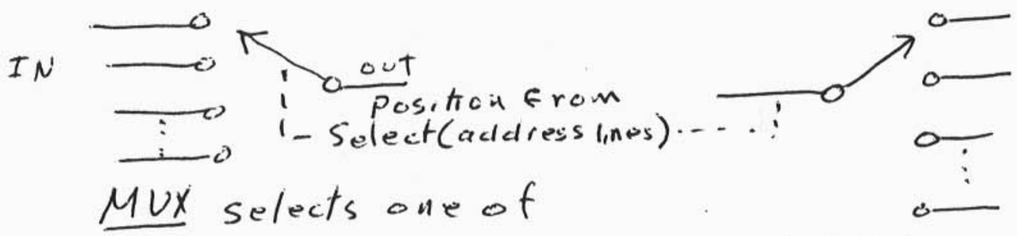
Demultiplexer

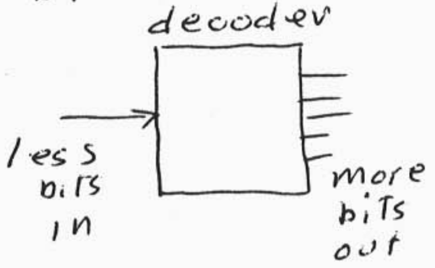
Opposite of MUX.



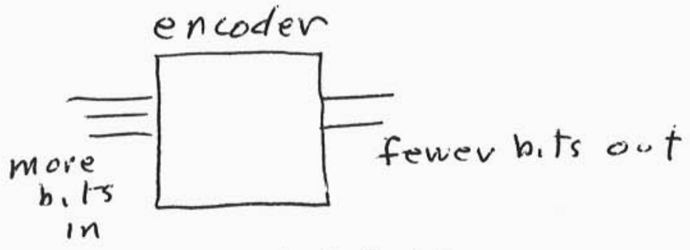
MUX selects one of many inputs

DMUX connects one input to one of many outputs.

Note MUX's and DMUX's are special forms of encoders and decoders which are circuits characterized by



e.g. a BCD to Decimal decoder (form of DMUX)

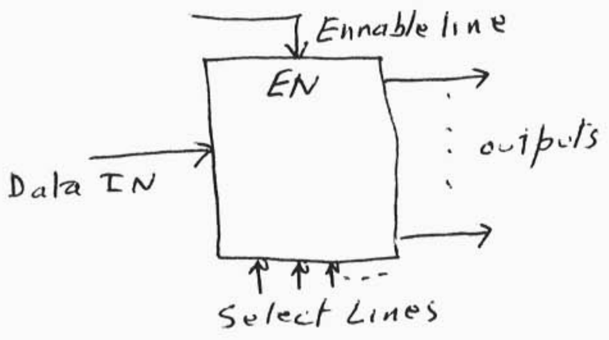


e.g. 8 bit # in
2 bits out to say input # is either prime or divisable by, say, 7.
(form of MUX)

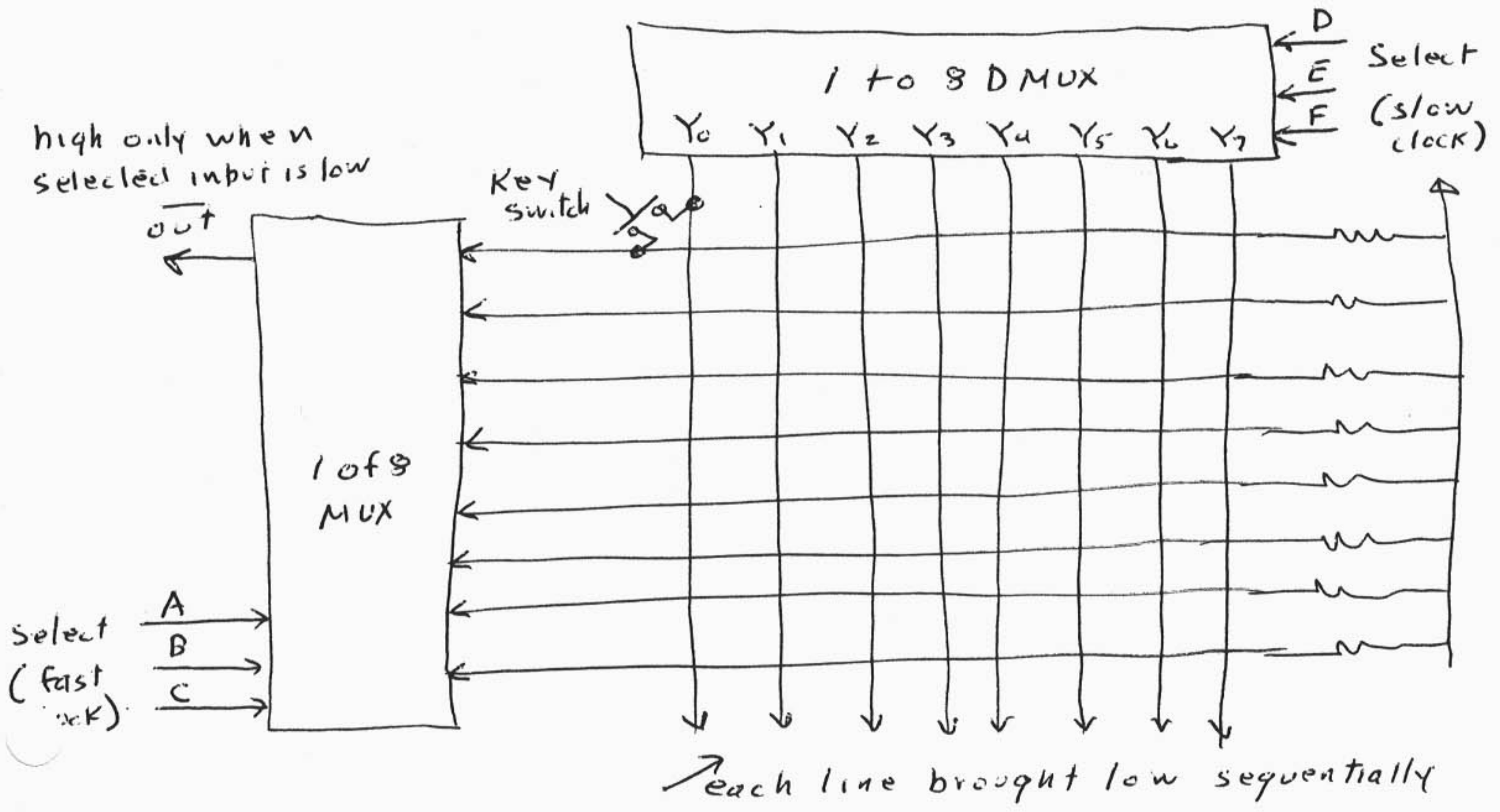
Representative DMUX's

74155	dual 1 to 4
74138	1 to 8
74154	1 to 16

Symbol:



An application:



1 to 8 DMUX
D
E
F
Select (slow clock)
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7
high only when selected input is low
$\overline{out}$
Key switch
1 of 8 MUX
Select (fast clock)
A
B
C
each line brought low sequentially

One Key switch at each row/column intersection.

DEF selects one output of the DMUX and brings it low so that column is low.
If a key has been depressed then one of the rows is connected to that column.
Scan (rapidly) the rows to find which row/column key has been depressed (when find it $\overline{out}$ will go high).
Then ABCDEF (6 bits) uniquely define the key.
Hence have a 64 key pad encoder here.

Sequential Circuits - Basics

Circuits studied so far have been "combinational logic circuits" - circuits whose output(s) is determined solely by its present inputs. What has gone on in the past has no bearing on what is happening now.

Sequential logic circuits have outputs which depend not only on present inputs but also, to a greater or lesser extent, on what has happened in the past, i.e. they have memory in one form or another. Some examples of applications include:

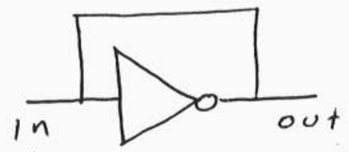
1. Latching; involves acquiring and holding on to states of a circuit at a particular time.
2. Parallel to serial and vice versa conversion.
3. Serialization of operations.
4. Counting of events
5. Sequencing of events.

The "Latch"

For comb. logic there are a small # of primitive logic gates (just one?) which can be combined and packaged to implement higher order functions.

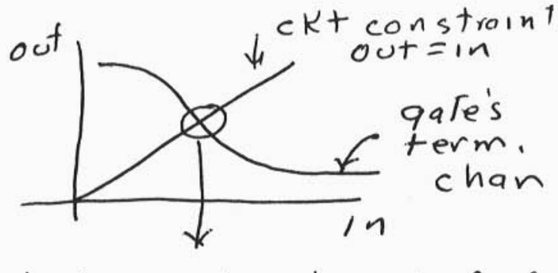
Same thing is true for sequential circuits. The primitive device is the "latch", which is sometimes called a "bi-stable".

Consider:



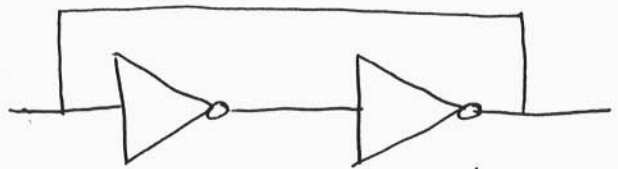
one inverter + f.b.

Two constraints



condition for both constaints satisfied.

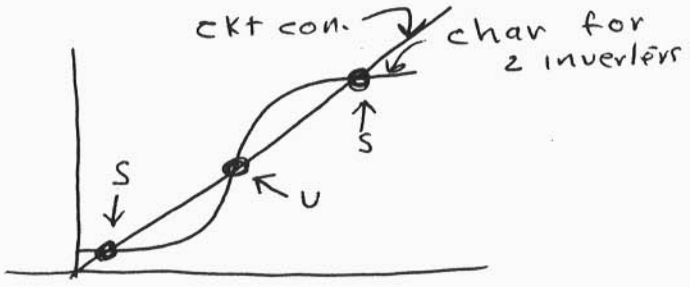
Now consider:



stays 1 0 1

2 inverters + f.b.

or 0 1 0



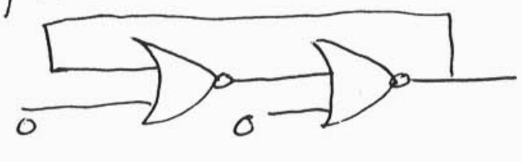
condition both constraints satisfied yields 3 pts, two of which are "stable" and one is "unstable".

Stable ⇒ ckt can remain there indefinitely until told to go to other S pt.

Unstable ⇒ ckt cannot stay there – will oscillate about this pt: topic for a more advanced electronics course.

As shown only way to change state is to bring appropriate point to +5 or gnd.

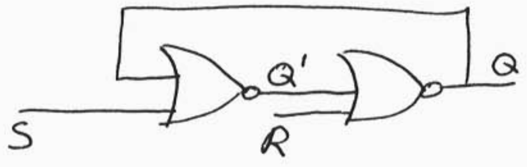
More practical implement the inverters with NOR or NAND gates



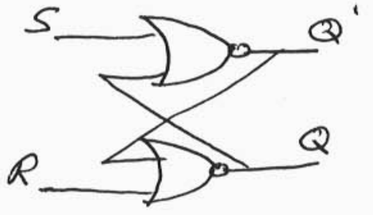
$(\overline{A+B} = \overline{A} \text{ if } B=0)$

Now make use of the inputs:

NOR type latch:



usually drawn as



Note have feedback: characteristic of sequential but not of comb. logic ckts.

NOR gates impose conditions on the 4 quantities; i.e. must satisfy:

$$Q' = \overline{S+Q} \qquad Q = \overline{R+Q'}$$

Check all possible states to determine those which are self-consistent: e.g.

S	R	Q	Q'	
0	0	0	0	← is $Q = \overline{0+0} = 1$? No good ×
0	0	0	1	← is $Q = \overline{0+1} = 0$, and $Q' = \overline{0+0} = 1$? } Yes - self con state

By going thru all possibilities end up with table:

S	R	Q	Q'	
0	0	0	1	} idle
0	0	1	0	
0	1	0	1	} reset
1	0	1	0	} set
1	1	0	0	} dissallowed

Comments re table:

1. States marked "idle" correspond to initial ckt with only inverters: outputs may be 1,0 or 0,1 depending on history.

2. If S, R = 0, 1 then Q is forced to 0 and the latch is said to be "Reset".

3. If S, R = 1, 0, then Q is forced to 1 and the latch is said to be "Set".

4. The condition where S = 1 = R is not allowed, although if it obtains both Q and Q' are both forced to 0. Not allowed since if were to go to idle state (S = 0 = R) then can't predict if Q, Q' would be 1, 0 or 0, 1 as determined by intermediate stage S, R = 0, 1 or 1, 0.

5. Idle state: AKA resting state, inactive state, do nothing state, memory state, or quiescent state.

6. Except for dissallowed state, Q and Q' are complements and can be written as $Q, \bar{Q}$, which is commonly done (with caveat).

Characteristic table:

Previous table can be simplified a bit by writing it as:

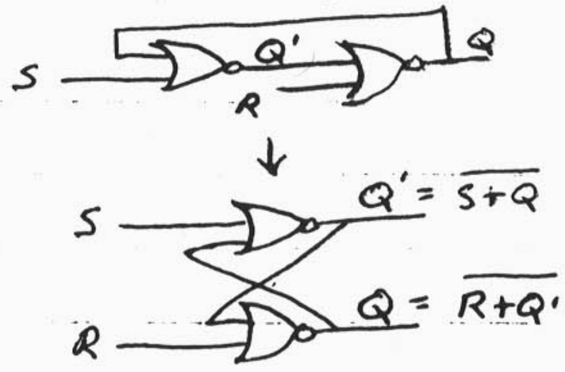
S	R	Q_{n+1}	
0	0	Q_n	no change idle,
0	1	0	reset
1	0	1	set
1	1	not allow (0, 0)	

S = set input, R = reset input
Q_n = output before input change
Q_{n+1} = " after " "
Assumes both Q and $\bar{Q}$

NAND type latch:

Fairly obvious that instead of using NOR with 1 input 0 as an inverter, could use NAND with 1 input 1 as an inverter. Dual development follows, summarized as →

NOR Latch

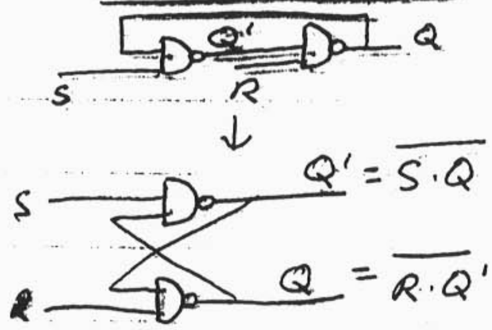


S	R	Q	Q'	
0	0	0	1	idle
0	0	1	0	
0	1	0	1	reset
1	0	1	0	set
1	1	0	0	disallowed

char table

S	R	Q_{n+1}	
0	0	Q_n	idle
0	1	0	reset
1	0	1	set
1	1	(0,0)	dis

NAND Latch

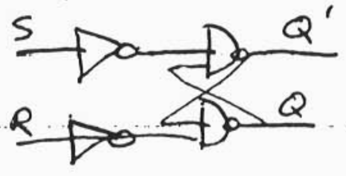


S	R	Q	Q'	
0	0	1	1	disallowed
0	1	0	1	reset
1	0	1	0	set
1	1	0	1	idle
1	1	1	0	

char table

S	R	Q_{n+1}	
1	1	Q_n	idle
0	1	0	reset
1	0	1	set
0	0	(1,1)	dis.

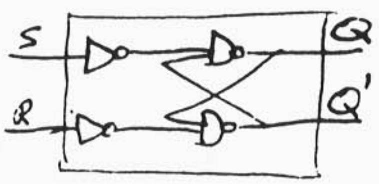
to get same char table put inver in front of NAND input



to get char table

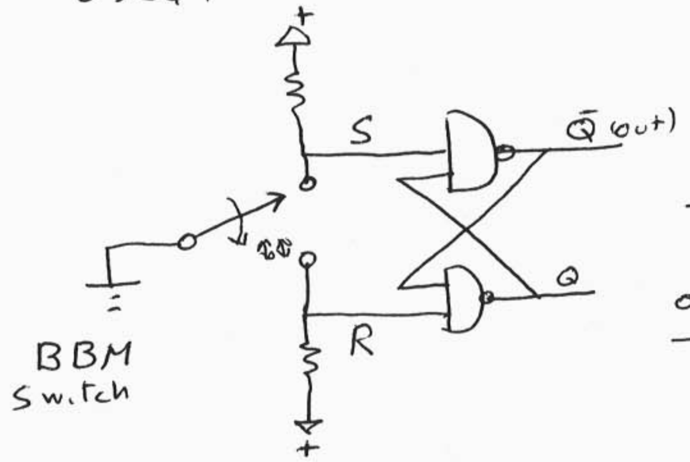
S	R	Q_{n+1}	
0	0	Q_n	(former S=R=1)
0	1	1	(former S=1, R=0) ←
1	0	0	(former S=0, R=1) ←
1	1	1,1 dis	(forme S=0, R=0)

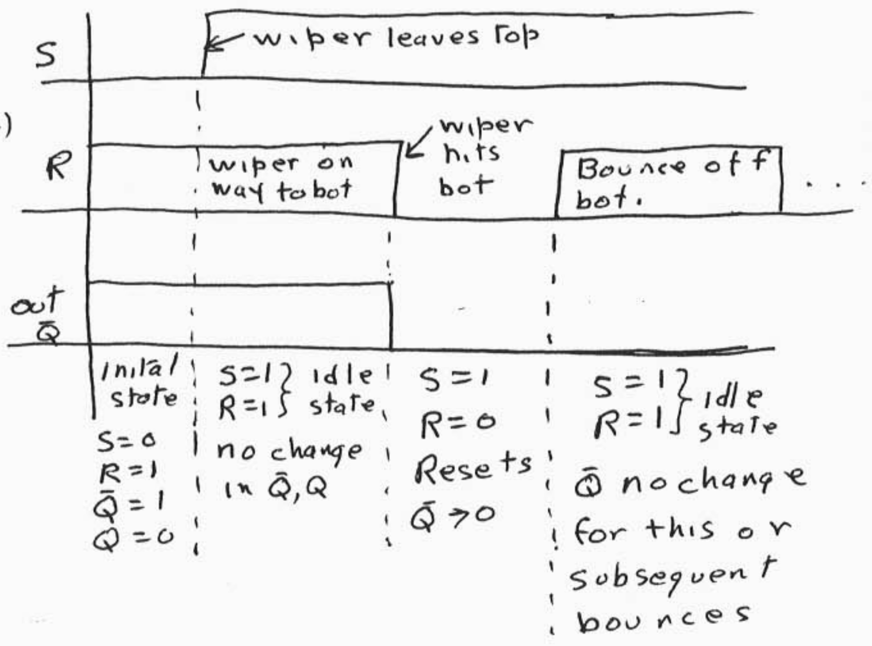
nw 0,0 is idle and 1,1 is s. but 01, 10 states are complements of NOR, so just relabel (ie switch outputs)



now NOR char table is generic

Abblication of a simple latch = switch debouncer. Ordinary switchs are commonly used to input into logic ckts. Many suffer from "bounce" i.e. actually physically bounce on the contact point. Can cause extraneous inputs. Debouncer used:

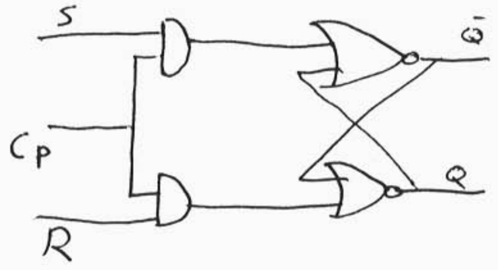




Controlled latches = S-R (or R-S) Flip-Flops

Latches so far have no controling inputs other than their S-R inputs and are there fore always ready to respond to changes in S,R : they are termed "transparent". Most often we want them to be operative (ennabled) only at specific times. Done by adding another input to gate the primary inputs: e.g.

Gated NOR latch:



If line Cp is high, then S, R have their usual effect on Q.

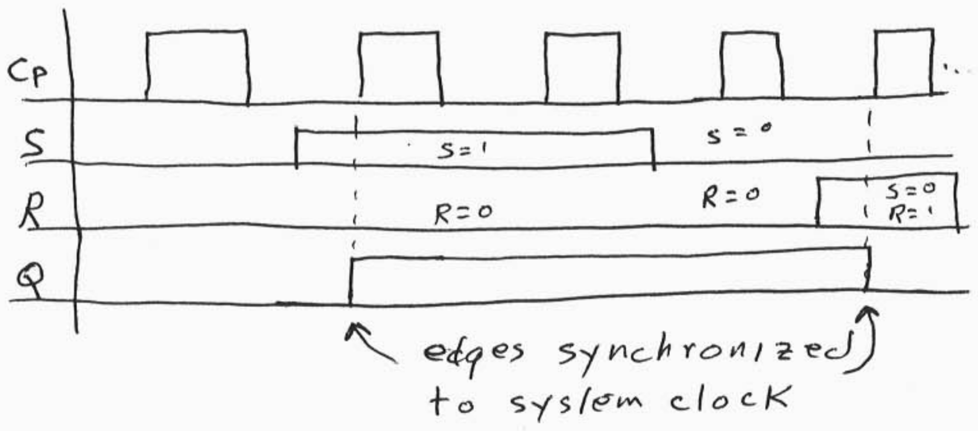
If Cp is low, then inputs to NOR's are brought low = idle state and no change in $\bar{Q}$, Q permitted regardless of what S, R are.

Types of C_p:

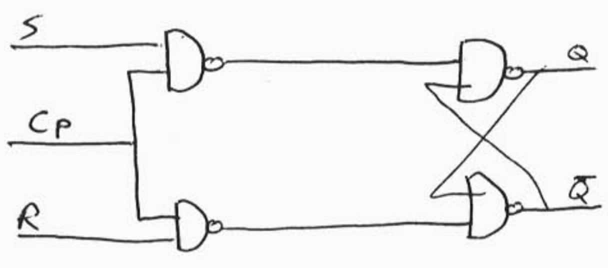
1. Most common: C_p is the system clock, latch (F-F) can respond any time clock is high.
2. Strobe: C_p is a pulse whose ON time is short w/r to system clock, Ennables F-F for brief time.
3. Gate: C_p is a pulse whose ON time is long w/r to system clock. Not very common.

Synchronizing feature of S,R FF (i.e. clocked latch):

Assume S, R coming in from outside. FF can only respond when C_p is high. (Note still can't have S,R high at same time)



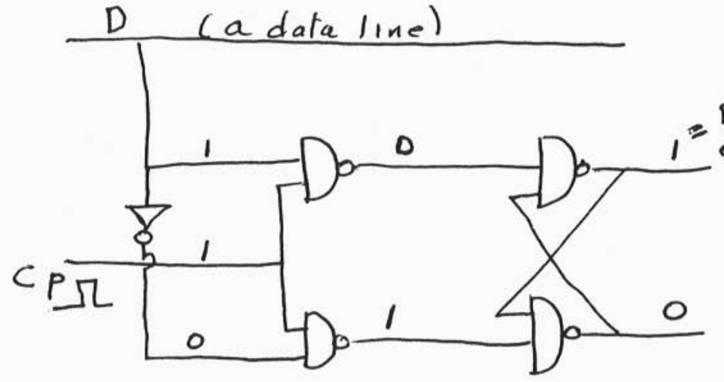
Gated NAND latch: same deal



C_p high $\Rightarrow$ normal operation

C_p low $\Rightarrow$ latch proper forced to idle state, no change permitted.

ata type latch (F-F): a common use of a gated latch is to acquire and retain state of a logic line at a given instant. Cp used may be clock or a strobe depending on application. Inverter used so that S and R are always complements.



Q acquires value of D then retains it when Cp goes low.

Need as many of these as there are D lines:

Naturally never build - instead buy. e.g.

74 75 = 4 bit latch (~50¢)

74100 = 8 " (i.e. "octal") latch (~\$1.25)

and an even more versatile one

74LS 373 (~\$1.35) = an "octal D type transparent latch with tri-state outputs"

A tri-state output has a special ennabling line which either causes the output to behave normally or, in effect, disconnects it from whatever it was driving. Very useful and quite common feature.