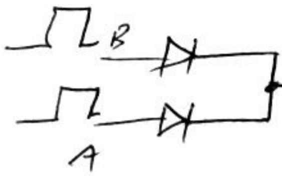
Digital logic

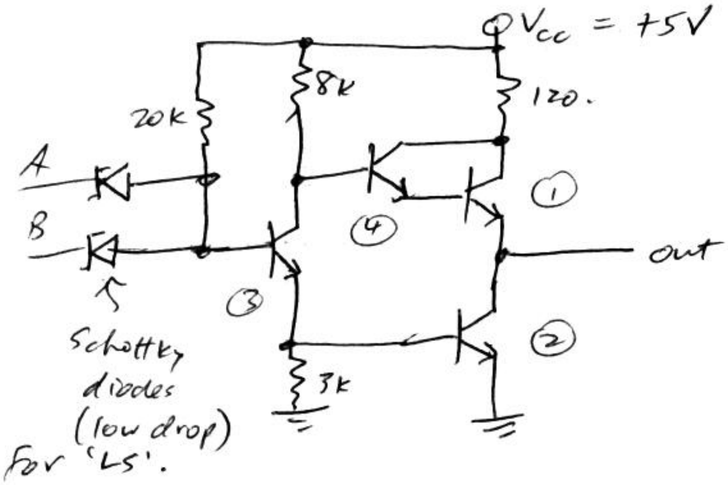
From now on we'll look at saturated switching.
(could use diodes, but no gain —



)

Two common (& related) families are TTL & CMOS.

Basic TTL output is push-pull: Here's a "NAND"



74LS00 or similar

If A, B $\gtrsim$ 2V, (3) conducts. This turns on (2) and turns off 1 $\Rightarrow$ output $\lesssim$ 0.4 V.

If A <u>or</u> B $\lesssim$ .8V, (3) turns off, (1) conducts, (2) doesn't and output ~ 4 V or so.

CMOS has same basic philosophy but swings closer to the power supply rails.

Logic levels are all formalized -- see p. 292.

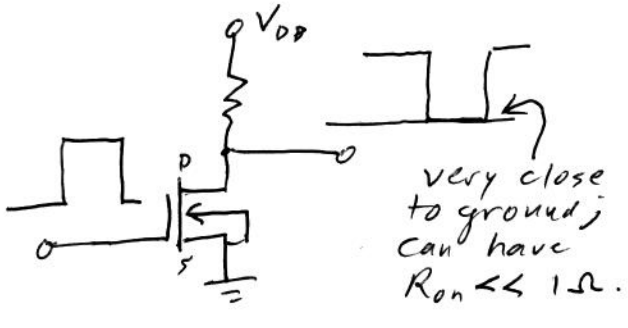
For "classic" TTL:

<u>input</u>: L < 0.8V
H > 2.0V

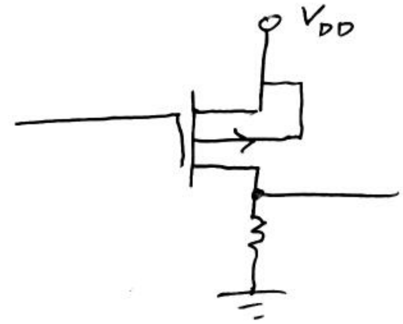
<u>output</u>: L < 0.4V
H > 2.4V } can drive 10 loads "fanout," more for "buffers"

In newer logic chip families, CMOS has almost entirely replaced bipolar junction transistors.

We already looked at an n-channel MOSFET as an analog switch. For saturated digital switching we can use

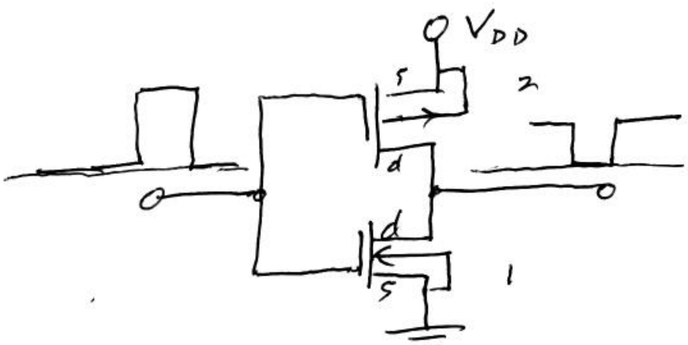


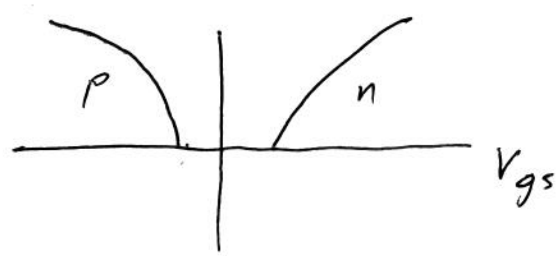
n-channel



p-channel

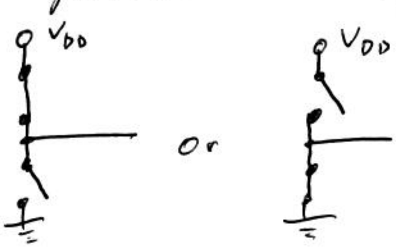
For fast logic switching, it's best to combine both approches, yielding the basic <u>CMOS</u> <u>inverter</u>:





Input grounded: $V_{gs1} \cong 0$ (off), $V_{gs2} \cong -V_{DD}$ (on)

Input positive: $V_{gs1} \cong V_{DD}$ (on), $V_{gs2} \simeq 0$ (off)



With additional circuitry, logic thresholds can be adjusted as desired — anything from 1.2 V logic to 15 V logic is available.

Some of the more commonly encountered logic "families" are:

pure TTL ; 74xx (obsolete)
pure CMOS : 40xx (+typ. 10-15 V)

TTL variations ;

74LSxx "low-power Schottky", common
74ASxx "advanced Schottky"
74F xx "fast"

CMOS variations :

74Cxx V_{cc} = 3-15 V, not really TTL-compatible
74HCxx V_{cc} = 2-6 V
74HCTxx V_{cc} = 4.5-6V fully TTL-compatible
74VHCxx V_{cc} = 2-5.5V TTL inputs OK even if $V_{cc} \ll 5$V!
↓ many others!

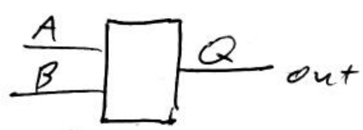
<u>Logic and truth tables</u>

Can write T = H or F = H
F = L T = L

normal — "inverted" or "negative-true" or "assertion level" logic

So usually, 0 = L = F, 1 = H = T.

<u>Two-input gates</u>



A	B	Q
0	0	
0	1	
1	0	
1	1	

4 entries, each either 0 or 1
$\Rightarrow 2^4 = 16$ possibilities

Only four are really common, though.
Some, like 0000, are entirely useless.

It turns out that any logical operation can be constructed from just <u>one</u> type, if it's a NAND or a NOR!

AND, AB ($A \cdot B$, sometimes)

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

NAND, $\overline{AB}$ ← not-and

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

OR, $A+B$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	~~0~~ 1

NOR, $\overline{A+B}$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

(also XOR, XNOR)

Names apply to $0 = F$, $1 = T$.
"Negative" Logic — or "assertion level" logic:
What if we reverse this? Then a NAND gate looks like,

$\underset{\sim}{A}$	$\underset{\sim}{B}$	Q
T	T	F
T	F	F
F	T	F
F	F	T

← This is a negative-logic NOR function.

Works for AND and OR also. This is due to de Morgan's theorem: $\overline{AB} = \bar{A} + \bar{B}$, $\overline{A+B} = \bar{A}\bar{B}$

$$= \overline{\underset{\sim}{A} + \underset{\sim}{B}}$$

$$\overline{\overline{\bar{A} + \bar{B}}}$$

$$\parallel$$

+ logic	− logic
NAND	NOR
AND	OR
NOR	NAND
OR	AND

etc.

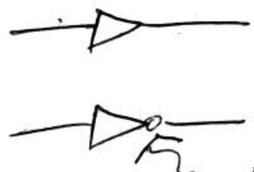
$\leftarrow \overline{AB} = \bar{A} + \bar{B}$ ①

$\leftarrow AB = \overline{\bar{A} + \bar{B}}$ (1)

$\leftarrow \overline{A+B} = \bar{A}\bar{B}$ ②

$\leftarrow A+B = \overline{\bar{A}\bar{B}}$ (2)

Can switch with an <u>inverter</u>:



buffer

inverter

polarity switched

<u>Assertion-level logic notation</u> (not in Meyer)

Convention: in normal logic show inversion (if any) at output. In neg. logic always draw inversion at input, then at output if needed. This is "assertion-level logic."

So we have,

+ logic — − logic

buffer

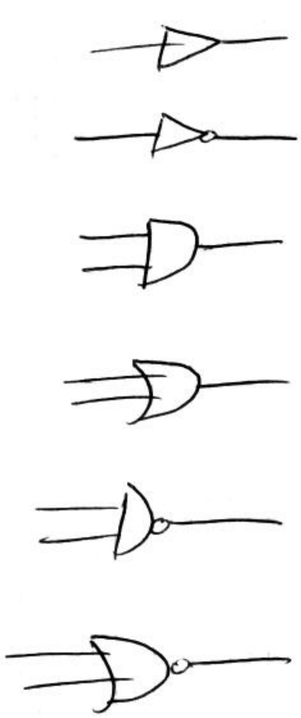
inverter

and

or

nand

nor

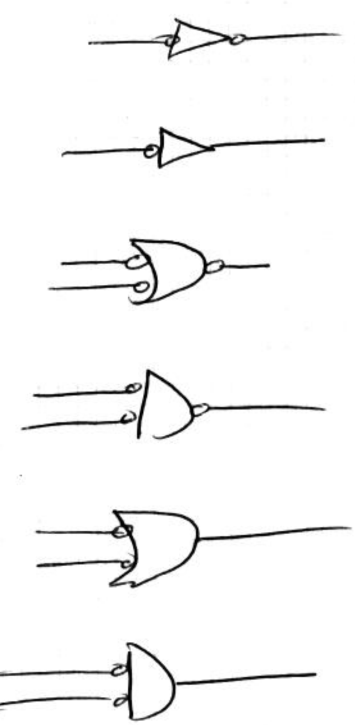


or

and

nor

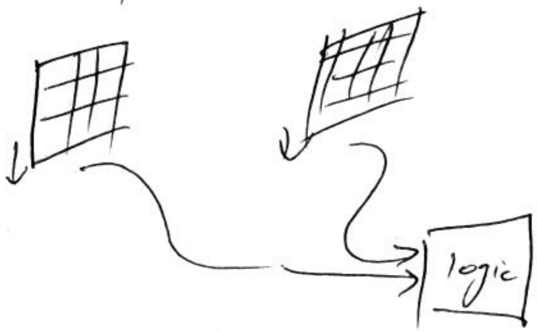
nand



"Universal gates" NAND, NOR can each be used to build <u>all</u> 2-input fcn's ; hence all n-input fcns.

(inverter: )

Example: security system



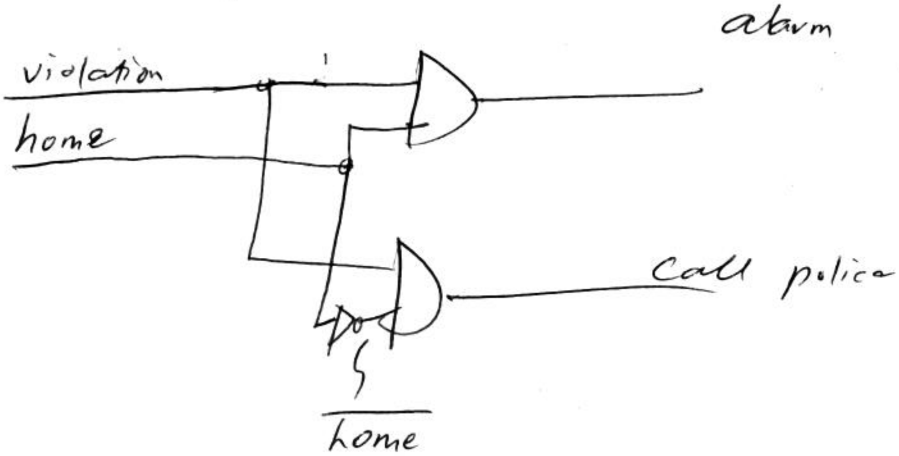
If (violation)(at home) sound alarm
(")(not ") call police
$\overline{\text{(violation)}}$ do nothing

So we need

$A_{(\text{viol})}$	$B_{(\text{home})}$	Q
0	0	0
0	1	0
1	0	1
1	1	2

oops!

Better design: 2 in, 2 out:



How to find violation? easiest to close switch to ⏚
(just ⏚ wire / entrance)

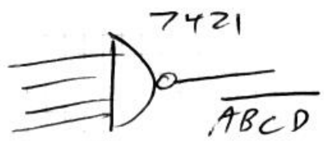
← want (+) if $\overline{V1}$ or $\overline{V2}$ or $\overline{V3}$ - -

need (−) logic <u>nor</u> function

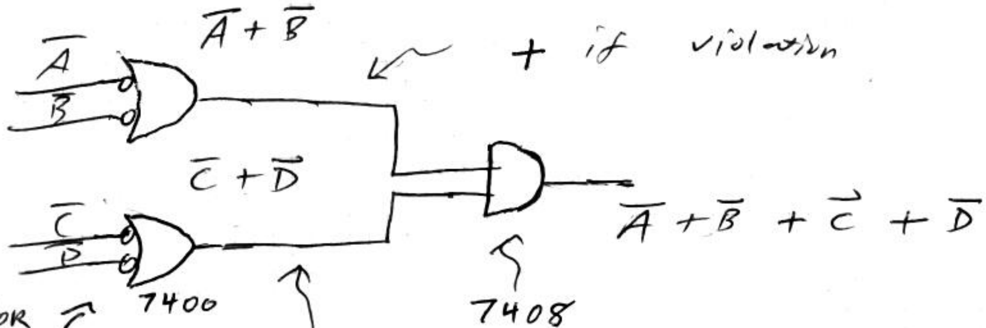
Other reasons for using (−) true --

1) TTL pulls high <u>easily</u> (one pullup/many circuits)
2) power consumption (really same)

Anyhow, (−) NOR is same as nand:



available directly, or



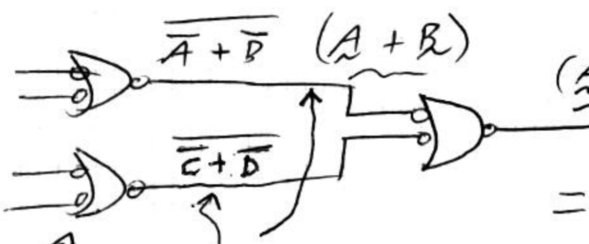
+ if violation

(−) NOR = (+) NAND

+ if violation

$\boxed{\text{or}}$

associative – can omit parentheses



$(A+B)+(C+D) = \overline{\overline{A}+\overline{B}+\overline{C}+\overline{D}}$

$=$ <u>low</u> if $\overline{A} + \overline{B} + \overline{C} + \overline{D}$

(wrong sign -- needs inversion)

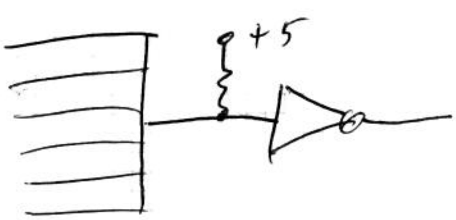
(+) logic AND

(−) if violation

=

still needs inversion, though. Perhaps can change design to use (−) logic output?

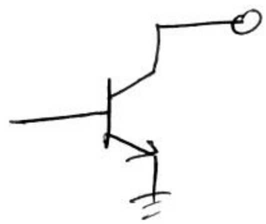
Or, <u>wired or</u>



Systematic logic design -- see Karnaugh maps, in texts. (really not much used, as optimization not very important in μ computer era.)

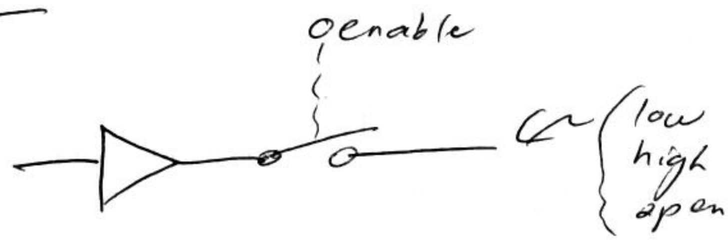
Variations:

1) open collector buffers -- lamps, slow buses, ...

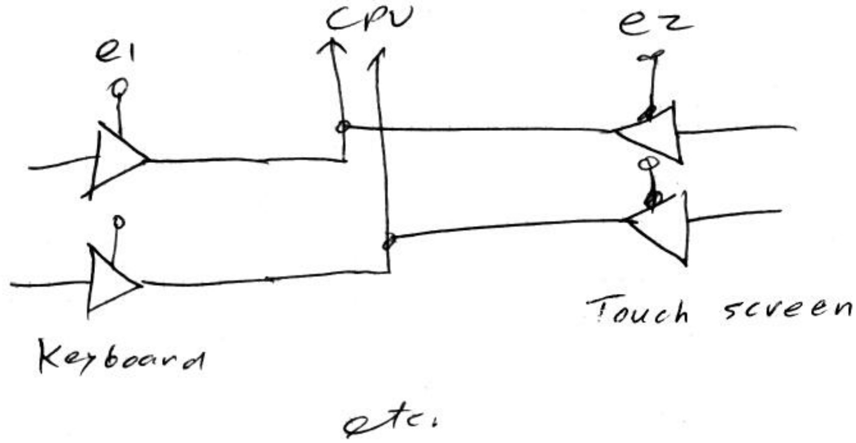


2) Tri-state logic

really



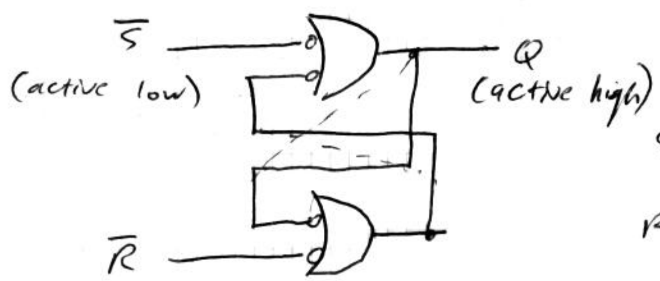
Much used on computer data buses, etc:

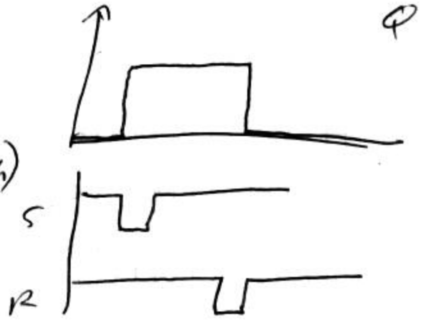


etc.

Next: flip-flops & counters.

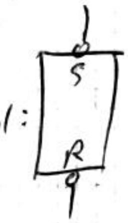
1. Simple S-R latch:





(activate only S or R)

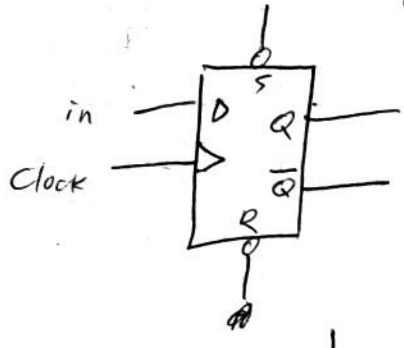
Symbol:



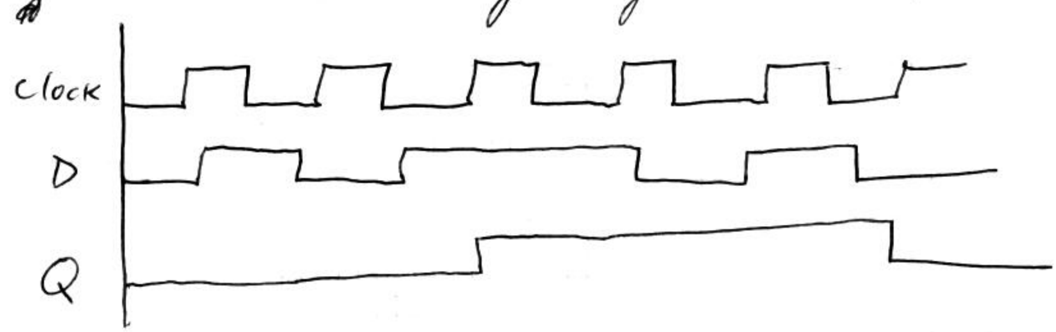
2. Clocked gates

A. D-type flip-flop: (74xx74 is classic)

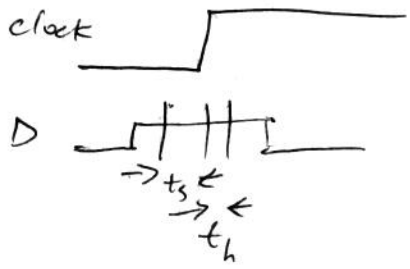
Idea is to combine S-R latch with a synchronized scheme -- examine input data when clock signal comes along (only).



1) S, R override everything
2) If S, R HI, then D is transferred to Q on rising edge of clock.



For 7474, $t_{setup} = 20$ ns
$t_{hold} = 5$ ns (prop. delay ~ 15 ns)

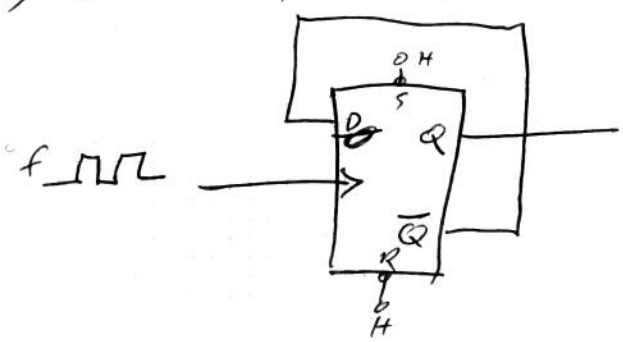


⇓

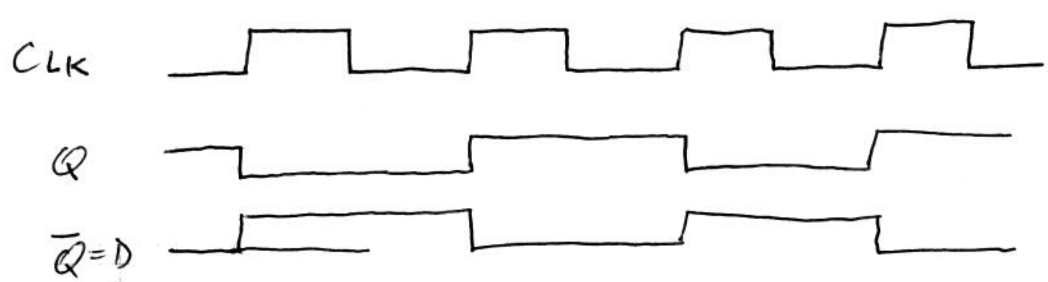
fast -- 1) decouple pwr supplies
2) clock must be clean

Many applications:

1) Divide-by-two



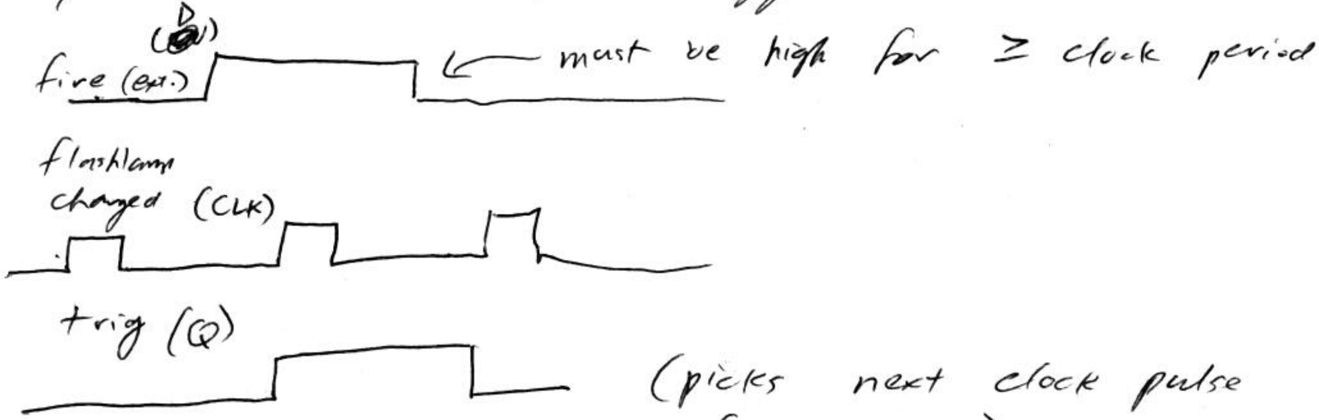
($\overline{Q}$ does not change until 20 ns after clock, so hold time ok.)



f/2 on "toggle"

cascade for divide-by-2^n
Combine with gates for divide-by-n.

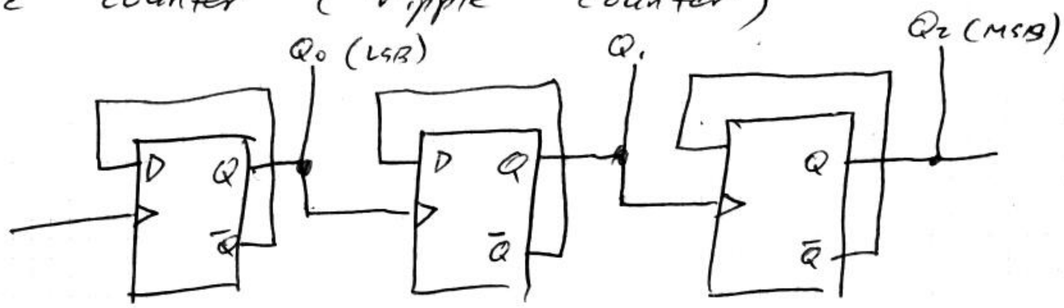
2) Synchronizer : ex: laser trigger

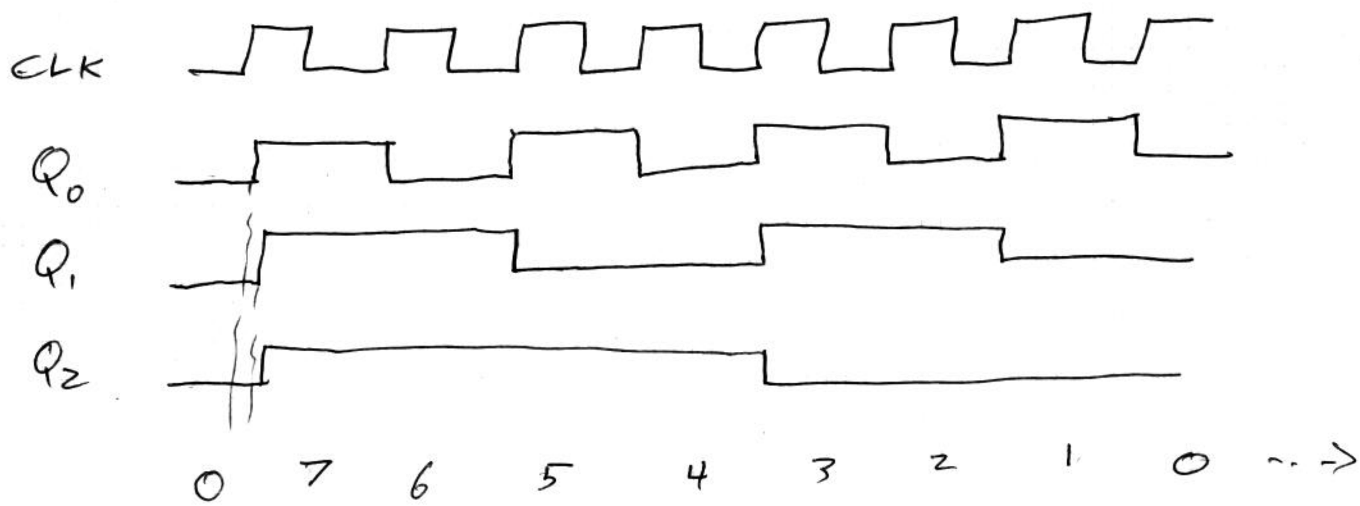


(picks next clock pulse after request).

Alternative -- select part of a pulse train - reverse D, CLK.

3) Simple counter ("ripple" counter)





(See H&H text for an up-counter -- use (-) edge or use $\bar{Q}$ outputs.

Note ripple effect -- 20ns delay per stage.

B. At this point, we need to learn more about how to <u>represent numbers</u>.

1, Binary (obvious)   1 = T   (can be H or L)
0 = F

64 32 16 8 4 2 1
0 1 1 1 0 0 1 = 32 + 16 + 8 + 1 = 57 dec.

This is just fine, but it's hard to think this way. Thus we group things (at least for humans) in 4-bit nibbles.

2. Hexadecimal

|0011| |1001| = 57 d

3   9   = 39 h

How to handle 10-15:

10	A
11	B
12	C
13	D
14	E
15	F

So 1100 = C h, etc.

(older version -- octal)

3, BCD -- sometimes must display in decimal.
So we throw away A-F --

|xxxx| |xxxx|
0-9 only   0-9 only

(Binary-coded decimal)

4. <u>Signs</u> :

What if we need (−) numbers?

a) offset binary -- just subtract ½ fs. (DAC's, scopes, etc.)

b) 2's complement :

For most work.

To make a negative #,

1) take complement $1 \rightleftarrows 0$

2) add 1.

For example, −7 is :

7: 0111

1's comp: 1000

2's comp: 1001

−2 is:

2: 0010

1's comp: 1101

2's : 1110

9: 0 1001

10 0110

−9 1 0111

MSB gives <u>sign</u>

addition is "normal"

$$-7 + -2 = \begin{array}{r} 1001 \\ 1110 \\ \hline 1\ 0111 \end{array}$$

overflow! we get +7! (but this <u>is</u> −9 in a 5+ bit rep.)

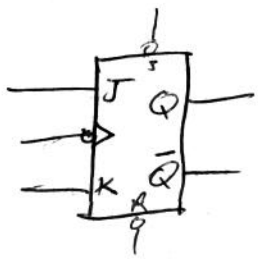
$$-7 + 2 = \begin{array}{r} 1001 \\ 0010 \\ \hline 1011 \end{array} = -5$$

$$-7 + 7 = \begin{array}{r} 1001 \\ 0111 \\ \hline 1\ 0000 \end{array}$$

overflow

C. For a better counter, use <u>JK-type flip-flop</u>

74xx73, 74xx107, 74xx112 ...



Often (−) edge triggered. Behavior depends on <u>2</u> inputs ...

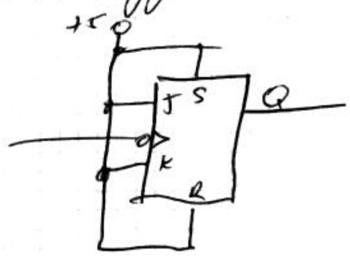
J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

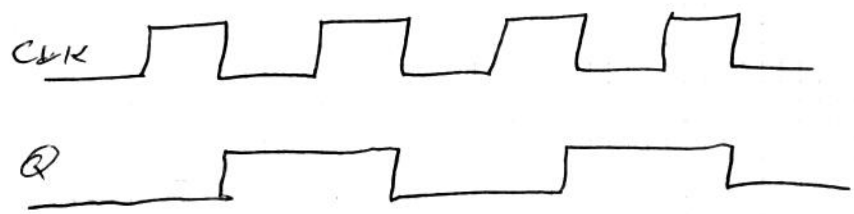
0, 1 } $Q \leftarrow J$ at next clock if $K = \bar{J}$

$\bar{Q}_n$ ← toggles

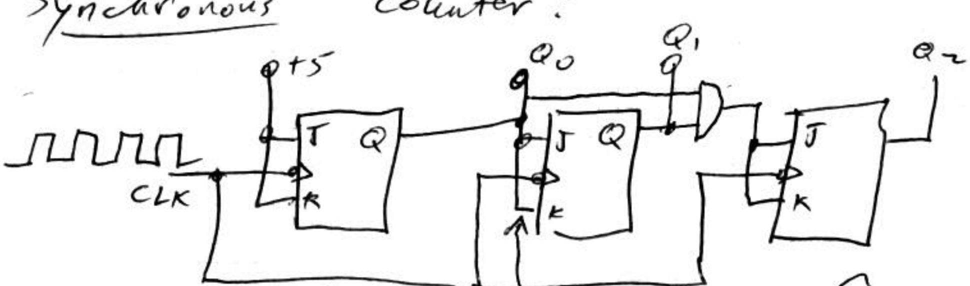
For older types, J, K must not change when clock is high!

Anyhow, toggle mode gives a trivial divide by 2:



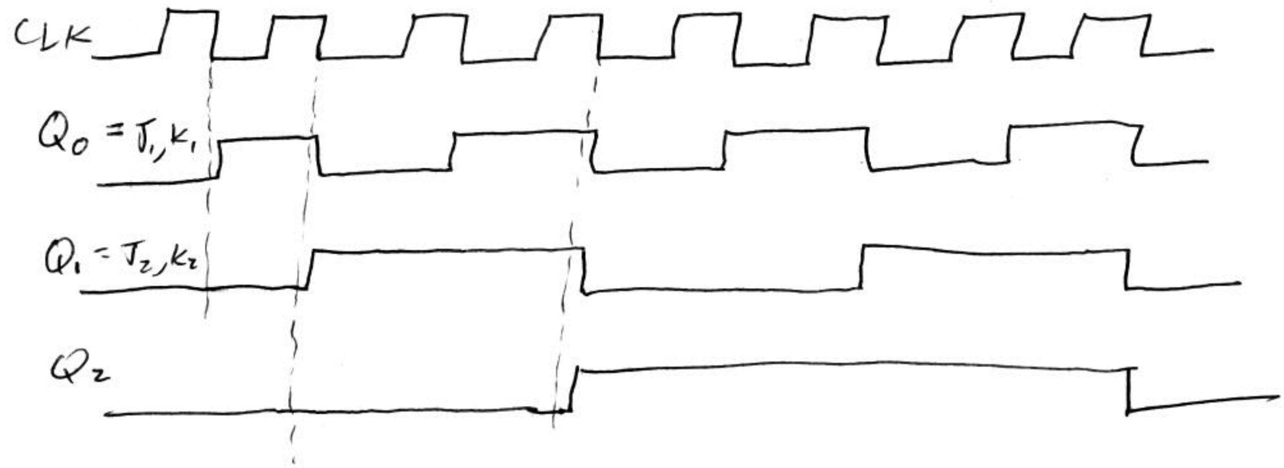


<u>Synchronous</u> counter:



divide-by-two

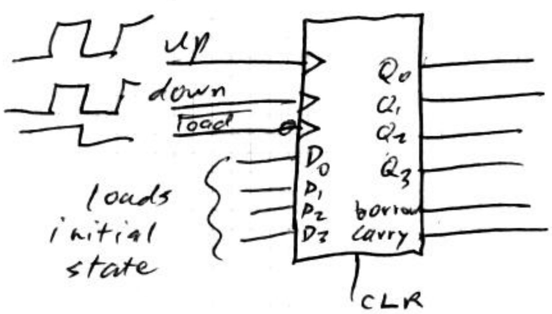
alternates between $Q_{n+1} = Q_n$, $\bar{Q}_n$



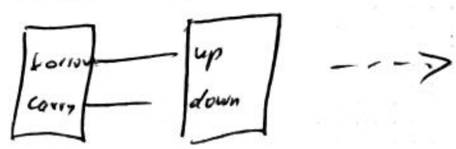
State 0 1 2 3 4 5 6 7 0 - - ->

Counter chips integrate these functions.

Ex: 74xx193

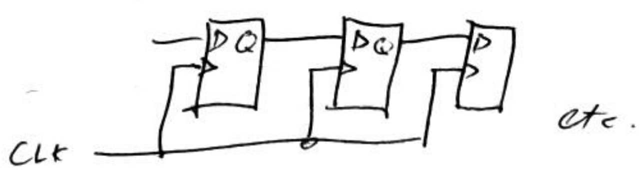


to cascade:



BCD version is 74xx192 -- counts 0-9

Shift register another important variation



let's say start at | 1 0 1 1 0 0 |

Hold D_0 low: | 0 1 0 1 1 0 | Clk 1

| 0 0 1 0 1 1 | Clk 2

| 0 0 0 1 0 1 | ⋮

Divides a number by 2, or shifts a pattern. (CCD cameras, etc.)

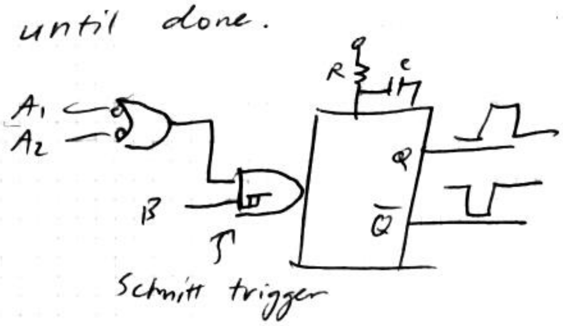
One-shots:

Sometimes we do need a variable delay that starts immediately. For this a variety of chips are available.

Classics: 1) 555 timer (a relaxation oscillator) & variants

2) TTL one-shot or "monostable multivibrator"

74(LS)121 is a nice one - it's "non-retriggerable": once started output pulse, ignores inputs until done.



triggers on:

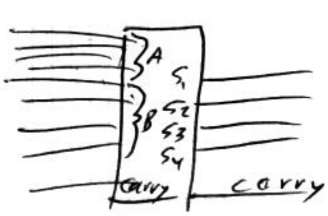
A_1	A_2	B
L	X	↑
X	L	↑
↓	H	H
H	↓	H

Some variations have built-in counters, too. Good for long delays (like warm up timers).

Decoders, Multiplexors, adders, etc.

Return now to gates. Clearly there's a need for more than 2-input NAND & NOR. We'll see a very general solution with PAL's & memory, FPGAs, etc. But a lot of MSI stuff is available ---

1) 4-bit adder:



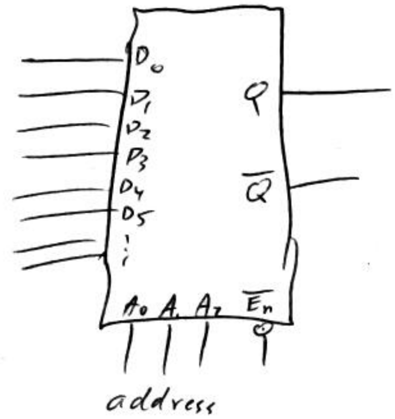
ALU is generalization (shifting, subtraction)

Multipliers & even FFT's also available (latter are really clocked logic)

Variation: magnitude comparators -- outputs $A>B$, $A=B$, $A<B$.

2) Multiplexers -- Select from many inputs:

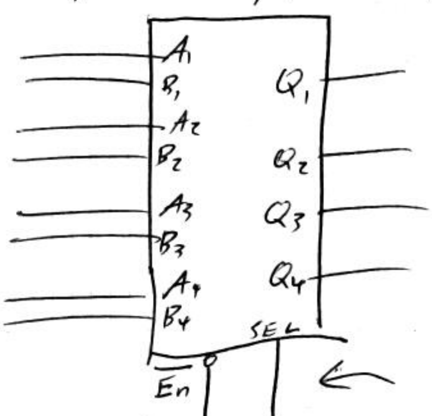
1 of 8:



A_0	A_1	A_2	Q
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
			⋮

"quad 2-input select" is sort of opposite extreme, a 2-input multiplexer (quad version)

"4PDT switch"



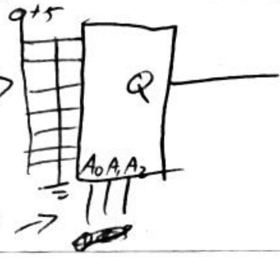
0: $Q_1 = A_1$, $Q_2 = A_2$ ⋮ $Q_n = A_n$

forces Q_n = low

1: $Q_n = B_n$

An n-input multiplexer can realize any n-entry truth table -- connect inputs as needed:

Connect L or H as needed



inputs

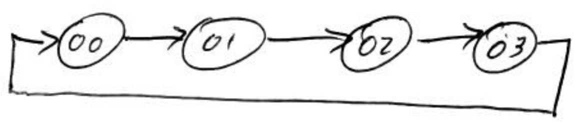
Better yet, with an additional inverter the multiplexer can realize any 2n-entry truth table. If the n+1'st address bit is X, the output for each $A_0 \cdots A_n$ must depend on B as one of H, L, X, or $\bar{X}$. So, just wire each input accordingly. See PS #5 for more.

The extension of this to multi-bit outputs can be accomplished with a memory chip -- for each address $A_0 \cdots A_n$, it produces an arbitrary output bit pattern $Q_0 \cdots Q_m$. The information can be interpreted in many ways -- as logical instructions, as data, as codes for future device states, or as addresses for subsequent operations.

Various other encoders and decoders exist too --- you've seen a 7-segment decoder in the lab.

There are also many types of programmable logic and gate arrays – PAL, PLA, FPGA, ···. Some are very fast and have $> 10^8$ transistors!

In designing complex sequences, state diagrams can be very helpful. For a 2-bit counter,



It's important to see if things can get stuck in "forbidden" states.