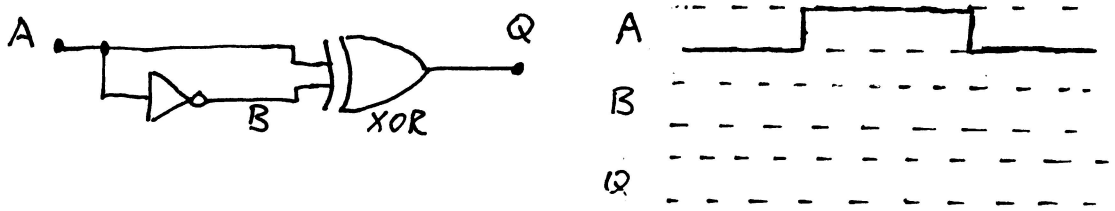
Phys. 334 **Day 7 Lecture Notes**

Assignment: Read in text: Sections 8.16-8.26 (pp. 504-527)
Read in manual: pp. 320-333

Practice problems (not to be turned in)

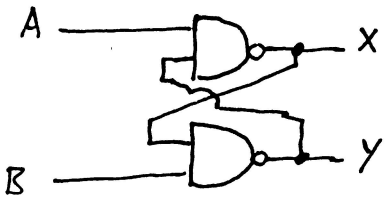
1.) problem 8.24 (p. 514) in text

2.) Given that a typical logic gate has a 10 nsec propagation delay (this is the time it takes for the output to respond to a change on the inputs), what do you expect to see on the output of the circuit below when the input pulses HIGH as shown? (Draw a timing diagram.)



Sequential Logic: These are logic circuits where the output depends upon both the present and previous inputs (ie. logic with memory). The basic unit is the flip-flop.

Basic Flip-Flop (FF): We can use either NAND or NOR gates -- use the output of one gate as an input to its partner and vice-versa:

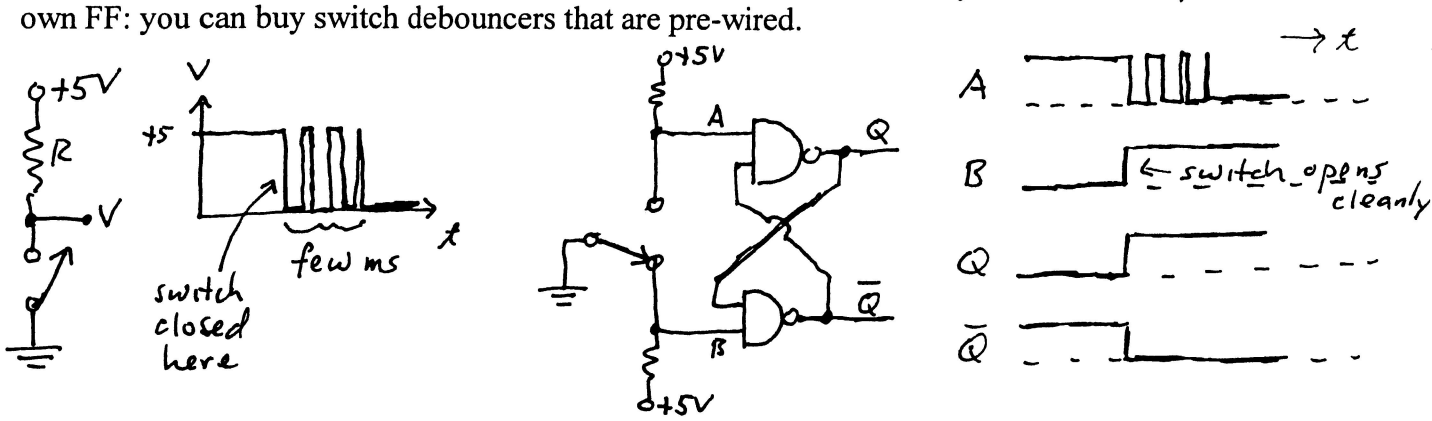


A	B	X	Y	
0	0	1	1	
0	1	1	0	
1	0	0	1	
1	1	1	0	} Two Stable States
		0	1	

When both A and B inputs are HIGH, the output has two stable states. Which state will it be in? It depends upon which input, A or B, was most recently LOW: if A and B are normally HIGH and you pulse one of them, say A LOW, then while A is LOW, the output goes to the state X HIGH and Y LOW as shown in the second line of the truth table. Because Y is LOW and is one of the inputs to the top NAND gate, when A goes back HIGH, the output stays in the state X HIGH and Y LOW. If B is pulsed LOW, the output ends up in the other stable state.

Switch Debouncer: This basic flip-flop is useful to 'debounce' mechanical switches. A serious problem with mechanical switches is that when contact is made between the two conductors, they vibrate and for a short time the contact bounces open and closed (at frequencies of kHz). This can be a disaster for high speed digital circuits where you need to switch instantly and cleanly. A solution is to use our basic flip-flop, as shown below. The idea is that the first time the switch makes contact, the FF output switches to the other stable state and stays there (because the switch bounce is not so big that it makes contact with the other

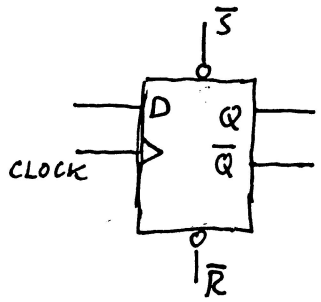
side of the switch. The timing diagram is shown below. You don't always have to build your own FF: you can buy switch debouncers that are pre-wired.



D-type Flip Flop (Latch): The most useful and common type of flip-flop are 'edge-triggered, clocked' flip-flops, of which the D-type is the most common. They are made from combinations of our basic flip-flops. The wedge on the CLOCK input signifies that the input responds only to a change in the input: without a circle, the input responds only to the rising edge of a LOW to HIGH transition, a circle on the wedge means the input responds only to the falling edge of a HIGH to LOW transition.

If the SET and RESET inputs are FALSE (as they usually are), then a trigger on the CLOCK input (the appropriate change in CLOCK state) causes whatever is on the D (data) input to be passed to the Q output. The output then remains unchanged even if the D input changes state. The output only changes if another CLOCK trigger occurs -- The D input gets passed across again. The output 'remembers' what was on the D input at the instant of the CLOCK trigger (or D gets 'latched' to Q).

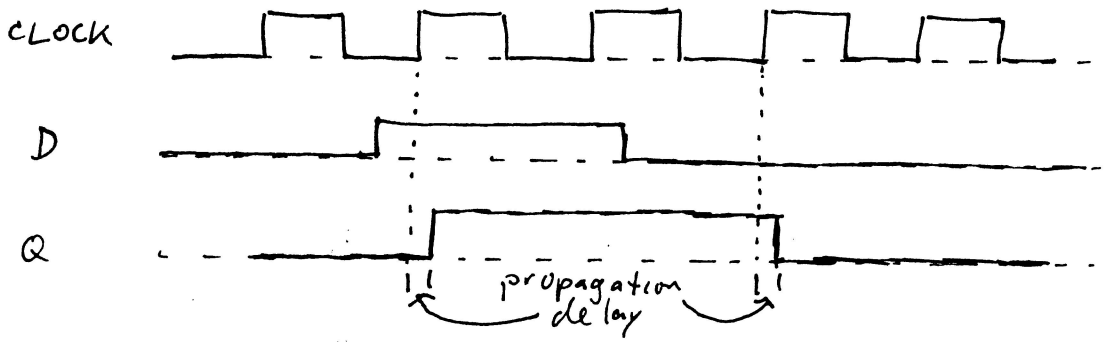
The SET and RESET inputs are used to put Q in a known state: SET 'sets' Q to HIGH and RESET 'resets' it to LOW. SET and RESET, when TRUE, cause the chip to ignore the other inputs, as shown in the truth table below.



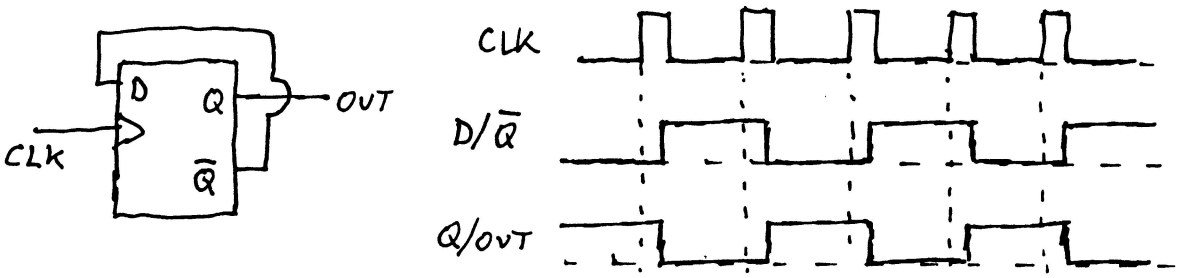
$\overline{S}$	$\overline{R}$	CLOCK	D	Q (after clock edge)
ϕ	1	X	X	1
1	ϕ	X	X	ϕ
1	1	↑	1	1
1	1	↑	ϕ	ϕ

↑ rising edge of clock transition

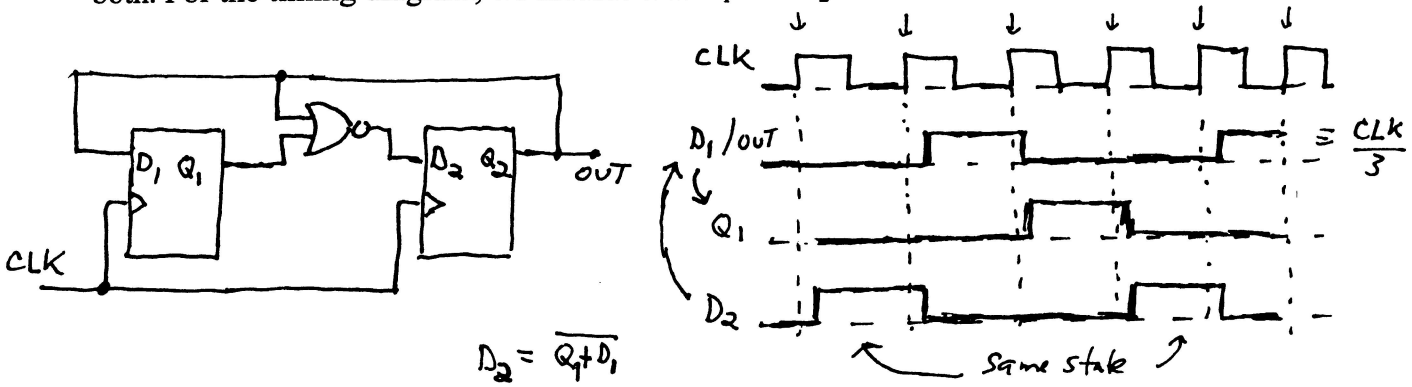
For example, when SET and RESET are FASLE, a pulse train on CLOCK and data on D will produce the timing pattern shown below. As indicated below, the value on D appears at the output Q (and Q) after a small propagation delay, typically 10 nsec (the time it takes for all of the transistors inside the chip to change their states).



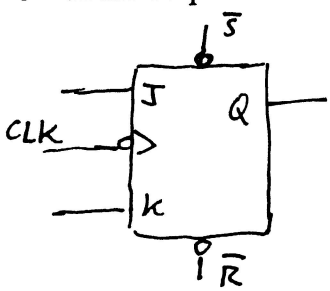
Divide by 2: We can take advantage of the propagation delay to use a D-FF to divide an input frequency (on the CLOCK input) by two. A timing diagram is shown below. Again, the value of D at the instant of the CLOCK transition is passed to Q. Because Q (and Q) change only after 10 nsec, there is no uncertainty about the value of D at the instant of the CLOCK transition. This circuit is also useful to convert an asymmetric square (unequal times HIGH and LOW) at a frequency 2f into a pure (symmetric) square wave at a frequency f.



Divide by 3 (synchronous): By using more D-ff's and additional gates, we can divide a CLOCK frequency by any integer, for example, 3, as shown below. This is called a 'synchronous' divider because both FF's act at the same time: the same CLOCK input goes to both. For the timing diagram, we assume that D_1 and D_2 both start LOW.



J-K Flip-Flop: Another common edge triggered FF, now with two inputs, J and K, is called the J-K FF. SET and RESET are the same as for the D-FF. Here, what the output does when a CLOCK transition occurs depends upon J and K: if J=K=0, the output after the CLOCK transition remains unchanged, if J=K=1, the output 'toggles' or goes to its complement, if J = K the output becomes J.

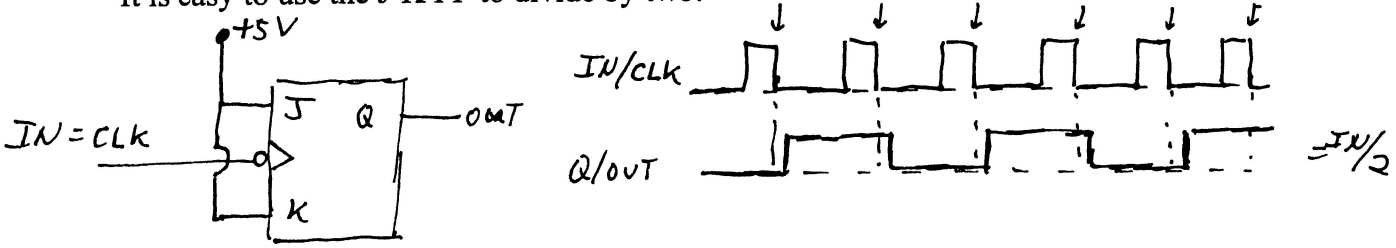


CLK	J	K	Q_{n+1}
↓	0	0	Q_n
↓	0	1	0
↓	1	0	1
↓	1	1	$\overline{Q_n}$

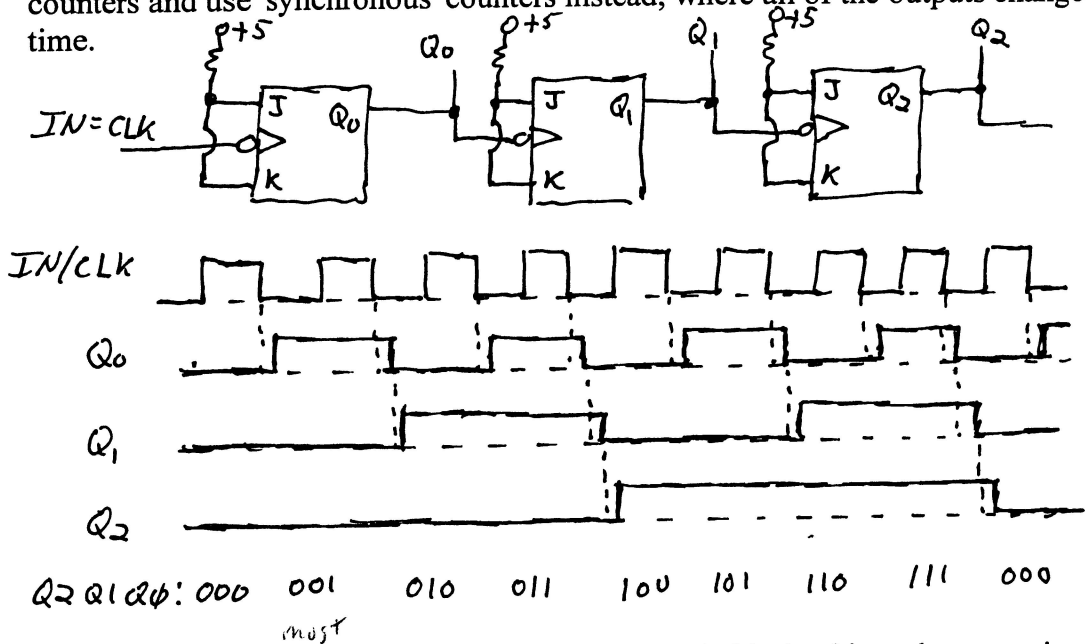
← output after n^{th} ↓

← output before n^{th} ↓

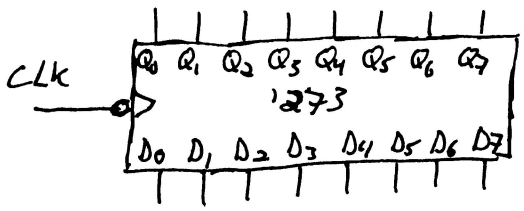
It is easy to use the J-K FF to divide by two:



Ripple Counter: By stringing together a series of N J-K FF's we can divide by 2^N, or equivalently count up to 2^N. This is called a 'ripple counter' because the outputs don't change at the same instant: first Q_0 changes and this causes Q_1 to change which causes Q_2 to change... The number of pulses that arrived on CLOCK can be read from the string of outputs: # of counts = Q_2 Q_1 Q_0, but you must be sure to read the outputs only after they have all changed. If you are unlucky and read the outputs while the signal was 'rippling' down the chain of FF's, you would get a wrong result. For this reason, it is a good idea to avoid ripple counters and use 'synchronous' counters instead, where all of the outputs change at the same time.



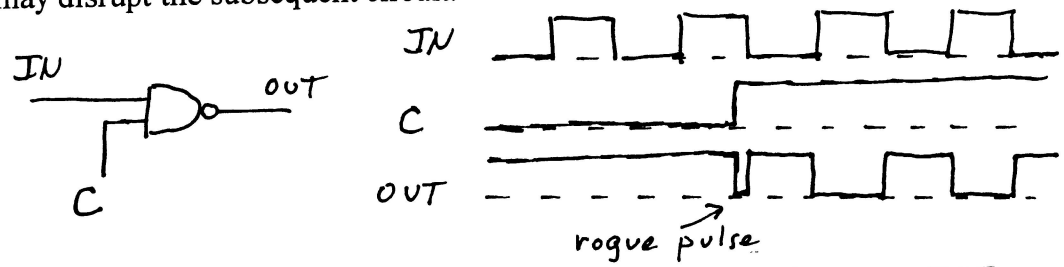
D-type Latches: The common use of D FF's is probably in chips where a series of identical D FF's are triggered by a common CLOCK pulse (called a D-type Latch). An example is the 747273 shown below. Whenever a negative-going CLOCK edge appears, whatever is on the D_n input gets passed to the Q_n output, for all n.



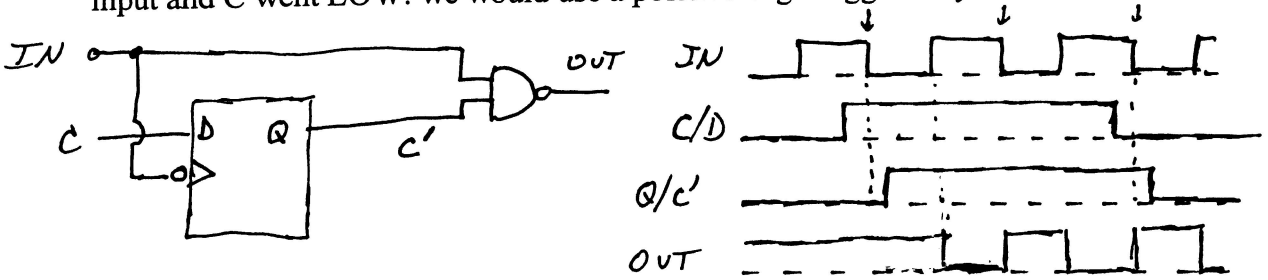
The 74373 is similar except it has tri-state outputs. There are also 'transparent latches'. These are similar to the D-type latches except that when CLOCK is in the state that preceeds its transition edge (eg LOW for a positive edge triggered CLOCK), the output = D input, ie the output changes as the input changes. When the edge trigger occurs, the last value of the output is latched until CLOCK changes state again.

Latches are useful to synchronize data. When a high speed computer CPU needs to send data to a slower device, the CPU does not slow down. Instead, it send its data at high speed and the slower device 'latches' the data with a '273 or similar chip. Now that data is stored on the outputs of the latch, allowing the slow device to process it at its leisure: when it is finished, it tells the CPU that it is ready for more data.

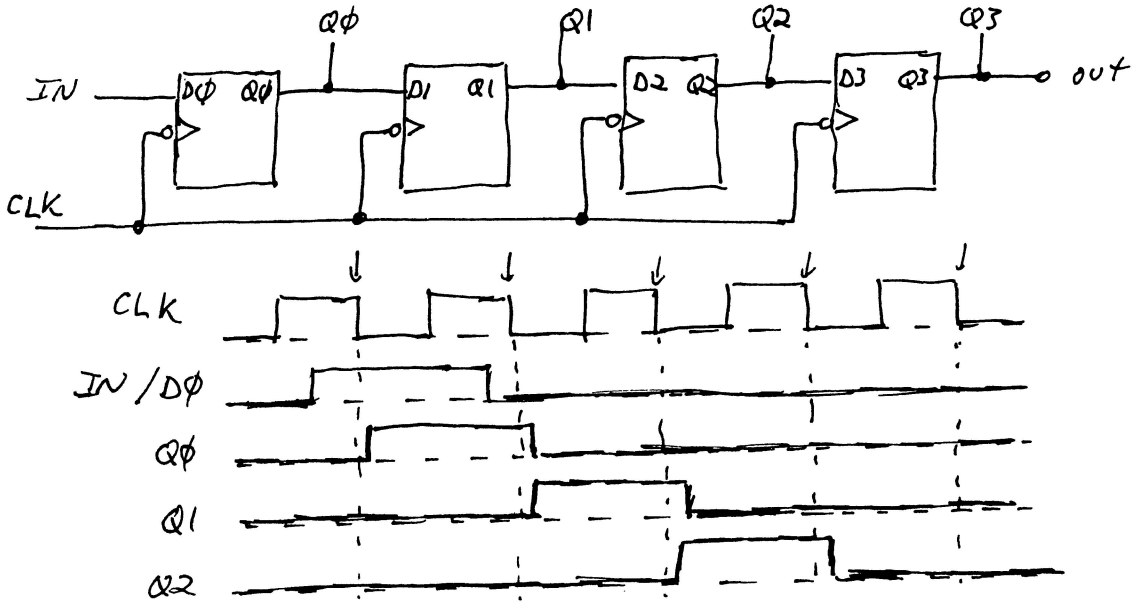
Pulse Synchronizer: If you have a pulse train that you want to pass or block via a switch, it is important to ensure that only full pulses be transmitted. Shortened (rogue) pulses correspond to a high frequency, may not be recognized as pulses by all chips, and may capacitively couple to places they should not be. For example, below we use a NAND gate as a digital switch: When C is LOW, the NAND output stays HIGH, blocking the input signal. When C is HIGH, the output is the complement of the input. If you are unlucky and turn the switch on at the time shown below, only a fraction of the first input pulse will get through the NAND and may disrupt the subsequent circuit.



The solution to this problem of generating shortened pulses is to use a D FF as a synchronizer: Run C into the FF D input, and use your pulse train to both CLOCK the FF and the NAND. We use a negative edge triggered FF because the rogue pulse was generated when the input went HIGH (and the negative edge guarantees that C' can go HIGH only when IN is LOW). Had we used an OR or NOR gate for our switch, the rogue pulse would be generated when the input and C went LOW: we would use a positive edge triggered synchronizer in that case.



Shift Register: Here we string a series of D FF's together, with $Q_n \rightarrow D_{n+1}$, and all of the CLOCK inputs tied together. This makes the device 'synchronous': all of the outputs change at the same time. As seen below, at each negative edge of CLOCK, the data on the outputs is 'shifted' one FF to the right, and new data from IN is shifted in from the left.



Shift registers are useful to convert serial data to parallel data. Serial data is data that is transmitted on a single line as a series of pulses (such as modems that use the telephone lines). Parallel data is groups of 8, 16, or 32 lines where each line represents a bit and all lines are processed at the same time (in parallel) as occurs in computers. To change serial data from your modem to parallel data that your computer needs, you connect the serial line to IN of a shift register and use the BAUD rate CLOCK pulse to clock in the serial data. After 8 or 16 clock pulses (one or two bytes of serial data), you read the Q outputs of the shift register for the data in parallel form: Q_0 holds the last serial bit sent, Q_N the first.