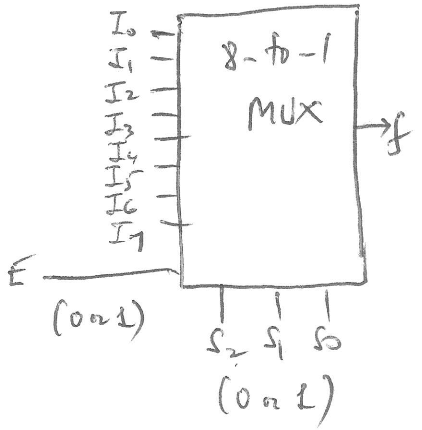
Multiplexers can be used to implement logic functions :

Multiplexer: (8-to 1)

$$f = E\left(I_0 \bar{S}_2\bar{S}_1\bar{S}_0 + I_1 \bar{S}_2\bar{S}_1 S_0 + I_2 \bar{S}_2 S_1 \bar{S}_0 + I_3 \bar{S}_2 S_1 S_0 \right.$$
$$\left. I_4 S_2\bar{S}_1\bar{S}_0 + I_5 S_2\bar{S}_1 S_0 + I_6 S_2 S_1 \bar{S}_0 + I_7 S_2 S_1 S_0\right)$$



Logic function: $f = \sum m(0,2,3,5) = f(x,y,z)$ 3-variable function.

How to implement f using a 8-to-1 MUX ?

$x \to S_2$
$y \to S_1$
$z \to S_0$

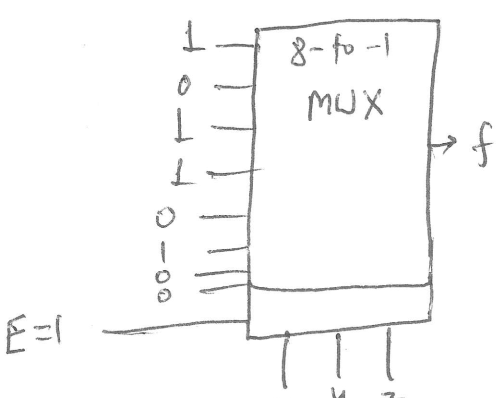
&

$m_0 \to I_0$
$m_1 \to I_1$
$\vdots$
$m_5 \to I_5$
$m_6 \to I_6$
$m_7 \to I_7$

Minterms:

Also : $m_0 = 1$, $m_2 = 1$, $m_3 = 1$, $m_5 = 1$

rest are 0's.



Sometimes it is also possible to implement a 3-variable function using only a 4-to-1 MUX ; requires some math simplification:

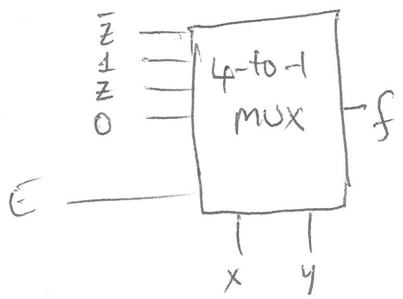
$$f(x,y,z) = \underset{(m_0)}{\bar{x}\bar{y}\bar{z}} + \underbrace{\underset{(m_2)}{\bar{x}y\bar{z}} + \underset{(m_3)}{\bar{x}yz}} + \underset{(m_5)}{x\bar{y}z}$$

$$= \bar{x}\bar{y}\bar{z} + \bar{x}y\underbrace{(\bar{z}+z)}_{1} + x\bar{y}z + xy.0$$

	x	y	z	f
m_0	0	0	0	1
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	1
m_4	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	0
m_7	1	1	1	0

$$f(x,y,z) = \sum m(0,2,3,5)$$

This same function can be written as:



Programmable Logic Devices (PLD's)

PLD's are combinations of AND-arrays & OR-arrays

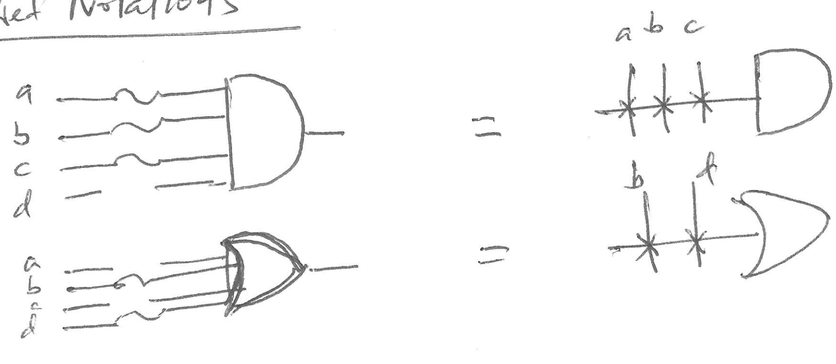
- PROM (Programmable Read Only Memory) — AND's are fixed & OR's programmable
- PLA (Programmable Logic Array) — AND's & OR's are programmable
- PAL (Programmable Array Logic) — AND's are programmable & OR's fixed

Programmable:
- field programmable (done by customer)
- mask programmable (done by manufacturer)

One way to program your connections is to blow fuses of unwanted connections:

fuses on a & b are blown → open connections for AND gates are 1's;
open connections OR gates are 0's!

Simplified Notations

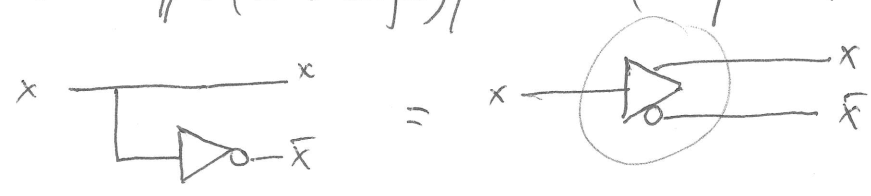


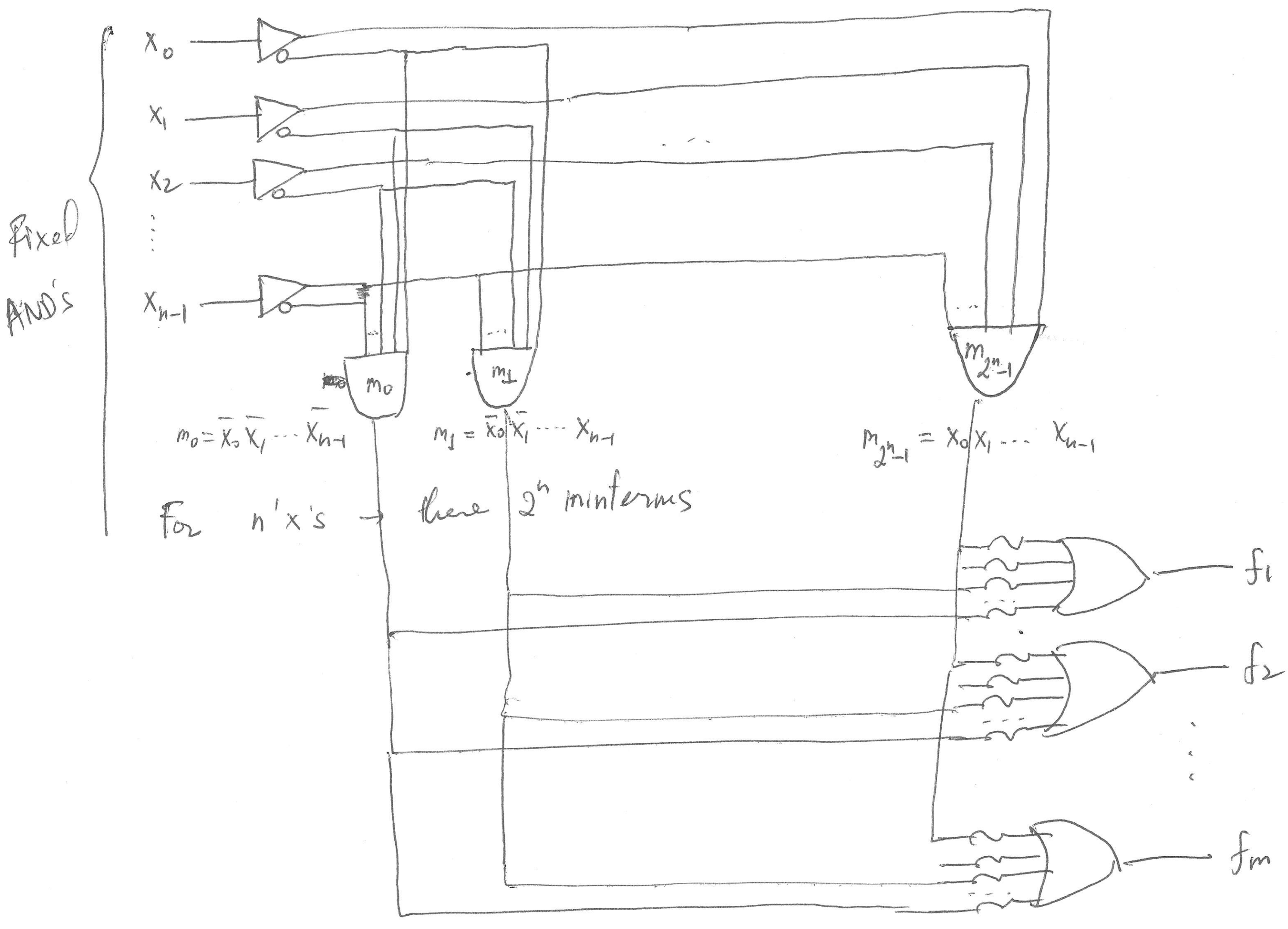
PROM: (AND's fixed & OR's programmable)

↓

decoders (AND gate versions)
(are minterm generators)

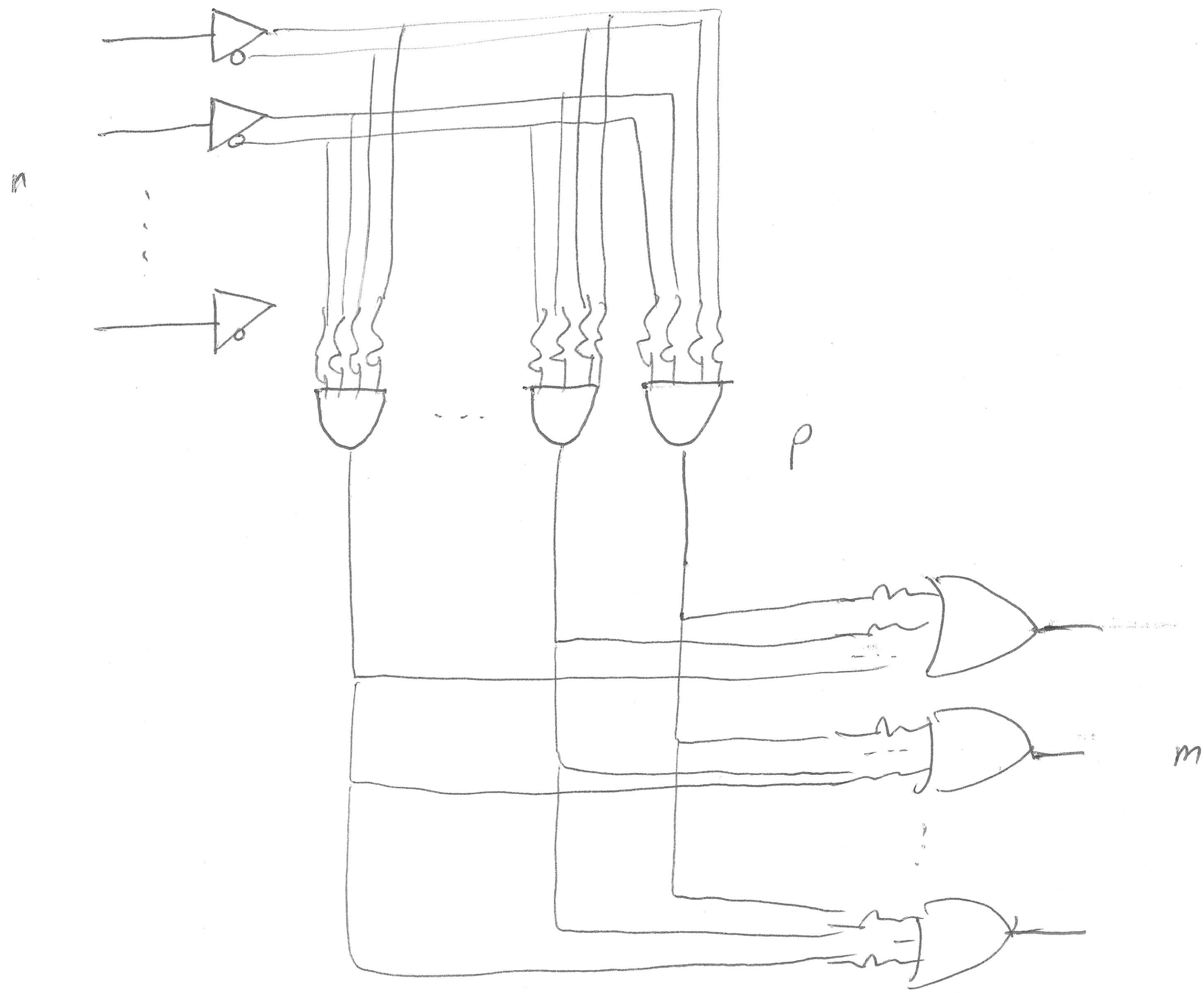
Inputs: use buffer (no change)/inverter (complemented)





2^n (n inputs) $\times$ m PROM.

PLA (AND's & OR's are programmable)



$n \times p \times m$ PLA.

PAL = (AND's programmable & OR's fixed.)

PLA for combinational logic design:

$$\left\{\begin{array}{l} f_1(x,y,z) = \sum m(0,1,3,4) \\ f_2(x,y,z) = \sum m(1,2,3,4,5) \\ \text{Use a } 3\times4\times2 \text{ PLA} \end{array}\right.$$

$3 \to$ inputs, $4 \to$ AND gates, $2 \to$ OR gates

Discussion: 2 OR gates sufficient for the 2 outputs f_1 & f_2; 3 inputs sufficient for x, y, z; 4 AND gates: not sufficient for the 6 min terms b/w f_1 & f_2 → we need simplification using Karnaugh maps or Quine - McKluskey table

m	x y z	f_1	f_2
0	0 0 0	1	0
1	0 0 1	1	1
2	0 1 0	0	1
3	0 1 1	1	1
4	1 0 0	1	1
5	1 0 1	0	1

Karnaugh Maps:

f_1

x \ yz	00	01	11	10
0	1	1	1	0
1	1	0	0	0

A B C

Groups of ones: 4, 2, 1.

3 groups of 2

→ Prime implicants: $\bar{y}\bar{z}$, $\bar{x}\bar{y}$, $\bar{x}z$
A B C

↳ Essential: $\bar{y}\bar{z}$ & $\bar{x}z$
A C

$$f_1(x,y,z) = \bar{y}\bar{z} + \bar{x}z$$

f_2

x \ yz	00	01	11	10
0	0	1	1	1
1	1	1	0	0

A B C D

Groups of ones: 4 groups of two ones:

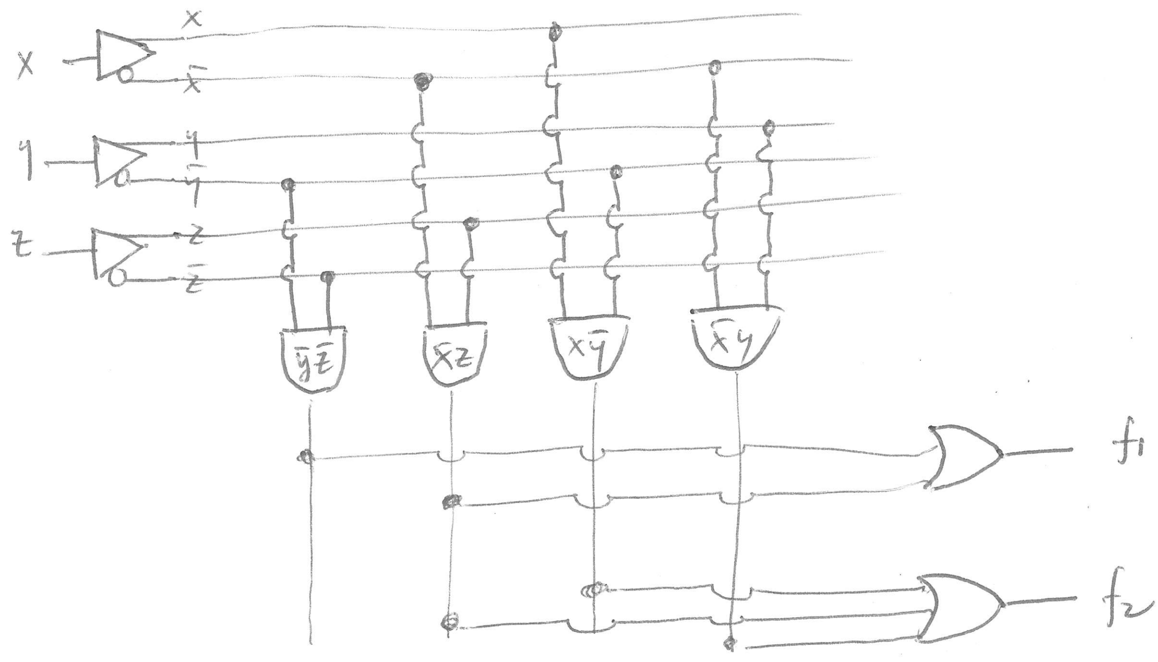
→ Prime implicants: $x\bar{y}$, $\bar{y}z$, $\bar{x}z$, $\bar{x}y$
A B C D

↳ Essential: $x\bar{y}$, $\bar{x}z$, $\bar{x}y$
A C D

$$f_2(x,y,z) = x\bar{y} + \bar{x}z + \bar{x}y$$

4 distinctive products: can now implement using the $3\times4\times2$ PLA

3x4x2 PLA



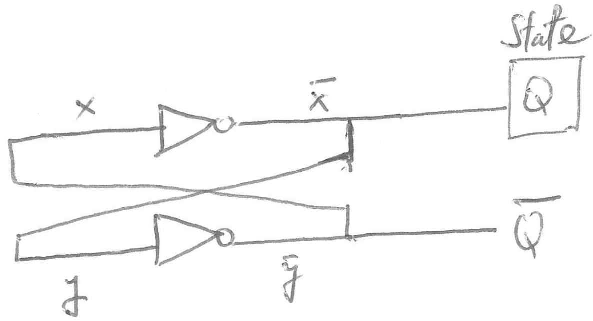
Ch6 Flip-Flops & Applications

Non-sequential Network (so far): output of a function depends on a set of inputs x, y, z, etc. @ same time instant.

Sequential Network:

- output of a function also depends on previous time instants (inputs). Network has memory.
- Two types: . synchronous (internal clock set update @ discrete time instants)
  . asynchronous
- Memory will be provided by the flip-flop.

– Bistable element: (central to all flip-flop circuit)

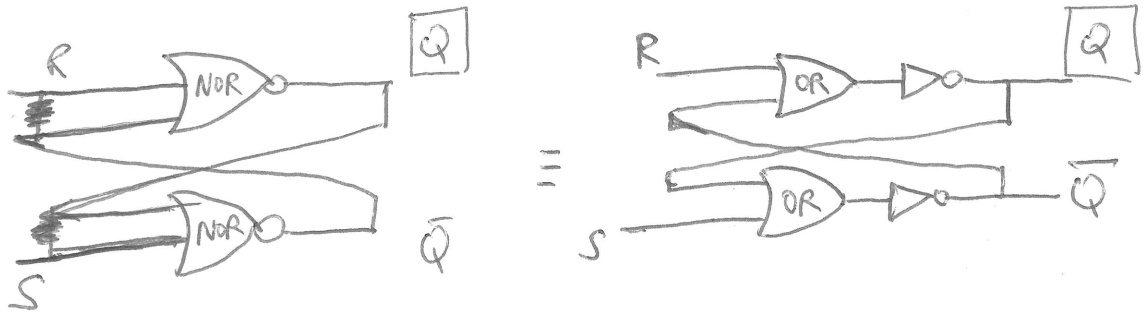


$x = \bar{y} = \bar{Q}$	$y = \bar{x} = Q$	
1	0	Bistable states
0	1	
0.5	0.5	Metastable state (should be avoided)

Observation:

If $x = 0 \rightarrow Q = 1$ and will stay in this state until some voltage is introduced to "clear the setting" or to "clear memory" or "resetting"

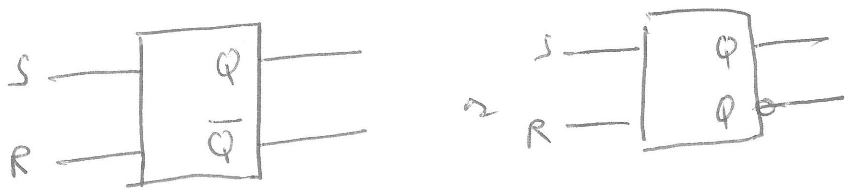
– SR Latch (Set / Reset Latch):



R	S	Q	$\bar{Q}$
0	0	0/1	1/0
0	1	1	$\leftarrow$ 0
1	0	0 $\rightarrow$	1
1	1	0	0 (unpredictable)

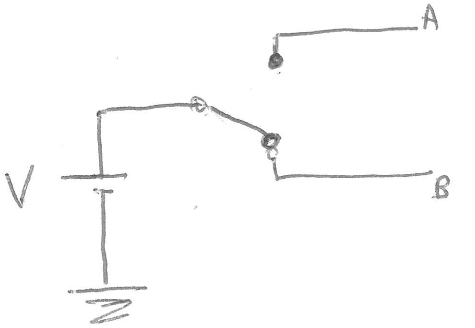
→ Bistable Element!

(1 or 0 / 1 or 1 } are both 1 → after ▷o, $\bar{Q} = 0$, so $Q = 1$)



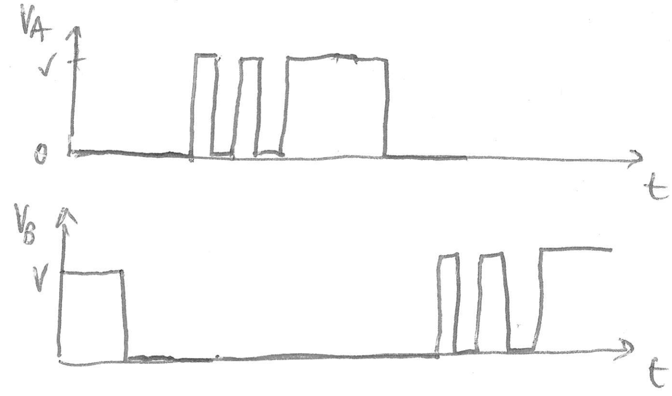
Application of SR Latch : switch debouncer

Contact bounce:



$V_A = 0$

$V_B = V$

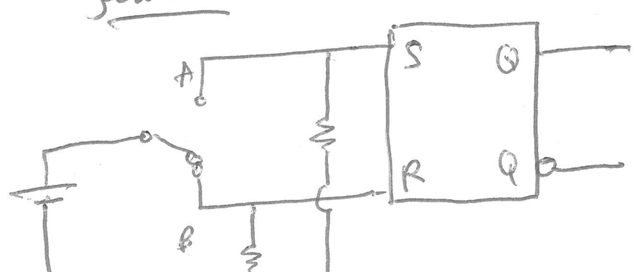


Hard ball collision effects.

When we switch from B to A: V_B drops to 0 while V_A is still @ 0. Then V_A will raise to V, but since the contacts bounce a few times before it stays, V_A oscillates (switches) between V & 0 a few times until it stays @ V. Same when we switch back to B.

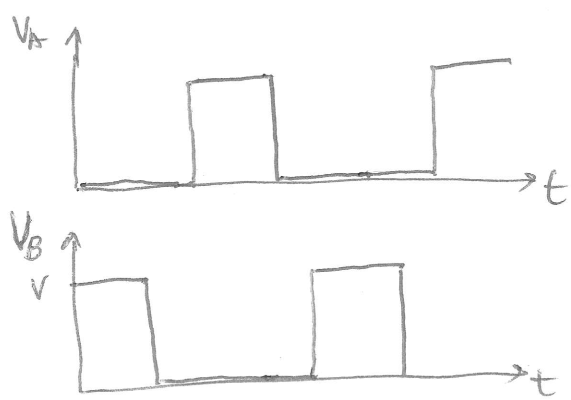
This is a problem: e.g. key on a keyboard: when we press once, it may interpret as 2 or more pressings.

Solution: connect an SR latch after the switch:

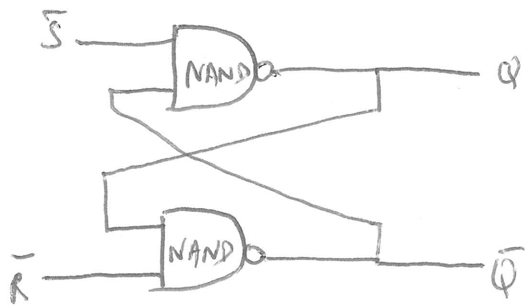


- When not connected A or B will stay @ 0. In the figure $V_A = 0$ & $V_B = V$ ($S = 0$ & $R = 1$)
- Switch from B to A → ($S = 1$ & $R = 0$) → ($Q = 1$; $\bar{Q} = 0$)
- If switch bounces away from A: ($S = 0$ & $R = 0$) → Bistable output → output state is not changed: ($Q = 1$; $\bar{Q} = 0$).

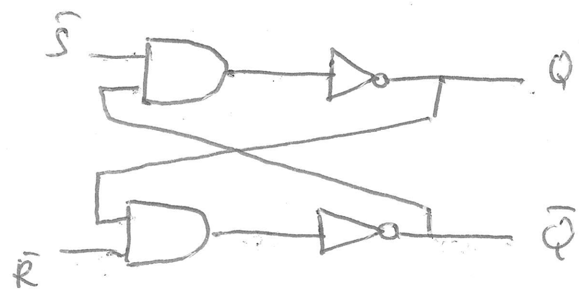
→ Bouncing effect is eliminated



$\bar S \bar R$ Latch: using two NAND's



$\equiv$



R	S	$\bar R$	$\bar S$	Q	$\bar Q$	
0	0	1	1	0|1	1|0	→ Bistable element.
0	1	1	0	1	0	
1	0	0	1	0	1	
1	1	0	0	1	1	(unpredictable).

$\boxed{\bar S = 1}$ if $\underbrace{\bar Q = 0} \rightarrow Q = 1$ & $\boxed{\bar R = 0} \rightarrow \underbrace{\bar Q = 1}$ $\Rightarrow$ if $\boxed{\bar S = 1} \Rightarrow \bar Q = 1.$

Not stable!

$\rightarrow Q = 0$ & $\boxed{\bar R = 0}$

$\rightarrow \bar Q = 1$ ✓

$\boxed{\bar S = 0}$ if $\underbrace{\bar Q = 1} \rightarrow Q = 1$ & $\boxed{\bar R = 1} \rightarrow \underbrace{\bar Q = 0}$ $\Rightarrow$ $\bar S = 0 \Rightarrow \bar Q = 0$

Not stable!

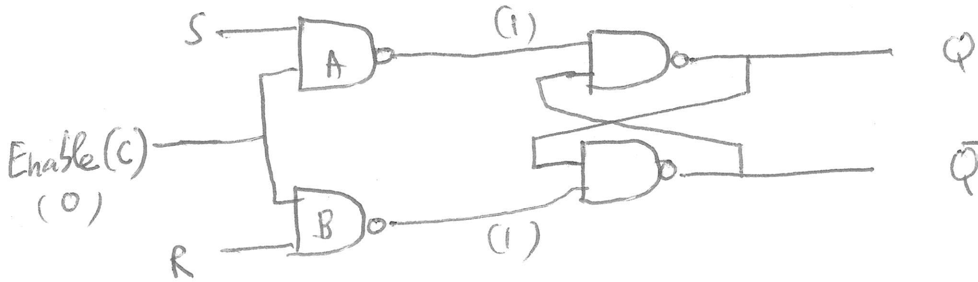
$\Rightarrow Q = 1$ & $\boxed{\bar R = 1}$

$\Rightarrow \bar Q = 0$ ✓

x	y	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

<u>Gated SR Latch</u>: to add an enable / disable ability:

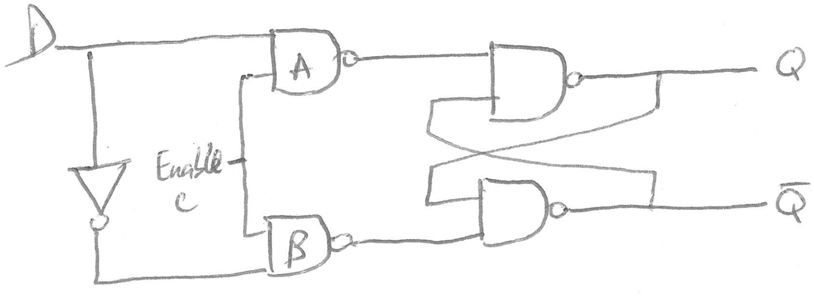
↳ 2 additional NAND's to a $\bar{S}\bar{R}$ Latch:



As long as the Enable input is 0, the outputs of A & B are 1's
→ from the $\bar{S}\bar{R}$ Truth Table we get a bistable element or
<u>Q & $\bar{Q}$ are not changed.</u>

(Inputs)			(Outputs)		
S	R	C	Q	$\bar{Q}$	
–	–	0	Q	$\bar{Q}$	(Bistable element)
0	1	1	0	1	
1	0	1	1	0	
1	1	1	1	1	(unpredictable)
0	0	1	Q	$\bar{Q}$	

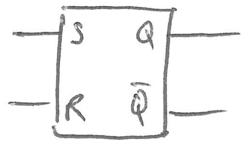
<u>Gated D Latch</u>:
- eliminates the inputs leading to unpredictable outputs in the previous 3 latches.
- evolved from the gated SR latch.

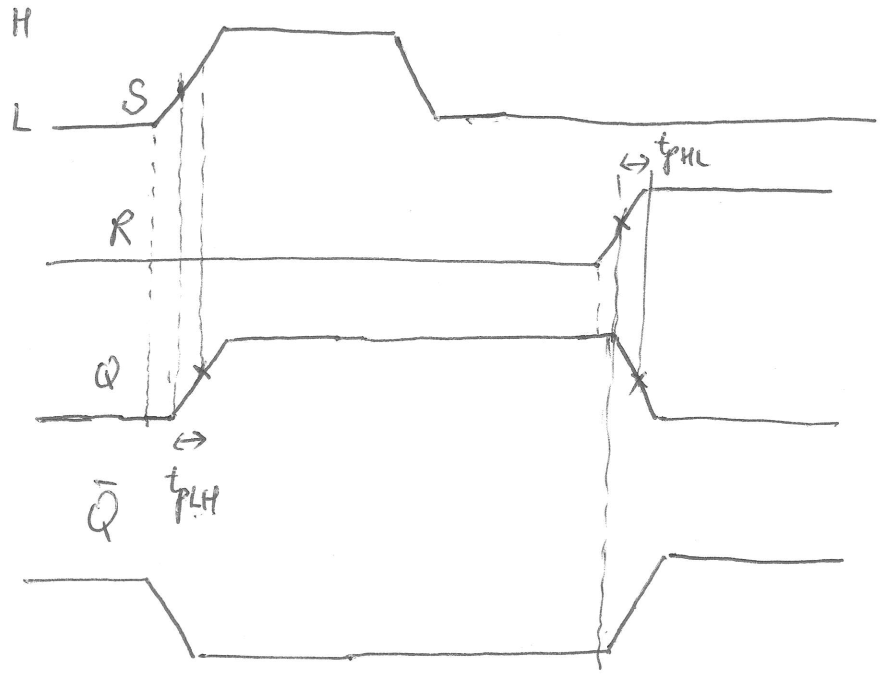


D	C	Q	$\bar{Q}$	
–	0	Q	$\bar{Q}$	(Bistable)
0	1	0	1	
1	1	1	0	

Propagation $\boxed{\text{delay}}$ in flip-flops:

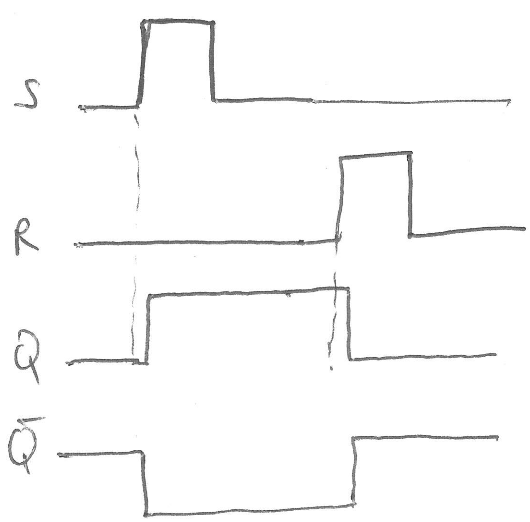
SR latch:





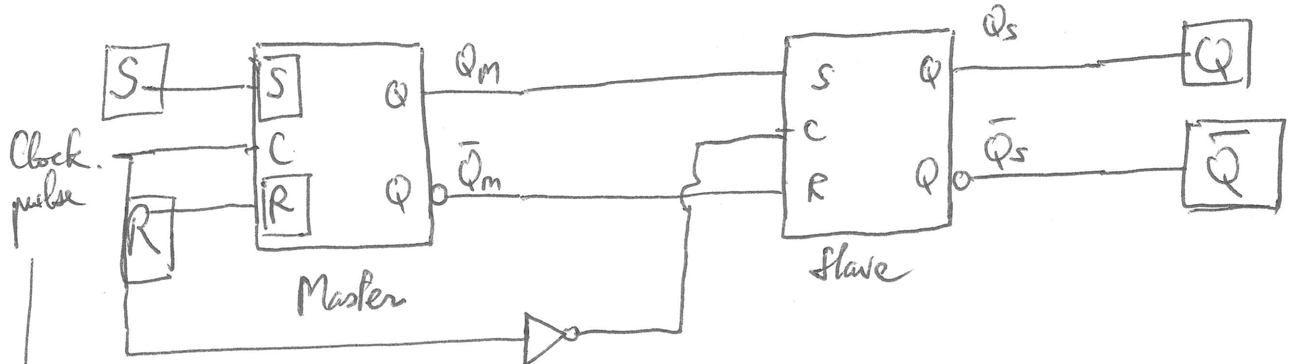
Delay times: t_{PLH} (Low-to-high); t_{PHL}.

Most of the time: omit finite slopes, all delay times are equal.

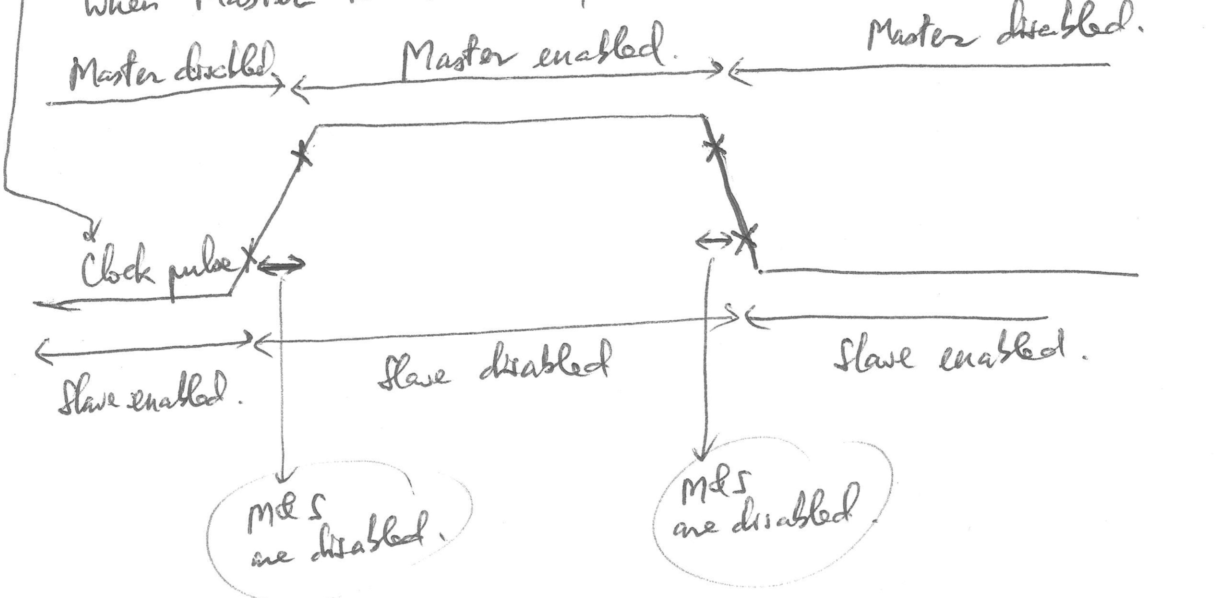


Master-Slave Flip Flops (pulse-triggered flip-flop)

SR Flip-flops: when we reset, information affects the output right away (except for delays) → these flip-flops are called "transparent". In a Master-Slave flip-flop: a pulse at the Master's Clock controls when the information is propagated the outputs.



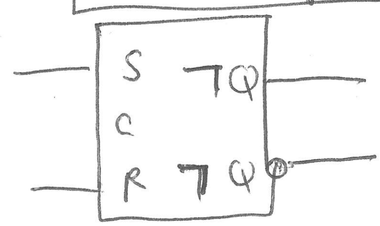
C = Enable input;
When Master is enabled, Slave is disabled & vice versa



M&S SR Flip-Flop:

S	R	C	Q	$\overline{Q}$
–	–	0	Q	$\overline{Q}$
0	0	1	Q	$\overline{Q}$
0	1	1	0	1
1	0	1	1	0
1	1	1	–	–

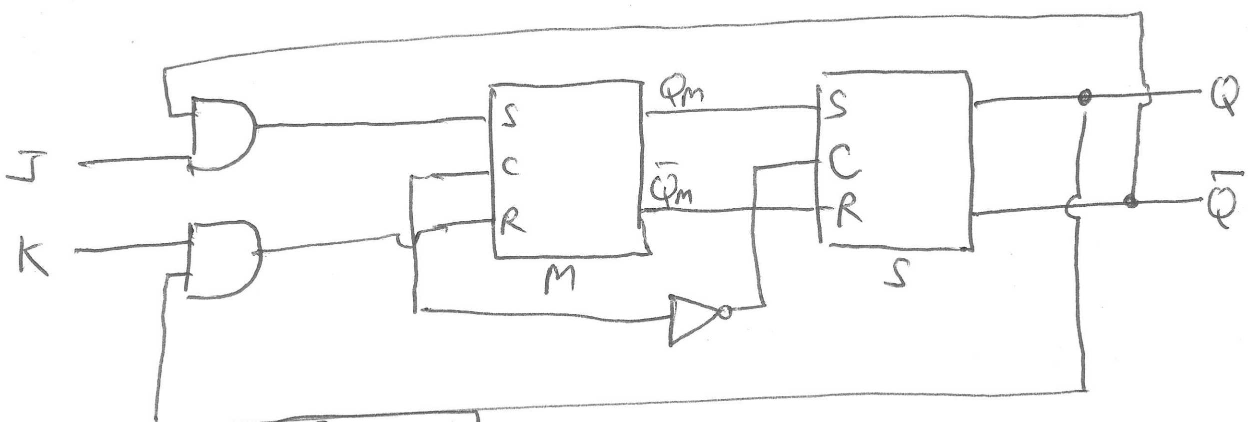
M&S SR Flip-Flop



Symbol for a M&S flip-flop.

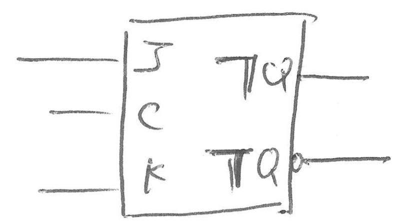
┐ Right edge of clock pulse is when the input information get propagated to the output.

Other types: M&S JK Flip-flop:



M&S JK

S	R	C	Q	$\overline{Q}$
–	–	0	Q	$\overline{Q}$
0	0	1	Q	$\overline{Q}$
0	1	1	0	1
1	0	1	1	0
1	1	1	$\overline{Q}$	Q



<u>M&S D Flip-Flop:</u> additional inverter b/w S & R on the M&S SR FF

<u>M&S T Flip-Flop:</u> J & K in the M&S JK FF are connected → T: