS04 TTL inverter

Now get used to a real TTL inverter chip, the 74LS04, which contains six inverters, as illustrated to the right. First connect pin 7 to ground and pin 14 to +5V. (Remember to make these power connections on each 74LSnn chip that you use.) Be careful never to drive these chips' input pins with a voltage above +5V or below 0V.



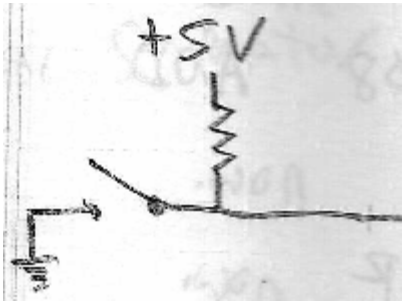
(a) Measure Vout on pin #2 vs. Vin on pin #1 and graph the response, showing the values the data sheet calls V_{IH} (input HIGH voltage), V_{IL} (input LOW voltage), V_{OH} (output HIGH voltage), and V_{OL} (output LOW voltage). Note that these values vary between logic families – i.e. the values for a CMOS-based 74HC04 would be different from those for a BJT-based 74LS04, etc.

(b) Try powering an LED from the pin 2 output, using a 220Ω series resistor to limit the LED current. Try connecting the LED+resistor between pin 2 and ground, so that it lights when the output is HIGH; and then try connecting the LED+resistor between pin 2 and +5V, so that it lights when the output is LOW. Which configuration is brighter? Measure (or infer) the LED current in both cases.



For what follows, I will assume that the LED burns noticeably more brightly when connected from pin 2 to +5V (with inverter output LOW) rather than from pin 2 to ground (with inverter output HIGH).

(c) Now set up a DIP switch bank so that you can supply up to eight logic inputs with 0V or +5V, depending on the switch settings. Each of the eight switches should be wired as shown at right. (So you'll need eight 1K resistors for pulling up the wires to +5V when the corresponding switches are opened.)

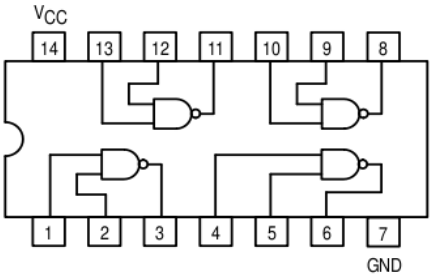


Connect all six of your 74LS04 inverter inputs to individual DIP switch lines. Then connect all six inverter outputs to LEDs and 220Ω resistors. Check that you can turn the LEDs on and off by toggling the switches.

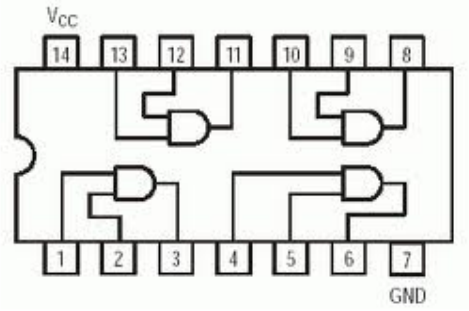
For most of what follows, we will use this DIP switch configuration to turn on and off the inputs of various logic gates, and we will use the inverter + LED + resistor to see the gates' outputs. Keeping the inverters in place will be useful so that a HIGH output from a gate lights the LED – more intuitive than having a HIGH output turn off the LED.

Part 3: 74LS00 NAND gate and its friends

(a) Connect power (+5V) and ground for a 74LS00 NAND gate. Now use two of your DIP switch channels to drive pins 1 and 2, and use one of your inverter+LED+resistor channels to observe pin 3. Write out the truth table for the NAND gate and check that the 74LS00 NAND behaves as expected.



(b) Now figure out how to use a second NAND gate (of the four available on your 74LS00) as an inverter, and invert the output of your NAND, so that it becomes an AND. Is the truth table what you expect?

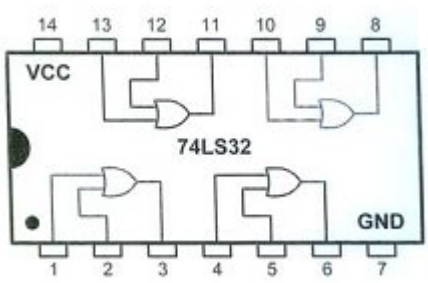


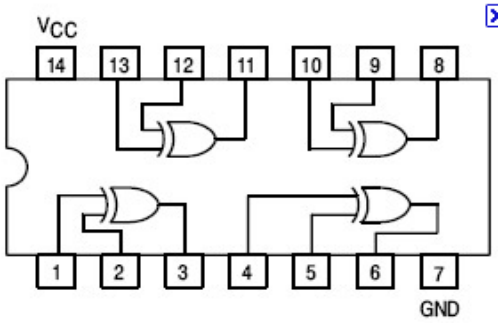
(c) Now remove the 74LS00, connect a single channel of a 74LS08 AND gate instead (don't forget power and ground!), and check the truth table.

(d) Now try a 74LS32 OR. Write out the truth table.

(e) Try a 74LS86 XOR. Write out the truth table.

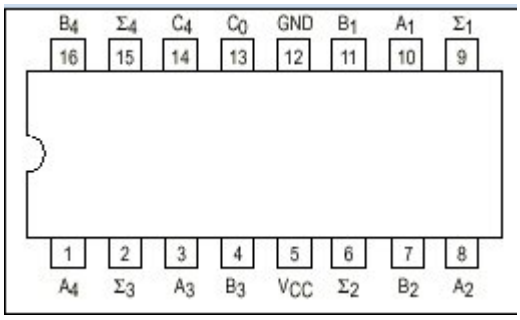
(Note that the 74 series quad two-input gates all have the same pinouts.)





Part 4: 74LS83 4-bit binary full adder

Wire up a 74LS83 4-bit binary full adder. (**Note unusual locations for power and ground!**) Use your inverter+LED+resistor channels to display the outputs C4, Σ4, Σ3, Σ2, Σ1. Use your DIP switch setup to drive inputs A4,3,2,1 and B4,3,2,1. Initially ground the input C0, then investigate its effect. Prove to yourself that the 74LS83 correctly adds two four-bit numbers. What are the purpose of the C0 carry input and C4 carry output? (Hint: how would you connect several 74LS83 units together to make an adder for 8-bit, 12-bit, or 16-bit numbers?)

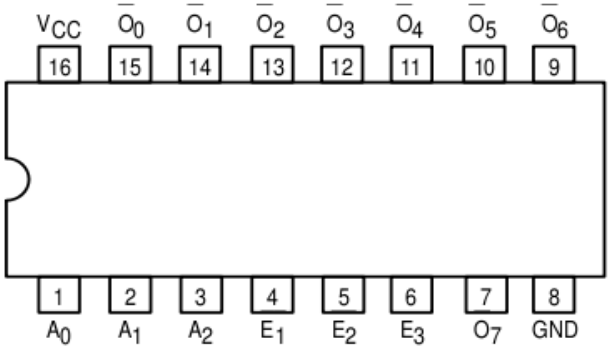


How would you build such an adder from discrete gates (AND, OR, XOR, inverter, etc.)? Sketch out a solution, but don't bother to build it. Remember to consider carry in and carry out. It is probably easiest to draw a one-bit full adder and to note how to connect several such units to make a four-bit adder.

Part 5: 74LS138 decoder

Wire up a 74LS138 decoder. Remember power and ground. Use DIP switches to drive the three address lines A0, A1, A2. Fix E1,E2 to ground and E3 to V_{CC}. Display outputs O0—O7 on LEDs. (How would you do that? Note that they are "active low" outputs.)

How does the pattern of illuminated LED(s) change as you count A0,A1,A2 through all eight possible values? What is happening here?



PIN NAMES

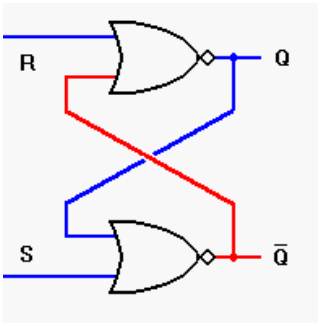
$\overline{A_0}$–$\overline{A_2}$	Address Inputs
$\overline{E_1}$, $\overline{E_2}$	Enable (Active LOW) Inputs
E_3	Enable (Active HIGH) Input
$\overline{O_0}$–$\overline{O_7}$	Active LOW Outputs (Note b)

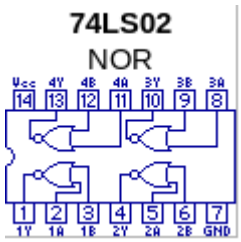
For added fun, make E3 oscillate by driving it with a 1Hz TTL pulse (do not go lower than 0V or higher than +5V) from the function generator. To make it a little bit harder to cook your decoder chip, put a 1K series resistor between the function generator output and the E3 logic input. Now as you change the address lines, you should see the corresponding output channel's LED blinking.

Part 6: home-made simple flip-flop

First get a sense of what this flip-flop should do by trying out the web-based simulation program at http://www.play-hookey.com/digital/rs_nor_latch.html. Now build it, using two channels of a 74LS02 NOR chip.

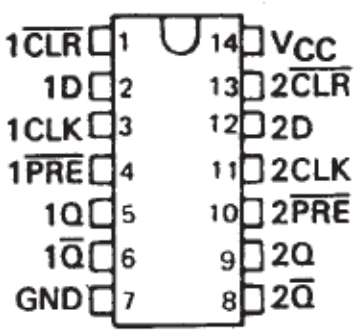
R	S	Q(next)
0	0	Q(old)
0	1	1
1	0	0
1	1	(forbidden / 0)





Now try out the three non-forbidden input combinations and watch how the flip-flop remembers and updates its stored value. (S stands for "set," and R stands for "reset," by the way.) You might find it convenient to use a pair of momentary-contact push-button switches in place of the DIP switches. Use whatever you find most comfortable.

Part 7: Synchronous logic



This will be our first circuit to use a clock. The clock plays the role, for synchronous logic, of the conductor in an orchestra or the bell that rings in a high school to tell everyone to change classrooms. Having a memory (hence statefulness) and having a clock (for coordinated changes of state) often go together.

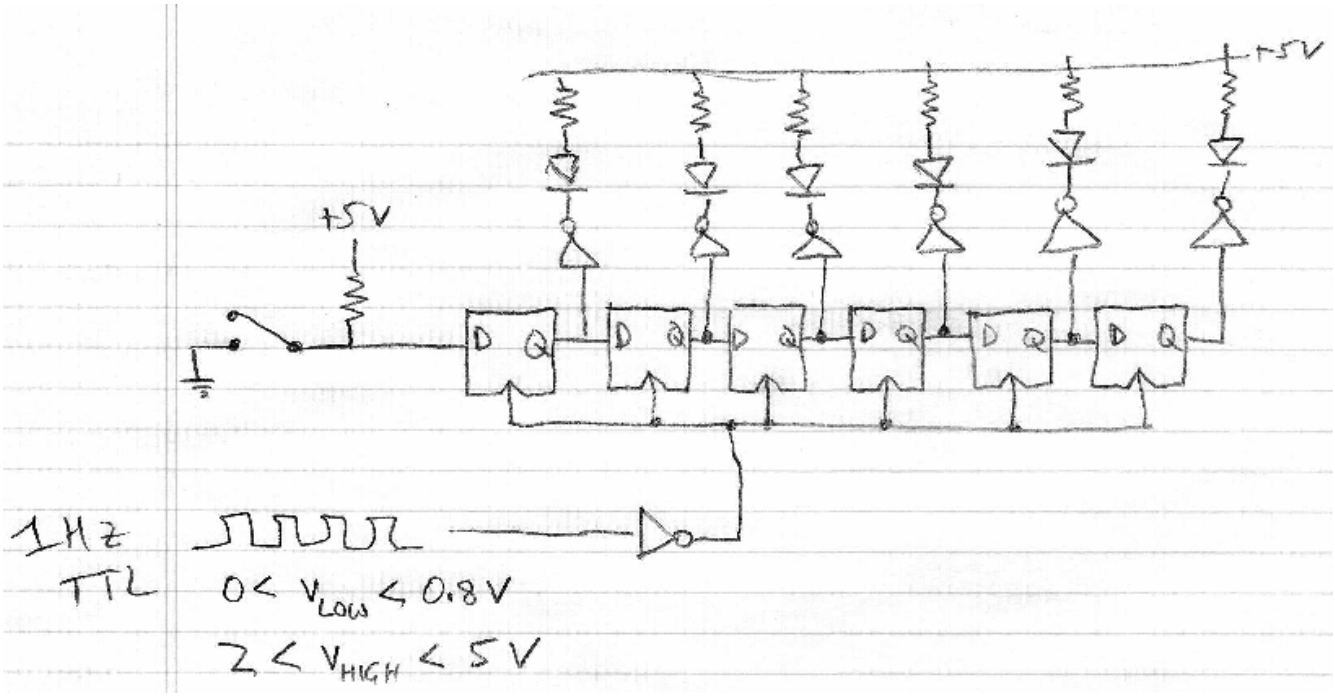
(a) Grab three 74LS74 dual D-type flip-flops, and set them up as a shift register. Remember to connect power and ground! Also grab a spare 74LS04 inverter: we will use one channel of inverter between the output of the function generator (which we'll use for a clock source) and the six 74LS74 clock inputs, so that any mishaps with the clock amplitude only burn out the single inverter, not the six flip-flops.

Program the function generator to output a 1Hz square wave whose low state is 0V and whose high state is +5V. Be careful not to go outside of the 0V,+5V range. In fact, +4V would be plenty high. Test the function generator output with the oscilloscope before feeding it to the inverter. Send the inverted clock signal to the six D-flop clock inputs. Monitor the six Q outputs with your six channels of inverter-LED-resistor combinations.

Connect the unused (active-low) CLR* and PRE* outputs to VCC. (CLR means "clear," PRE means "preset" (which basically means "set", but ignoring the clock), and the * means active-low, i.e. 0 volts to assert the signal and +5V to leave the signal unasserted.)

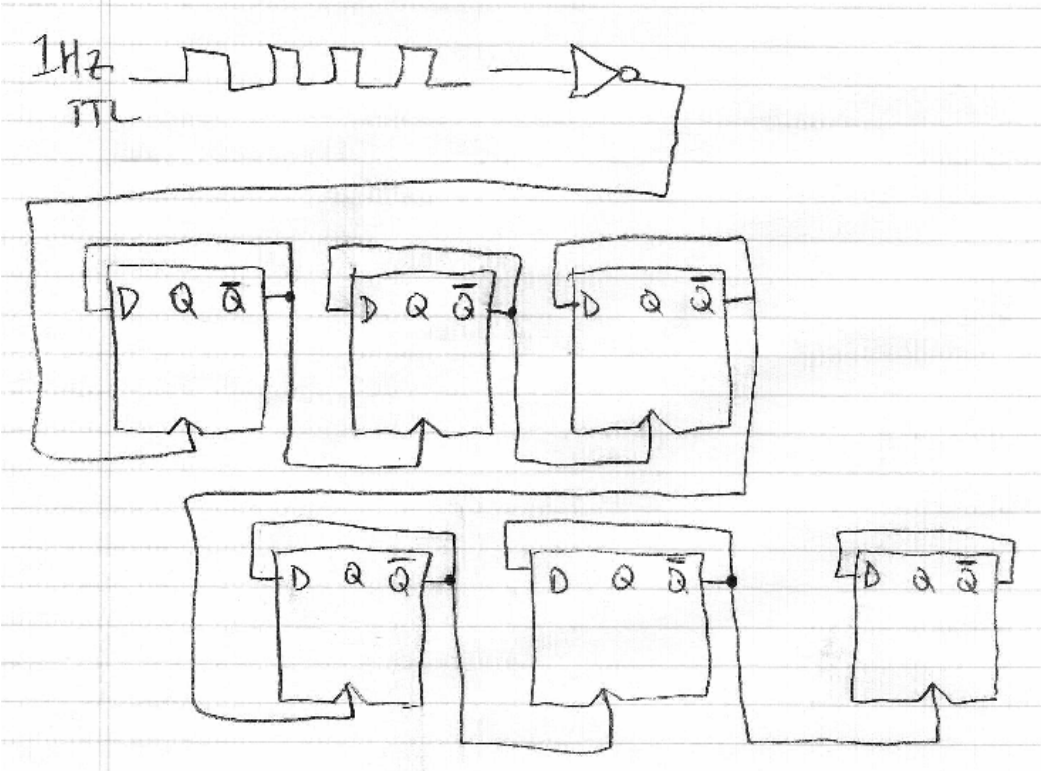
Connect each Q output to the next flip-flop's D input. The last Q goes only to the inverter-LED-resistor display. The first D comes from one of your DIP switches (or a momentary-contact switch) so that you can send a one or a zero into the shift register at each clock cycle.

Your set-up should look something like the drawing on the next page.



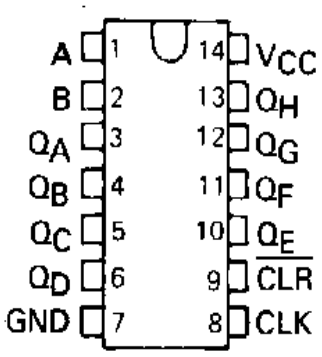
(b) Now partially rewire your D-flops to make a six-bit ripple clock, as shown below. Each flip-flop's (active low) Q* will drive its own D and will drive the next flip-flop's clock. Continue to send the six (active high) Q outputs to the six inverter-LED-resistor displays so that you can watch the action.

Notice that each successive Q counts at half the speed of its predecessor. (Using a ripple clock is actually a bad design, as we no longer have a single drummer to which everyone responds. The six bits actually change states at slightly different times (tens of nanoseconds, corresponding to the O(20ns) propagation delay of each flip-flop) – something you can probably see with the oscilloscope.)



Part 8: monolithic shift register and counter

(a) Now try a monolithic shift-register chip, the 74LS164 8-bit shift register, connected as the shift register was in Part 7. Connect both CLR* and input B to +5V and input A to your DIP or momentary-contact switch. Monitor the eight Q outputs (or at least six of them) with your inverter-LED-resistor displays.



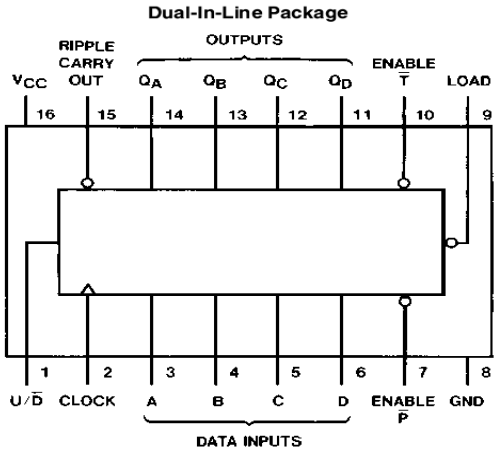
FUNCTION TABLE

INPUTS				OUTPUTS			
$\overline{CLEAR}$	CLOCK	A	B	Q_A	Q_B ...		Q_H
L	X	X	X	L	L		L
H	L	X	X	Q_{A0}	Q_{B0}		Q_{H0}
H	↑	H	H	H	Q_{An}		Q_{Gn}
H	↑	L	X	L	Q_{An}		Q_{Gn}
H	↑	X	L	L	Q_{An}		Q_{Gn}

H = high level (steady state), L = low level (steady state)
X = irrelevant (any input, including transitions)
↑ = transition from low to high level.
Q_{A0}, Q_{B0}, Q_{H0} = the level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Gn} = the level of Q_A or Q_G before the most-recent ↑ transition of the clock; indicates a one-bit shift.

(b) Now try the 74LS169 4-bit counter, driven by 1Hz clock (buffered by the inverter, as before), displaying to four inverter-LED-resistor displays. Remember VCC and GND! Connect ENABLE T*, ENABLE P*, LOAD*, and the four data inputs A,B,C,D to ground. Connect U/D* to +5V. Connect outputs Qa,Qb,Qc,Qd to your LED display. Send the 1Hz clock to the CLOCK input. See whether the outputs count.

What does the RIPPLE CARRY OUT* pin do? Why is it there? How would you connect two of these counters together if you wanted to make an 8-bit counter?



Part 9: one-shot ("monostable")

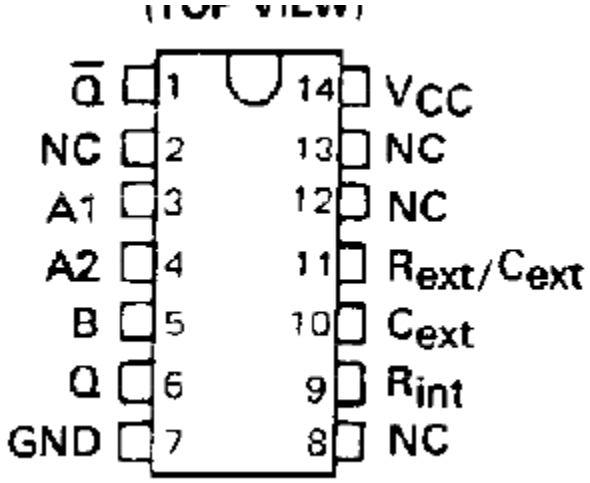
The "one-shot" is one of the few useful discrete components, as it can generate a long pulse in response to a rising edge. I have used these to keep LEDs on for long enough (e.g. 200ms) for them to be visible. Set up a 74LS121 (or perhaps 74121). Remember VCC (= +5V) and GND.

Connect A2 and B to VCC. Connect A1 to VCC through a 1K resistor. Connect Q so that it will light an LED when it is on (perhaps using your inverter+resistor+LED from before).

Connect a 33K resistor between pins 11 (Rext/Cext) and VCC. Connect a 100uF capacitor between pins 11 (Rext/Cext) and 10 (Cext).

Now momentarily ground A1 (pin 3) with a wire or a push-button switch. You should see a pulse at Q that is about 2 seconds wide, whether you keep A1 grounded for several nanoseconds or several seconds. Give it a try.

This sort of circuit can be useful for stretching out a pulse to a sufficient length for it not to be missed – e.g. by the human eye or by a device that might miss a very narrow pulse.



FUNCTION TABLE

INPUTS			OUTPUTS	
A1	A2	B	Q	$\overline{Q}$
L	X	H	L	H
X	L	H	L↑	H↑
X	X	L	L↑	H↑
H	H	X	L↑	H↑
H	↓	H	⌐⌐	⌐⌐
↓	H	H	⌐⌐	⌐⌐
↓	↓	H	⌐⌐	⌐⌐
L	X	↑	⌐⌐	⌐⌐
X	L	↑	⌐⌐	⌐⌐