Laboratorio di Elettronica e Tecniche di Acquisizione Dati
2025-2026

Elettronica digitale

(cfr. http://physics.ucsd.edu/~tmurphy/phys121/phys121.html

https://www.allelcoelec.com/blog/XOR-Gate-Explained-Symbol,Truth-Table,Construction-Methods,and-Applications.html)

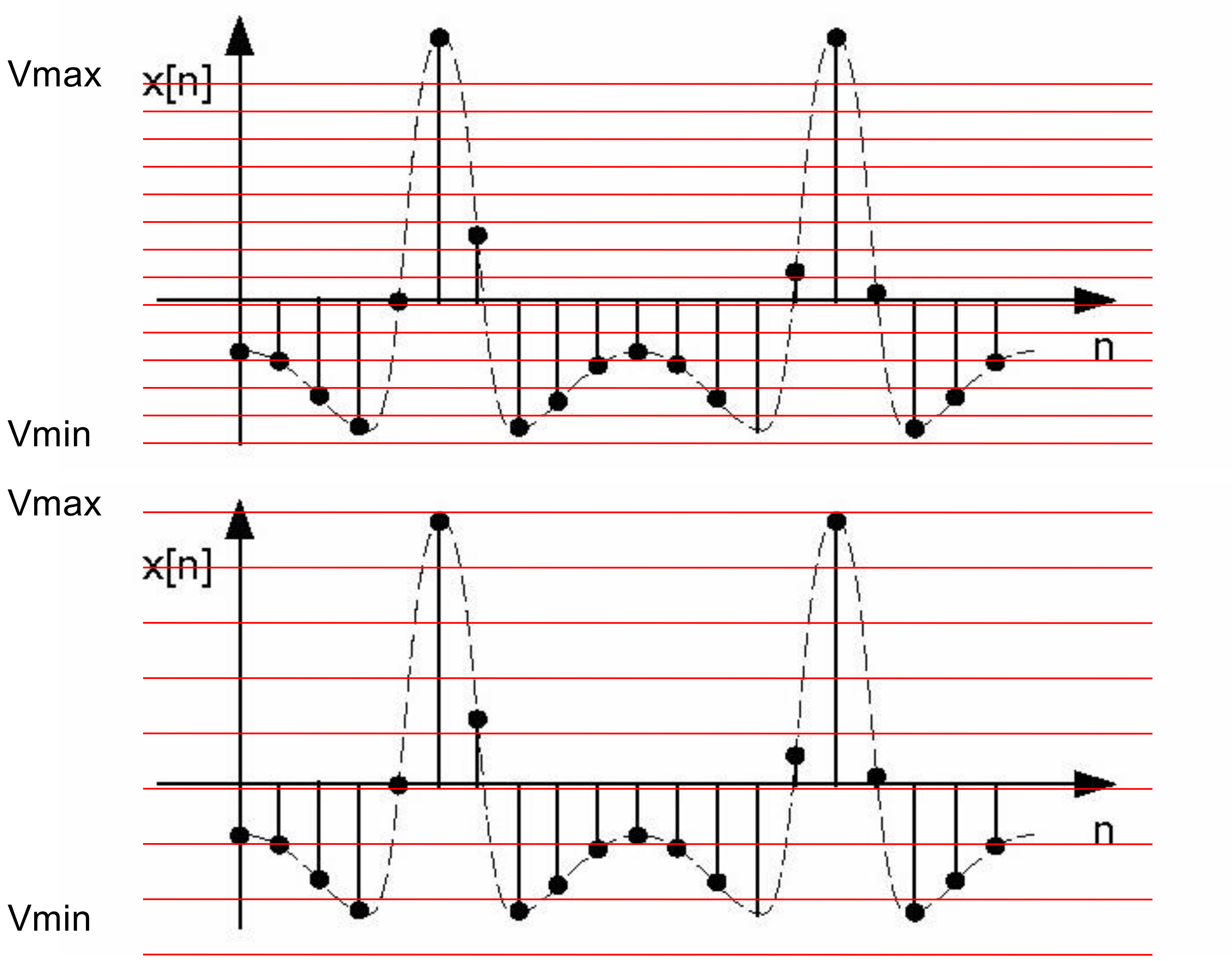
ADC (1)

- Dal punto di vista funzionale gli ADC sono dei *classificatori:*
  - l' intervallo di variabilità del segnale V_x viene diviso in n intervalli, detti *canali*, di ampiezza costante K. Definiamo quindi $V_i = K\, i + V_o$
  - il segnale in ingresso V_x viene *classificato* nel canale i-esimo se è verificata la relazione

$$V_{i-1} < V_x < V_i$$

  - inevitabilmente si ha un errore di quantizzazione

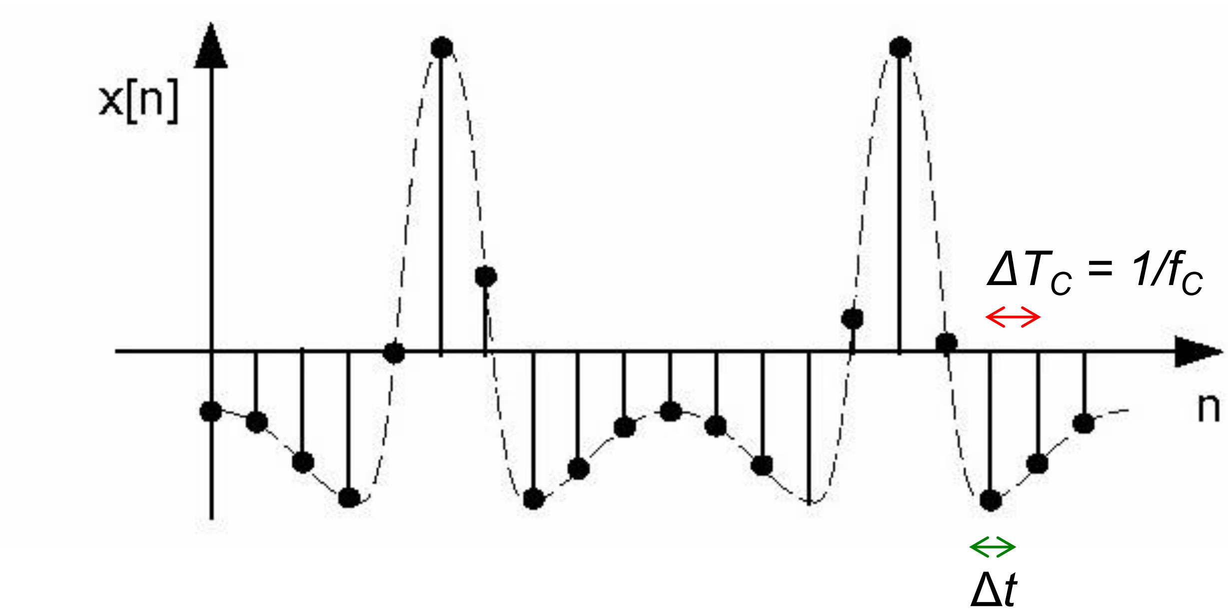
ADC (2)



ADC: Analog to Digital Converter

ADC “commerciali” sono caratterizzati da:

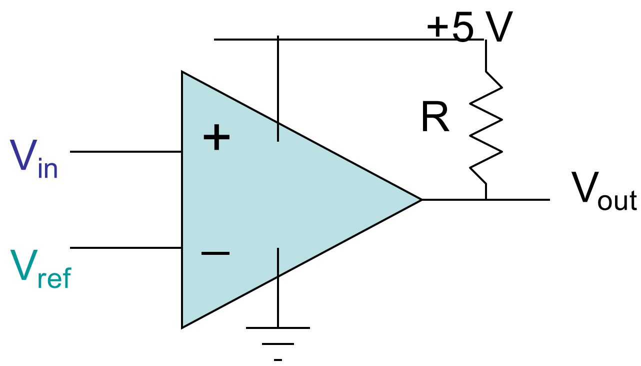
- **Range**: Intervallo di tensione che l’ADC può accettare in ingresso: *[Vmin, Vmax]*
- **Numero di canali in cui è diviso il range**: definito dal numero n di bit
  - n = 12 bit: $N = 2^{12} = 4096$
  - n = 16 bit: $N = 2^{16} = 65536$
- **Risoluzione**: minima variazione di tensione rivelabile: *(Vmax-Vmin) / n*
- **Sampling rate**: frequenza di campionamento $f_C = 1/\Delta T_C$
- **Sampling time**: intervallo di tempo necessario ad effettuare una operazione di campionamento Δt

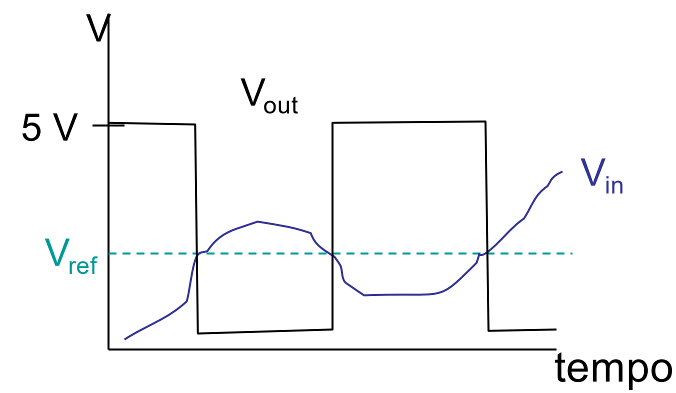


Comparatori

- è spesso utile generare un segnale elettrico "forte" associato con un certo evento (cfr. *trigger*)
- possiamo utilizzare un comparatore per confrontare un segnale con una certa soglia
  - può essere una temperatura, una pressione, etc…: qualsiasi cosa che possa essere trasformata in un voltaggio
- possiamo utilizzare un operazionale senza feedback
  - input invertente alla soglia
  - input non-invertente collegato al segnale da testare
  - l'operazionale farà uscire un segnale (a fondo scala) negativo se il segnale è < della soglia, positivo se il segnale è > della soglia
- purtroppo l'operazionale è lento (basso "slew rate")
  - 15 V/μs significa 2 μs per arrivare a fondo scala se alimentato $\pm$15 V

Esempio (reale) di comparatore





- quando $V_{in} < V_{ref}$, V_{out} va a V_{CC}^{-}, i.e. 0 (*)
- quando $V_{in} > V_{ref}$, V_{out} va a V_{CC}^{+}, i.e. 5V (*)
  - nell'esempio è anche “pulled-up” (attraverso il resistore di “pull-up”, usualmente 1 kΩ o più): se la corrente richiesta dal carico è maggiore di quella possibile per l'op.amp (*O*(10 mA)), l'extra-corrente arriva dall'alimentazione esterna e non dall'op.amp
- l’uscita è una versione “digitale” del segnale
  - i valori “alto” e “basso” sono configurabili (ground e 5V, nell’esempio)
- possono essere utili anche per convertire un segnale “lento” in uno “veloce”
  - se è necessaria una maggiore precisione di “timing”

(*) in realtà a meno di N potenziali di contatto / soglie di conduzione del diodo…

“digitale”

continuo → discreto

analogico → digitale

Stati logici solo due possibili stati

1, alto (H), vero (true)
0, basso (L), falso (false)

<u>Algebra booleana</u>

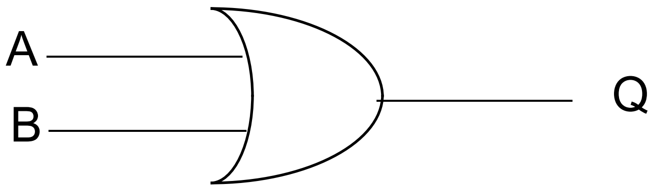
sistema matematico per l’analisi di stati logici

solo 3 funzioni logiche di base → **AND OR NOT** ← Funzioni logiche

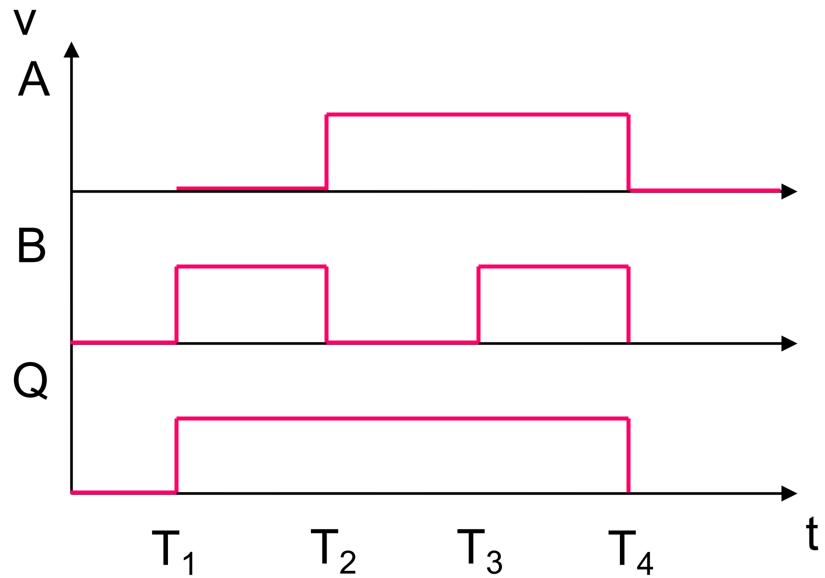
circuiti usati per la realizzazione di funzioni logiche → Porte logiche

Porte logiche di base - OR

OR



Q = A+B



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

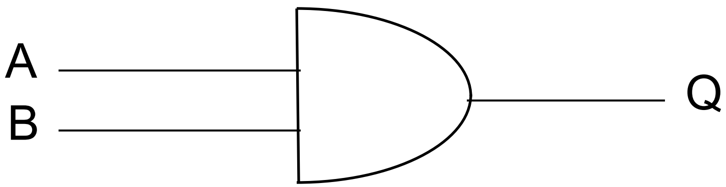
A+B+C = (A+B)+C = A+(B+C)

A+B = B+A

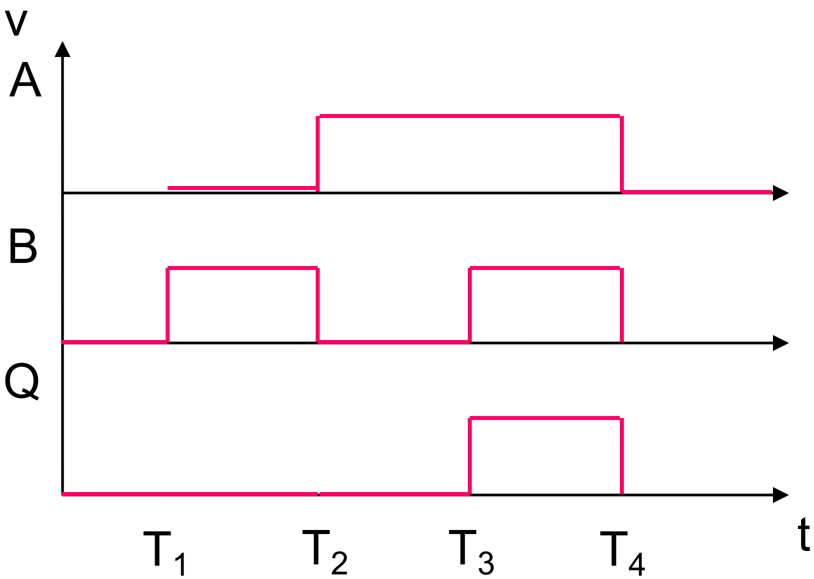
A+1 = 1, A+A = A, A+0 = A

Porte logiche di base - AND

AND



$$Q = A \cdot B$$



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

$$A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

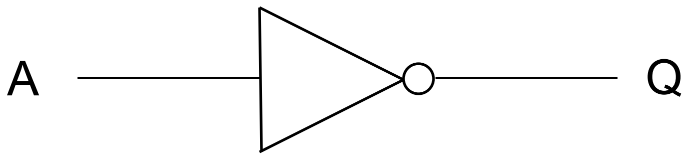
$$A \cdot B = B \cdot A$$

$$A \cdot 1 = A, \; A \cdot A = A, \; A \cdot 0 = 0$$

$$A \cdot (B+C) = A \cdot B + A \cdot C$$

Porte logiche di base - NOT

NOT



A	Q
0	1
1	0

$$\overline{\overline{A}} = A$$

$$\overline{A} + A = 1$$

$$\overline{A} \cdot A = 0$$

$$A + \overline{A} \cdot B = A + B$$

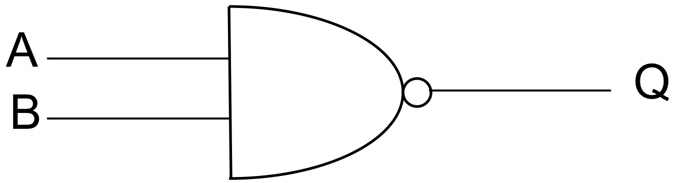
sapendo che

$$B + 1 = 1,\ A \cdot 1 = A,\ \overline{A} + A = 1$$

$$A + \overline{A} \cdot B = A \cdot (B + 1) + \overline{A} \cdot B = A \cdot B + A + \overline{A} \cdot B =$$
$$\left(A + \overline{A}\right) \cdot B + A = B + A = A + B$$

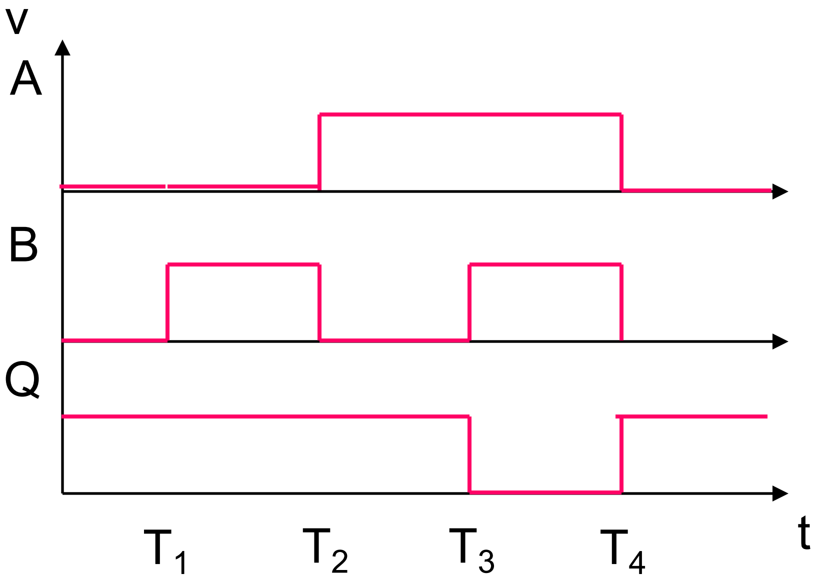
Porte logiche di base – NAND

NAND



$$Q = \overline{A \cdot B}$$

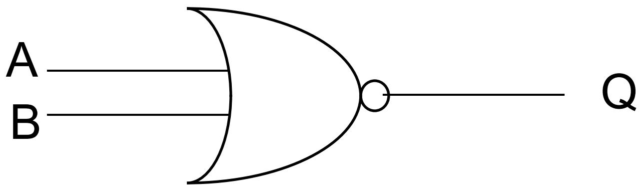
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



porta universale

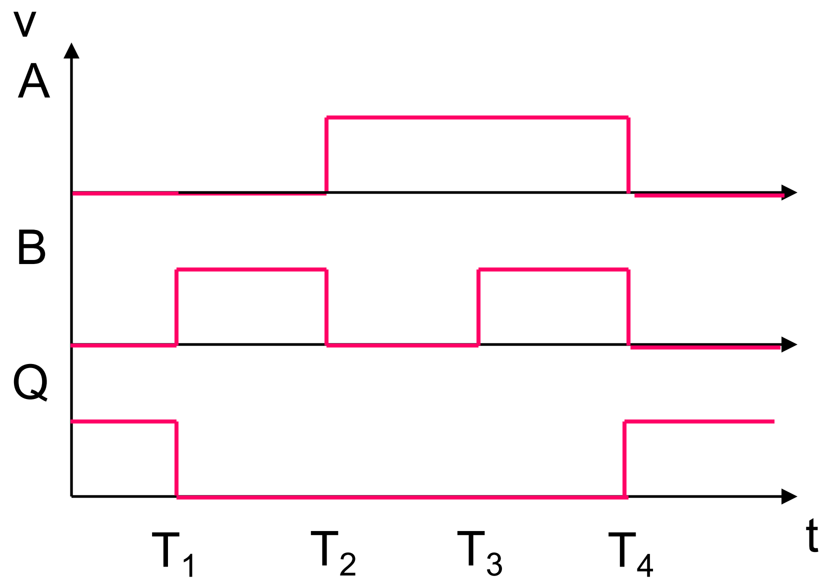
Porte logiche di base – NOR

NOR



$$Q = \overline{A + B}$$

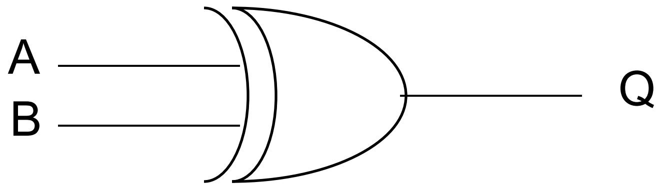
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



Porte logiche di base – XOR

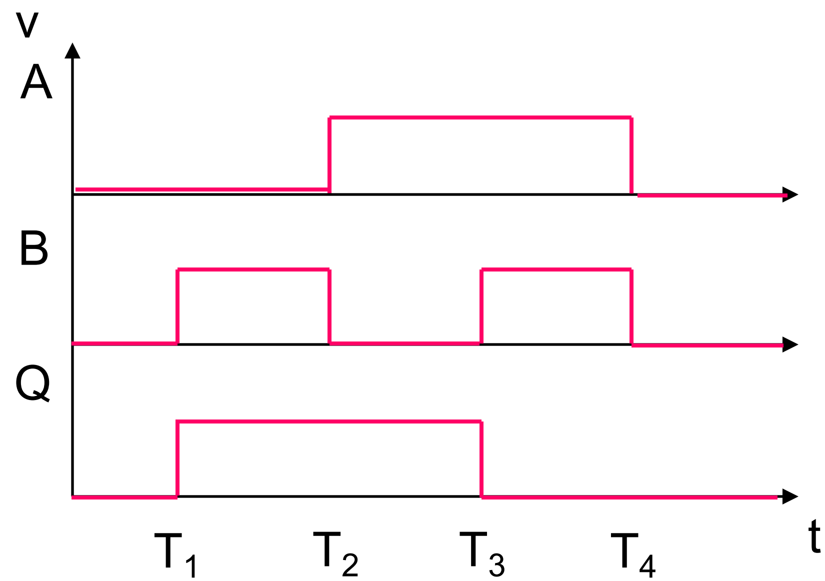
XOR

OR esclusivo



$$Q = A \oplus B$$

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



XOR con AND, NOT e OR (1)

OR

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

XOR

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

3 casi sono già ok, va sistemato solo l'ultimo

XOR con AND, NOT e OR (1)

OR				XOR		
A	B	Q	Q_2	A	B	Q
0	0	0	1	0	0	0
0	1	1	1	0	1	1
1	0	1	1	1	0	1
1	1	1	0	1	1	0

Q_2 è una NAND (AND negata)

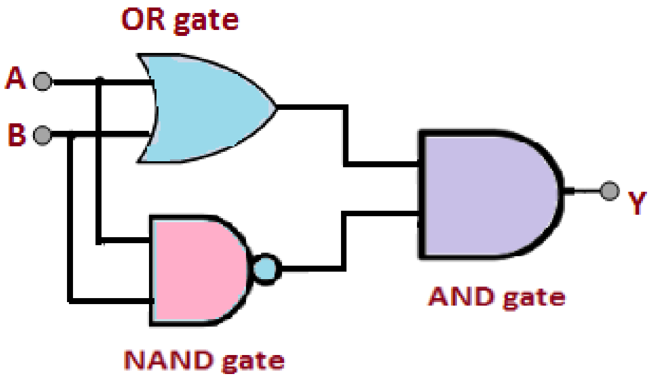
se inserisco un'altra operazione posso agire sull'ultimo combinando questa operazione con l'OR (Q)

XOR con AND, NOT e OR (1)

OR | **XOR**

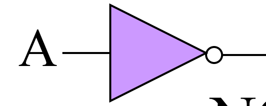
A	B	Q	Q_2	Y	A	B	Q
0	0	0	1	0	0	0	0
0	1	1	1	1	0	1	1
1	0	1	1	1	1	0	1
1	1	1	0	0	1	1	0

se combino la OR (Q), in modo opportuno (AND) con la NAND (Q2) ottento esattamente la mia XOR (Y)



Manipolazione dati

- tutta la “manipolazione” è basata sulla *logica*
- la logica segue regole ben precise, producendo uscite deterministiche, funzione solamente degli input



NOT

A	C
0	1
1	0

AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

XOR

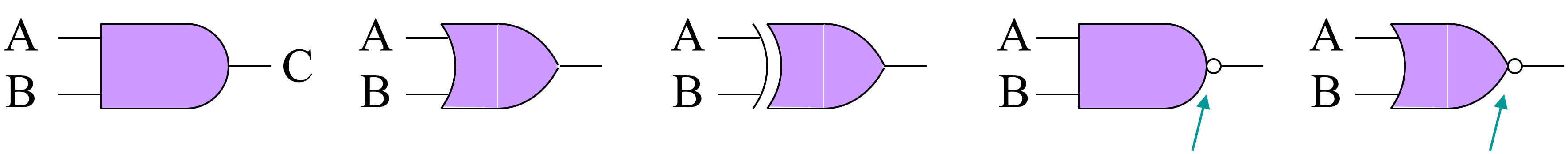
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



la “palletta” significa (e.g., NOT AND → NAND)

Algebra Booleana

A	B	C	$f(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Prima forma canonica (esempio)

$$f(A,B,C) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$$

Ogni riga come prodotto (AND) dei termini naturali (se 1) o complementati (se 0)
Somma (OR) delle righe con valore pari a 1.

Seconda forma canonica (esempio)

$$f(A,B) = (A+B)(\overline{A}+B)(\overline{A}+\overline{B})$$

Ogni riga come somma (OR) dei termini naturali (se 1) o complementati (se 0)
Prodotto (AND) delle righe con valore pari a 0.

A	B	$f(A,B)$
0	0	0
0	1	1
1	0	0
1	1	0

Algebra Booleana

A	B	C	$f(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Prima forma canonica (esempio)

$$f(A,B,C) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$$

Ogni riga come prodotto (AND) dei termini naturali (se 1) o complementati (se 0)

Somma (OR) delle righe con valore pari a 1.

Seconda forma canonica (esempio)

$$f(A,B) = (A+B)(\overline{A}+B)(\overline{A}+\overline{B})$$

A	B	$f(A,B)$
0	0	0
0	1	1
1	0	0
1	1	0

Ogni riga come somma (OR) dei termini naturali (se 1) o complementati (se 0)

Prodotto (AND) delle righe con valore pari a 0.

Algebra Booleana

A	B	C	$f(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Prima forma canonica (esempio)

$$f(A,B,C) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$$

Ogni riga come prodotto (AND) dei termini naturali (se 1) o complementati (se 0).

Somma (OR) delle righe con valore pari a 1.

Seconda forma canonica (esempio)

$$f(A,B) = (A+B)(\overline{A}+B)(\overline{A}+\overline{B})$$

A	B	$f(A,B)$
0	0	0
0	1	1
1	0	0
1	1	0

Ogni riga come somma (OR) dei termini naturali (se 1) o complementati (se 0)

Prodotto (AND) delle righe con valore pari a 0.

Algebra Booleana

A	B	C	$f(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Prima forma canonica (esempio)

$$f(A,B,C) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$$

Ogni riga come prodotto (AND) dei termini naturali (se 1) o complementati (se 0)
Somma (OR) delle righe con valore pari a 1.

Seconda forma canonica (esempio)

$$f(A,B) = (A+B)(\overline{A}+B)(\overline{A}+\overline{B})$$

A	B	$f(A,B)$
0	0	0
0	1	1
1	0	0
1	1	0

Ogni riga come somma (OR) dei termini naturali (se 1) o complementati (se 0)
Prodotto (AND) delle righe con valore pari a 0.

Algebra Booleana

Algebra booleana

trasformare una funzione logica in un'altra di più facile implementazione hardware

Teoremi di De Morgan

$$\overline{A \cdot B \cdot C \cdot ....} = \overline{A} + \overline{B} + \overline{C} + ...$$

$$\overline{A + B + C + ...} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot ....$$

Il complemento dell'AND di più variabili logiche è dato dall'OR dei complementi

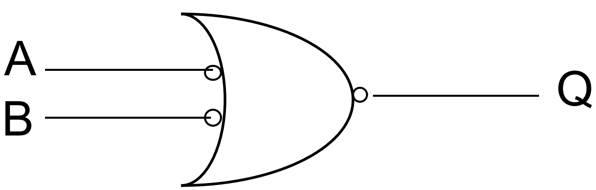
Il complemento dell'OR di più variabili logiche è dato dall'AND dei complementi

Algebra Booleana

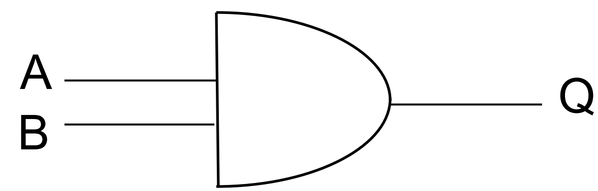
Un circuito AND per logica positiva funziona come un OR per logica negativa

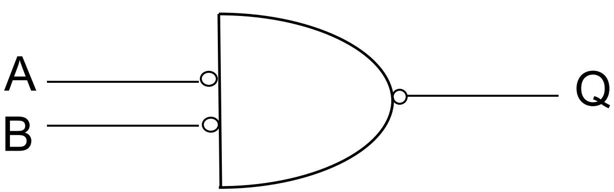
non è necessario usare i tre circuiti di base

bastano due OR e NOT oppure AND e NOT

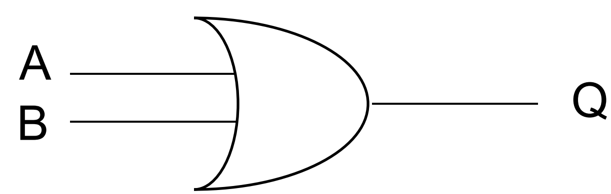


$$\overline{\overline{A}+\overline{B}} \Leftrightarrow A\cdot B$$





$$\overline{\overline{A}\cdot\overline{B}} \Leftrightarrow A+B$$

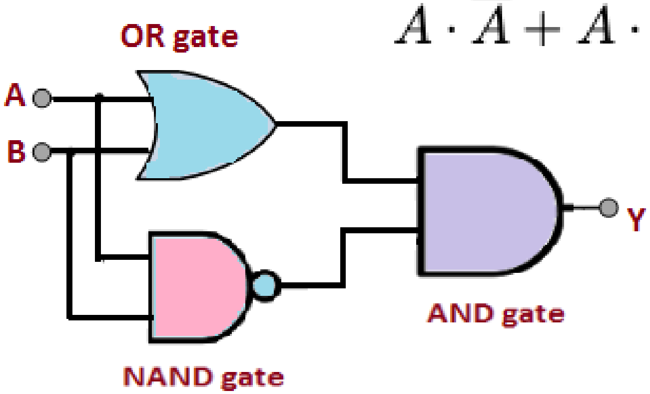


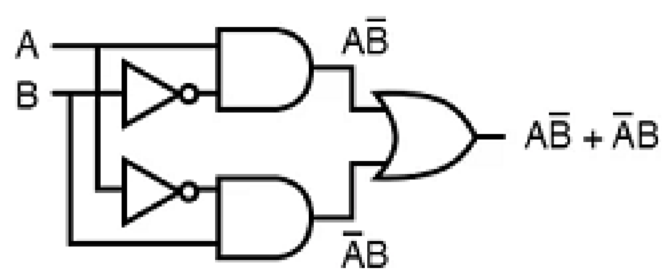
XOR con AND, NOT e OR (2)

OR					XOR		
A	B	Q	Q_2	Y	A	B	Q
0	0	0	1	0	0	0	0
0	1	1	1	1	0	1	1
1	0	1	1	1	1	0	1
1	1	1	0	0	1	1	0

$$(A+B)\cdot\overline{(A\cdot B)} = (A+B)\cdot(\overline{A}+\overline{B})$$

$$A\cdot\overline{A}+A\cdot\overline{B}+B\cdot\overline{A}+B\cdot\overline{B} = A\cdot\overline{B}+B\cdot\overline{A}$$

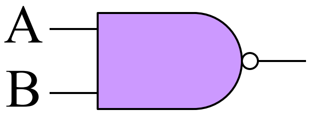




Tutta la logica con la sola NAND

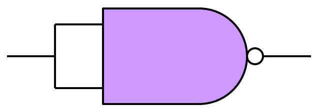
NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



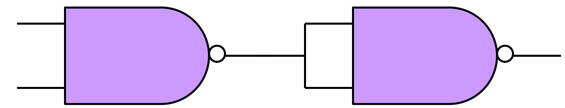
NOT

A	C
0	1
1	0

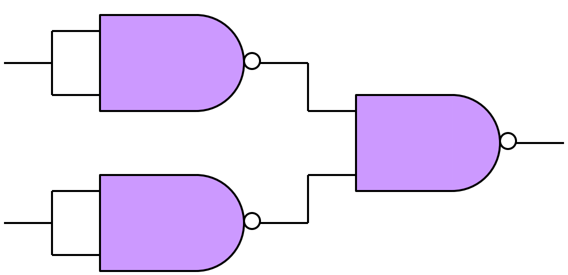


AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



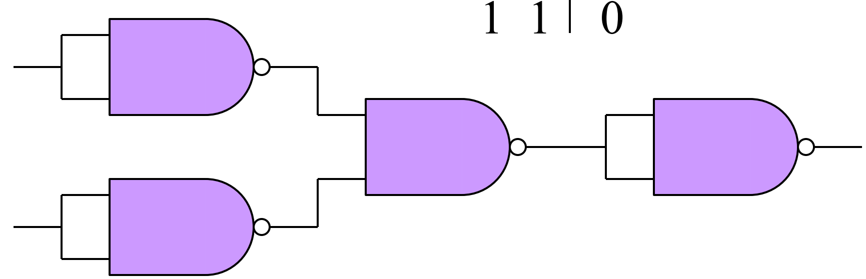
OR



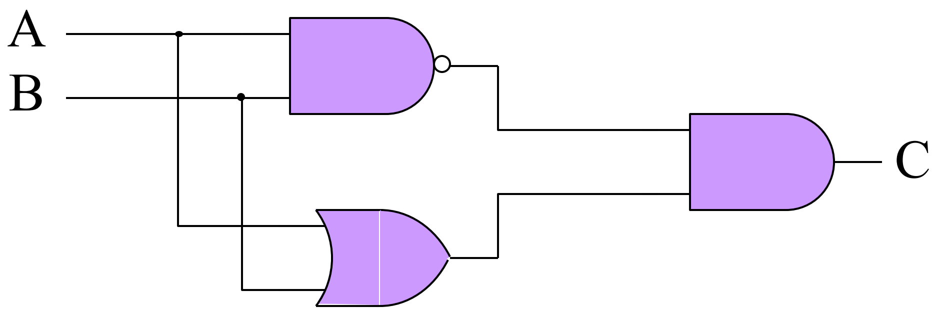
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0



Tutta la logica con la sola NAND



XOR = (A NAND B) AND (A OR B)

- la OR già sappiamo come farla di sole porte NAND
- 6 NAND in totale: 3 per la OR, 2 per la AND e 1 per la NAND
- questa è una XNOR, che utilizzando un’altra NAND viene negata, cioè diventa la XOR desiderata

Aritmetica

- sommiamo due numeri binari:

  00101110 = 46

  + 01001101 = 77

  01111011 = 123

- come lo abbiamo fatto? Definiamo le nostre “regole”:

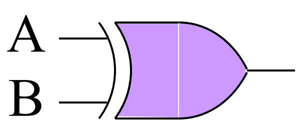
  0 + 0 = 0;

  0 + 1 = 1 + 0 = 1;

  1 + 1 = 10 (2): (0, riporto 1);

  1 + 1 + (1 di riporto) = 11 (3): (1, riporto 1)

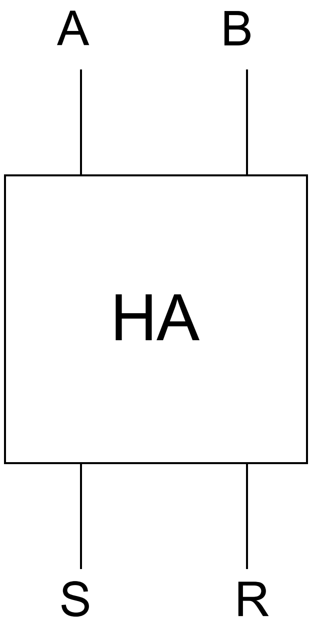
- proviamo ad associare una porta logica a queste “regole”
  - essendo una somma pensiamo subito alla OR
  - il caso “’1+1=0 (riporto1)’ si adatta meglio alla XOR
  - ancora manca le gestione del riporto



XOR

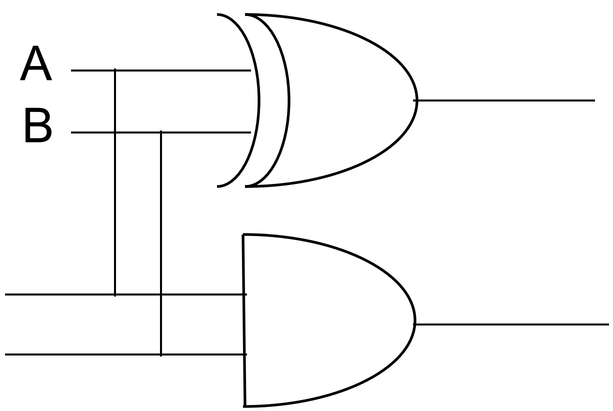
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Half Adder



A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

XOR AND



$$S = A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$$

$$R = A \cdot B$$

Half Adder

Somma binaria è analoga alla somma decimale:

1) sommare i due bit corrispondenti al digit 2^n
2) sommare il risultato al riporto dal digit 2^{n-1}

Il circuito sommatore a due ingressi è detto Half Adder
(ne occorrono due per fare una somma completa: riporto precedente!)

due input ➡ i bit da sommare
due output ➡ la somma e il riporto

può essere costruito con i circuiti di base

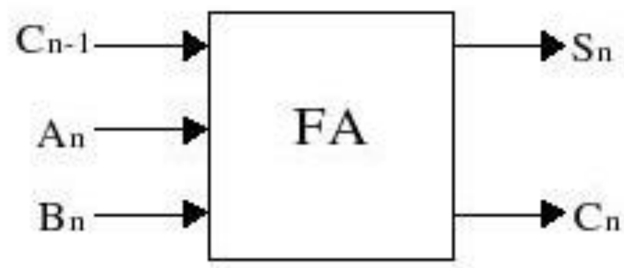
Full Adder

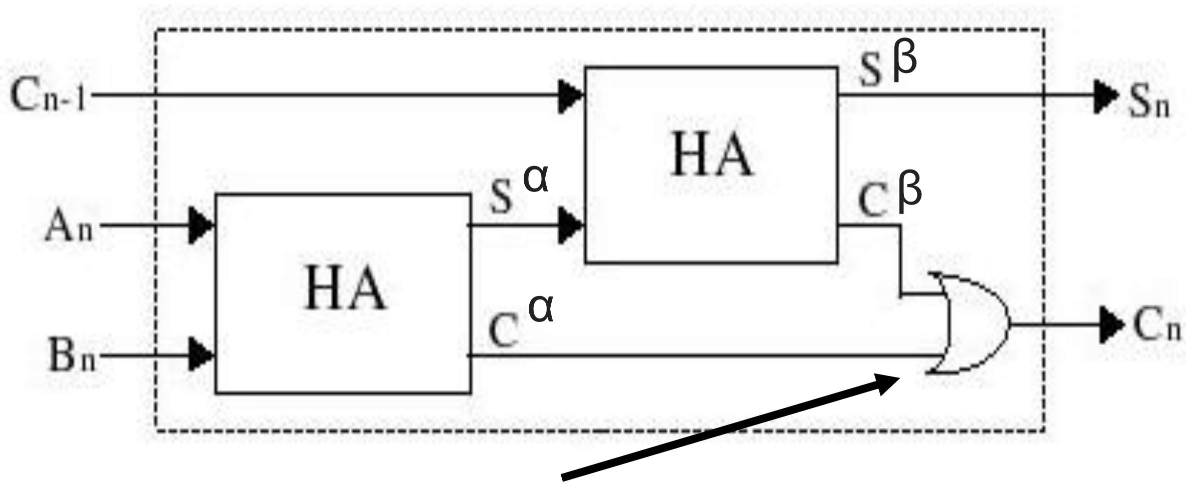
Tabella di verità della somma di 3 bit

A_n	B_n	R_{n-1}	S_n	R_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder

L'insieme di due half-adder e una porta logica OR, opportunamente collegati, restituisce un full-adder





Il caso in cui entrambi gli "C" (C^{α} e C^{β}) sono 1 sarebbe un problema (come facciamo col resto?), ma non esiste (quindi OR o XOR sono equivalenti):

- $C^{\alpha} = 1$ solo se A_n e B_n sono entrambi 1 → $S^{\alpha} = 0$
- ma se $S^{\alpha} = 0$ → $C^{\beta} = 0$, indipendentemente da C_{n-1}

Half Adder:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

Tabella di verità della somma di 3 bit

A_n	B_n	R_{n-1}	S_n	R_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A_n	B_n	R_{n-1}	S_n	R_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder

Espressione booleana corrispondente alla tabella di verità

$$S_n = \overline{A_n}\,\overline{B_n}R_{n-1} + \overline{A_n}B_n\overline{R_{n-1}} + A_n\overline{B_n}\,\overline{R_{n-1}} + A_nB_nR_{n-1}$$

$$R_n = \overline{A_n}B_nR_{n-1} + A_n\overline{B_n}R_{n-1} + A_nB_n\overline{R_{n-1}} + A_nB_nR_{n-1}$$

possiamo riscrivere R_n, sapendo che Q+Q+Q = Q

$$R_n = \left(\overline{A_n}B_nR_{n-1} + A_nB_nR_{n-1}\right) + \left(A_n\overline{B_n}R_{n-1} + A_nB_nR_{n-1}\right) + \left(A_nB_n\overline{R_{n-1}} + A_nB_nR_{n-1}\right)$$

$$R_n = \left(\overline{A_n} + A_n\right)B_nR_{n-1} + \left(\overline{B_n} + B_n\right)A_nR_{n-1} + \left(\overline{R_{n-1}} + R_{n-1}\right)A_nB_n$$

$$R_n = B_nR_{n-1} + A_nR_{n-1} + A_nB_n = \boxed{A_nB_n + \left(A_n + B_n\right)R_{n-1}}$$

Full Adder

$$S_n = \overline{A_n}\,\overline{B_n}R_{n-1} + \overline{A_n}B_n\overline{R_{n-1}} + A_n\overline{B_n}\,\overline{R_{n-1}} + A_nB_nR_{n-1}$$

$$R_n = \overline{A_n}B_nR_{n-1} + A_n\overline{B_n}R_{n-1} + A_nB_n\overline{R_{n-1}} + A_nB_nR_{n-1}$$

possiamo riscrivere la somma S_n

$$S_n = R_{n-1}\left(A_nB_n + \overline{A_n}\,\overline{B_n}\right) + \overline{R_{n-1}}\left(\overline{A_n}B_n + A_n\overline{B_n}\right)$$

ma $A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$ (la tab. di verità della XOR è simmetrica)

$$\left(A_nB_n + \overline{A_n}\,\overline{B_n}\right) = \overline{A_n \oplus B_n}$$

$$\left(\overline{A_n}B_n + A_n\overline{B_n}\right) = A_n \oplus B_n$$

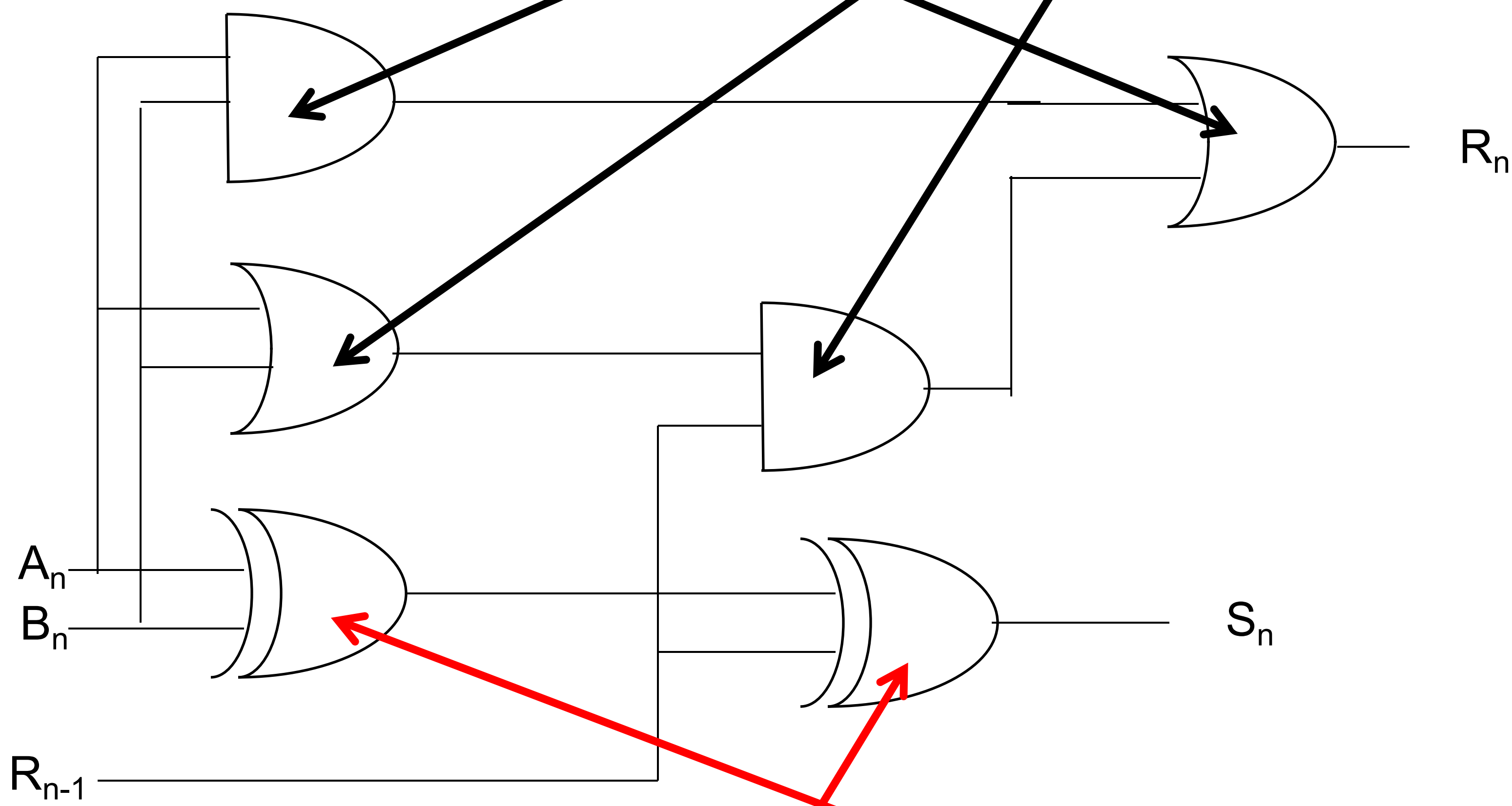
quindi

$$S_n = R_{n-1} \cdot \overline{A_n \oplus B_n} + \overline{R_{n-1}} \cdot A_n \oplus B_n$$

$$\boxed{S_n = R_{n-1} \oplus \left(A_n \oplus B_n\right)}$$

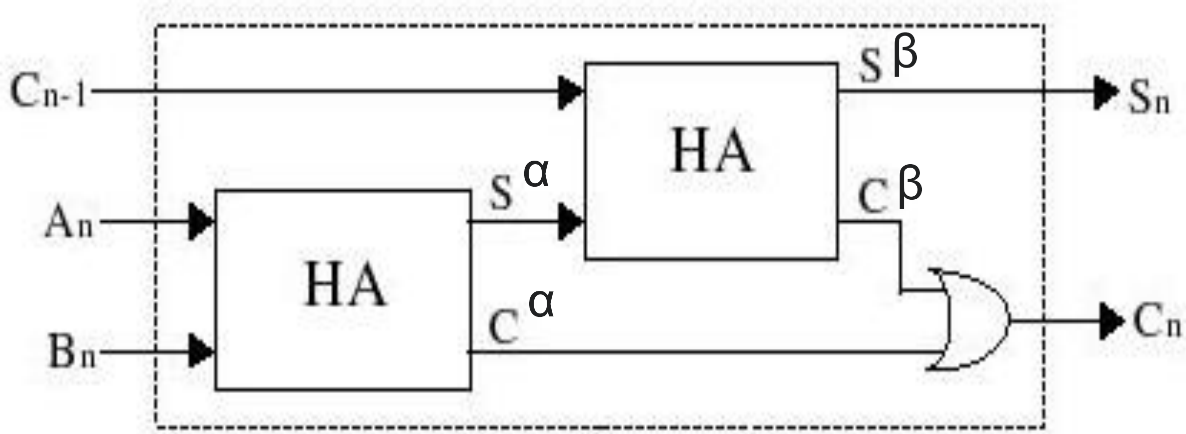
Full Adder - circuito

$$R_n = A_n B_n + \left(A_n + B_n\right) R_{n-1}$$

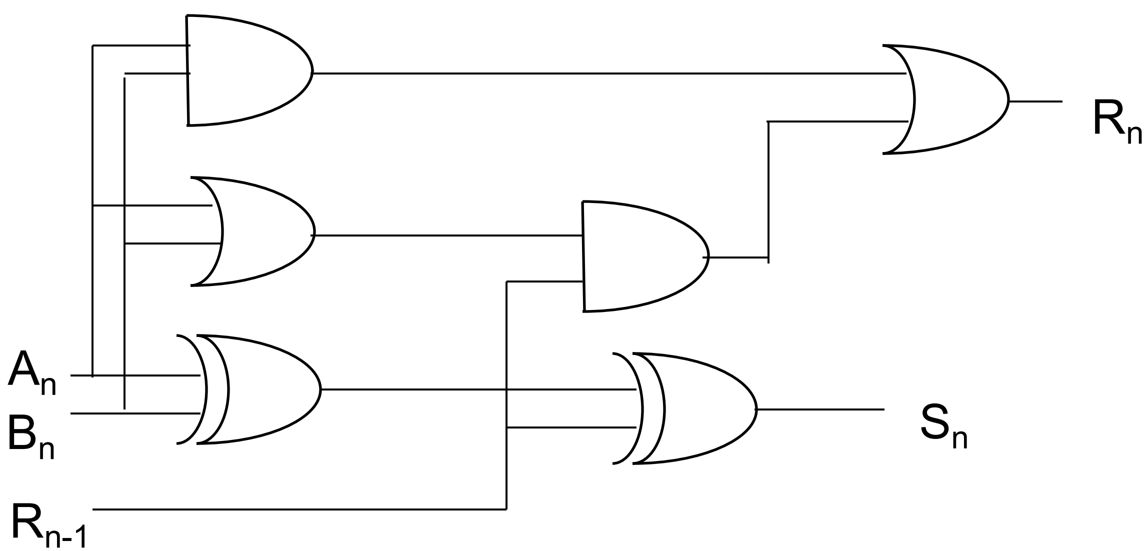


$$S_n = R_{n-1} \oplus \left(A_n \oplus B_n\right)$$

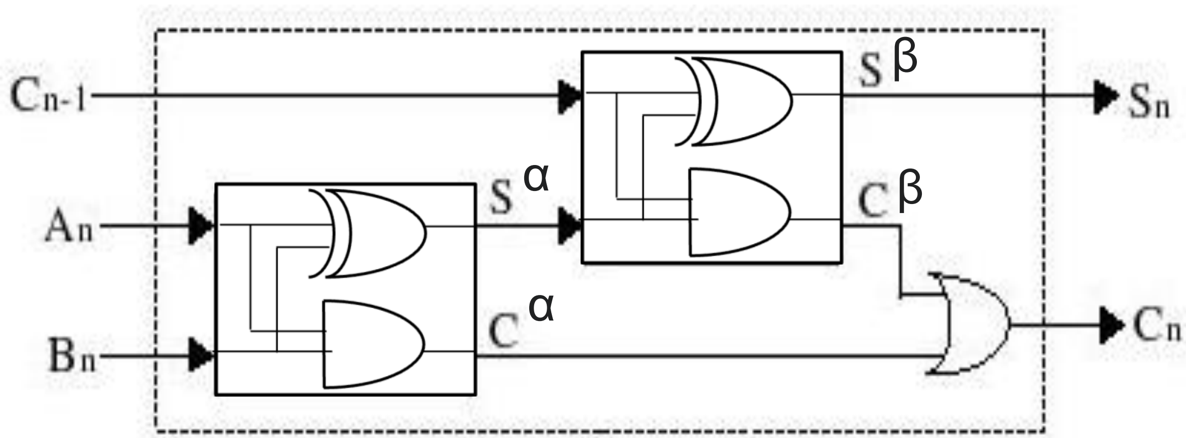
Full Adder



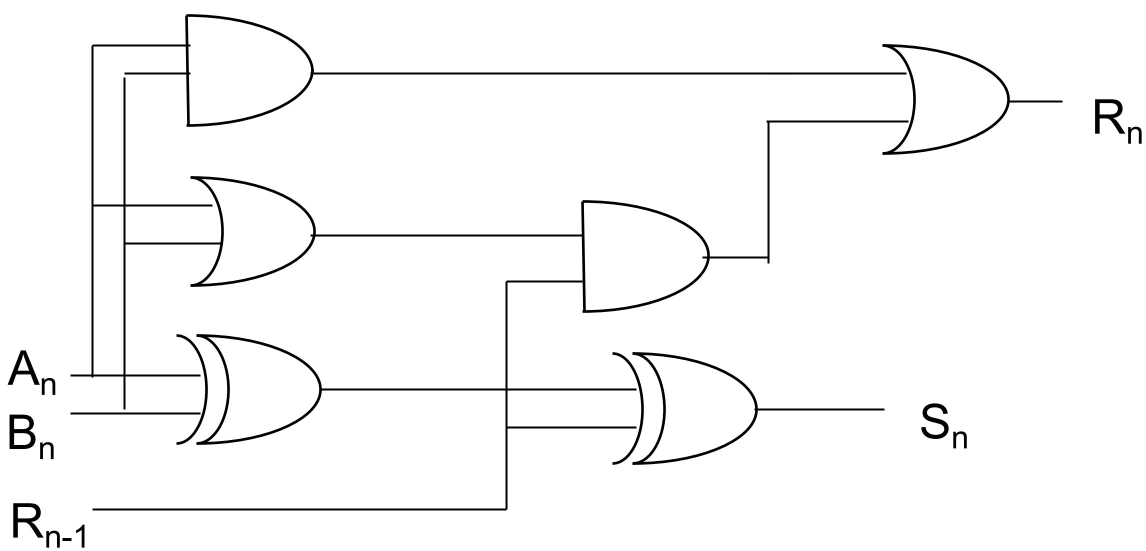
vs.



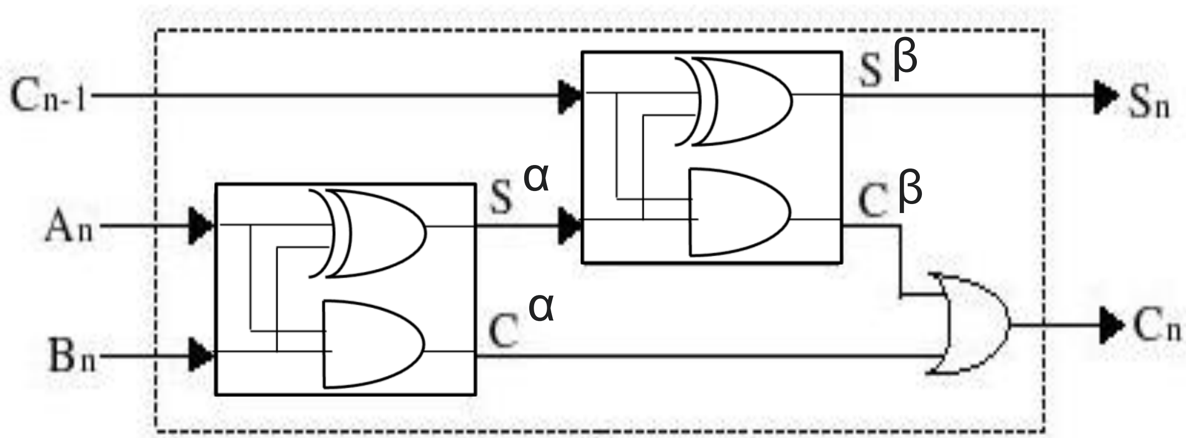
Full Adder



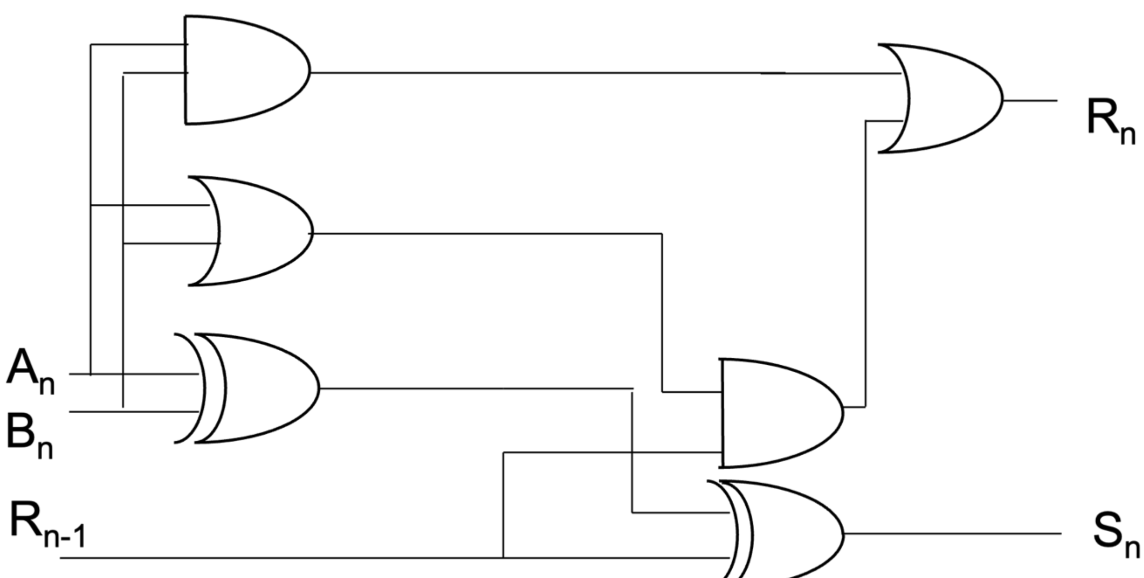
vs.



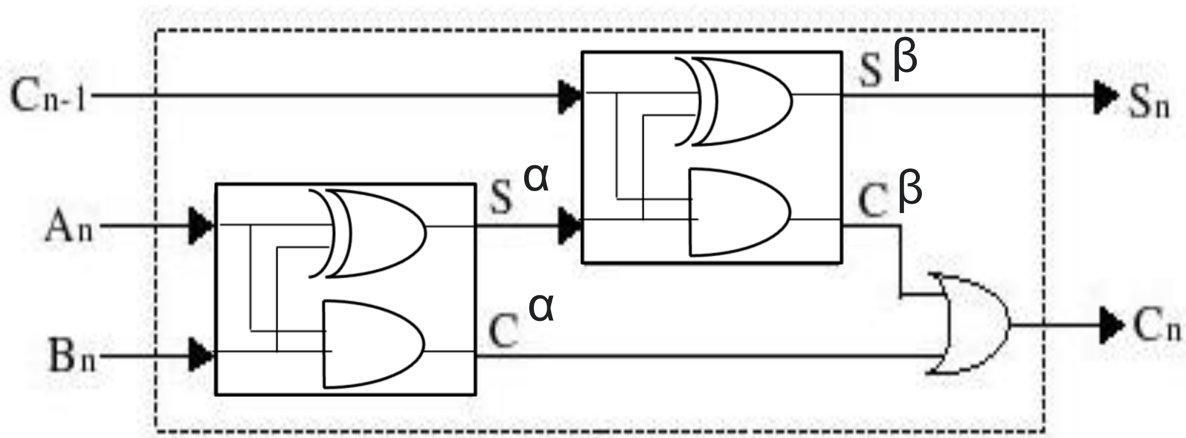
Full Adder



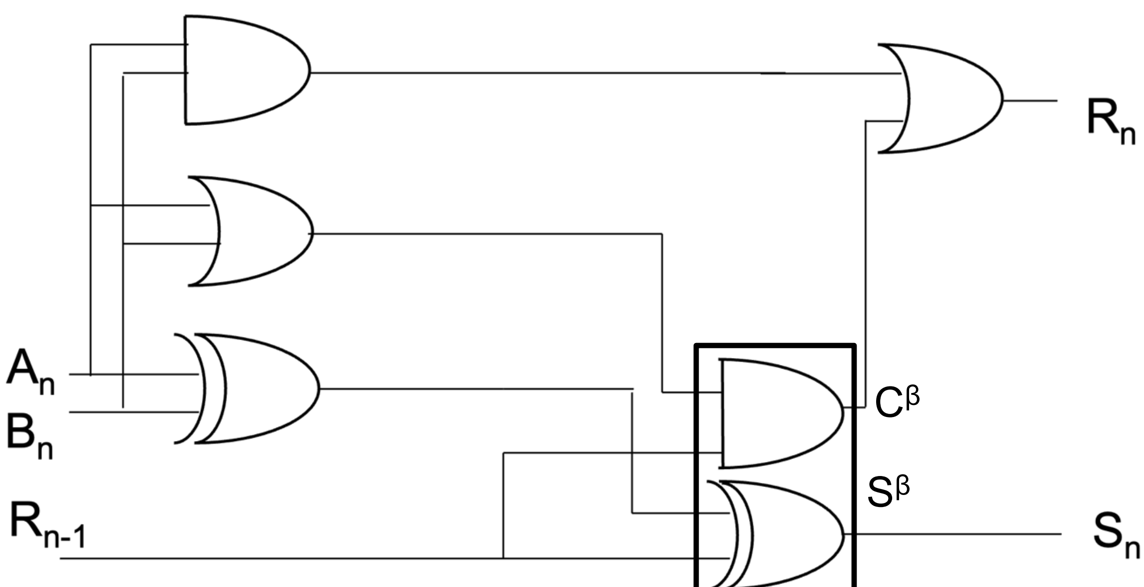
vs.



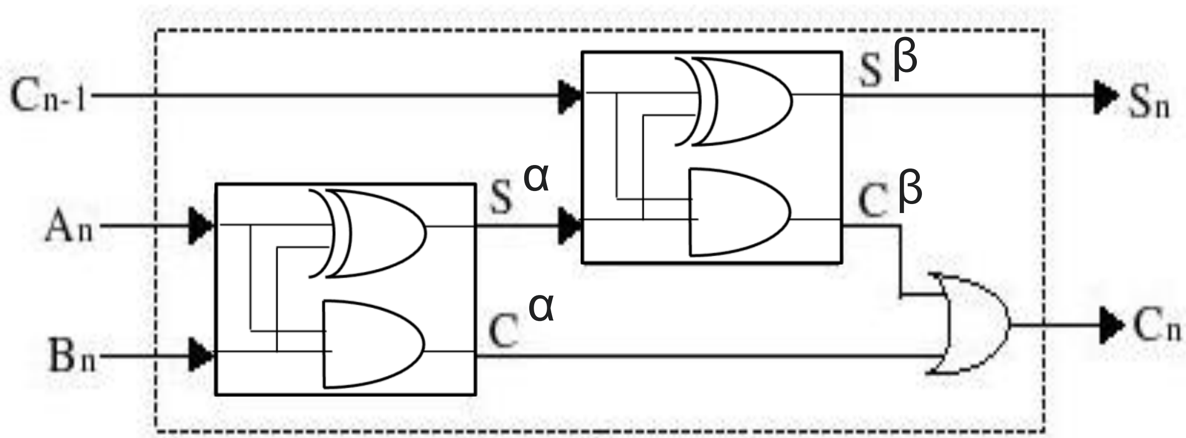
Full Adder



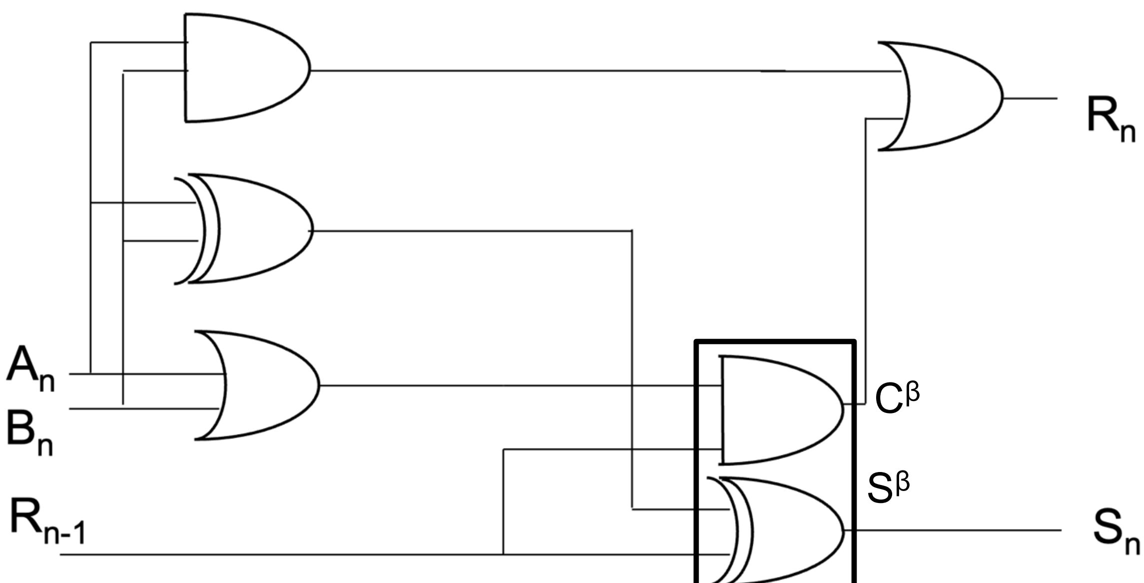
vs.



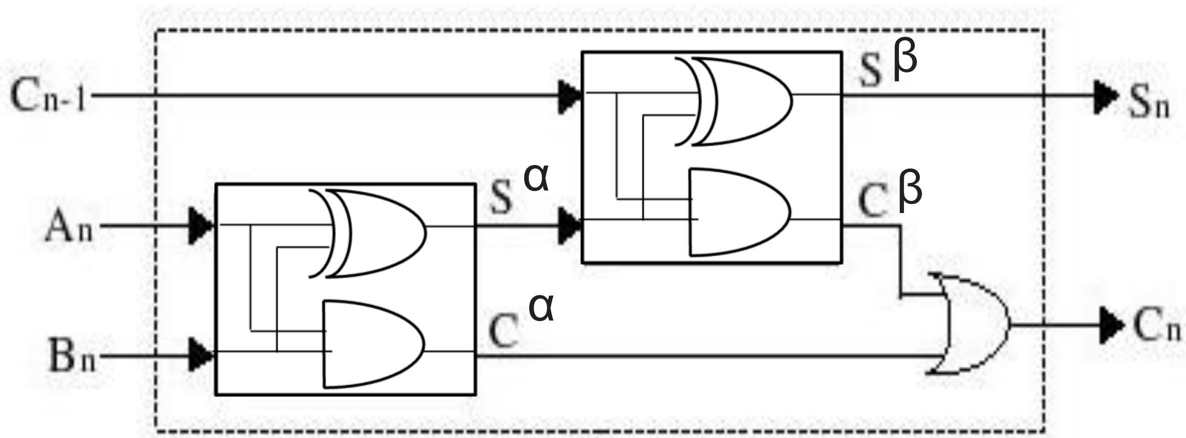
Full Adder



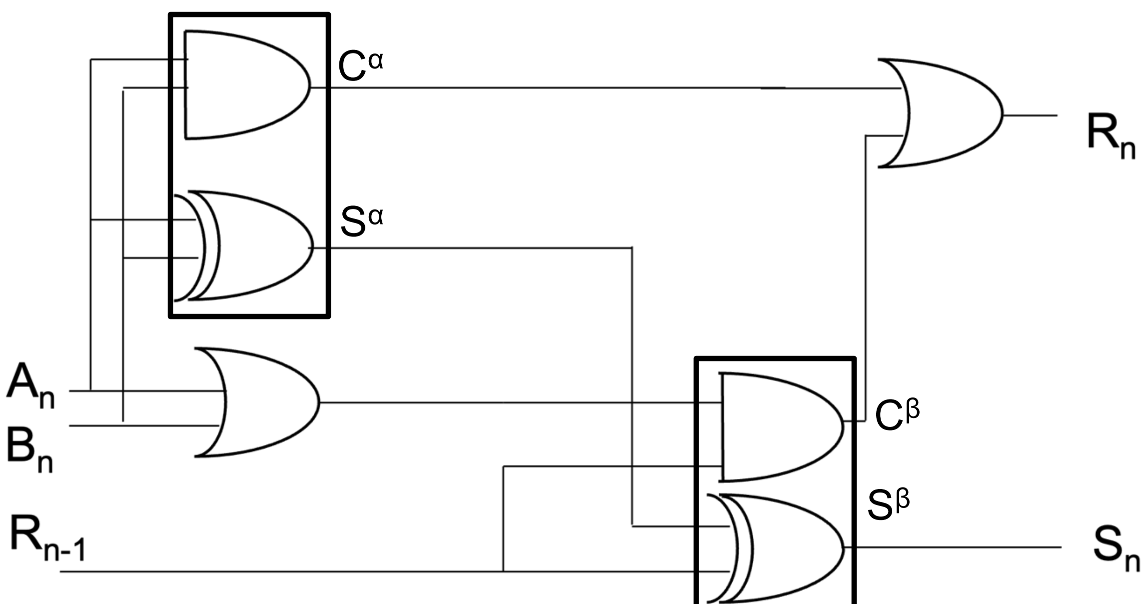
vs.



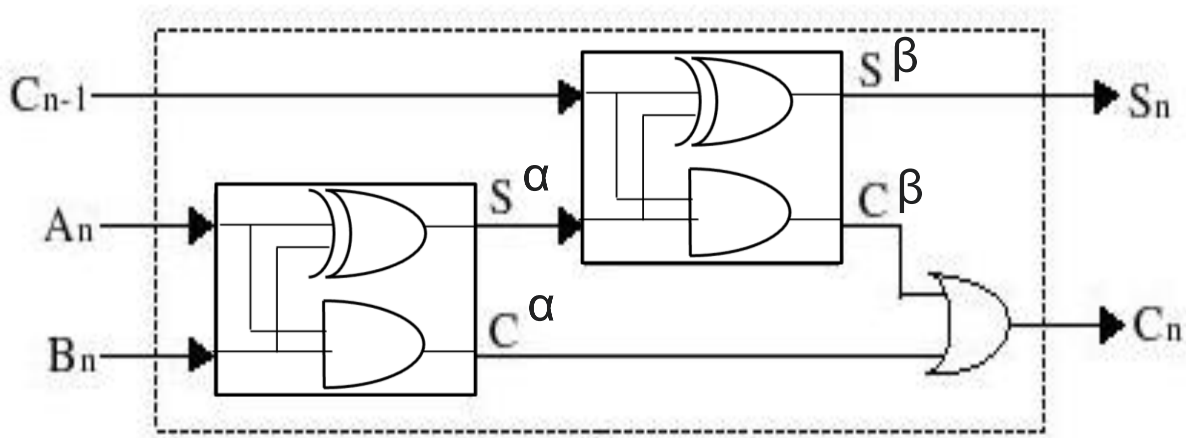
Full Adder



vs.



Full Adder



Nella AND del secondo HA (HA^β) entrano:

C_{n-1}

e

S^α

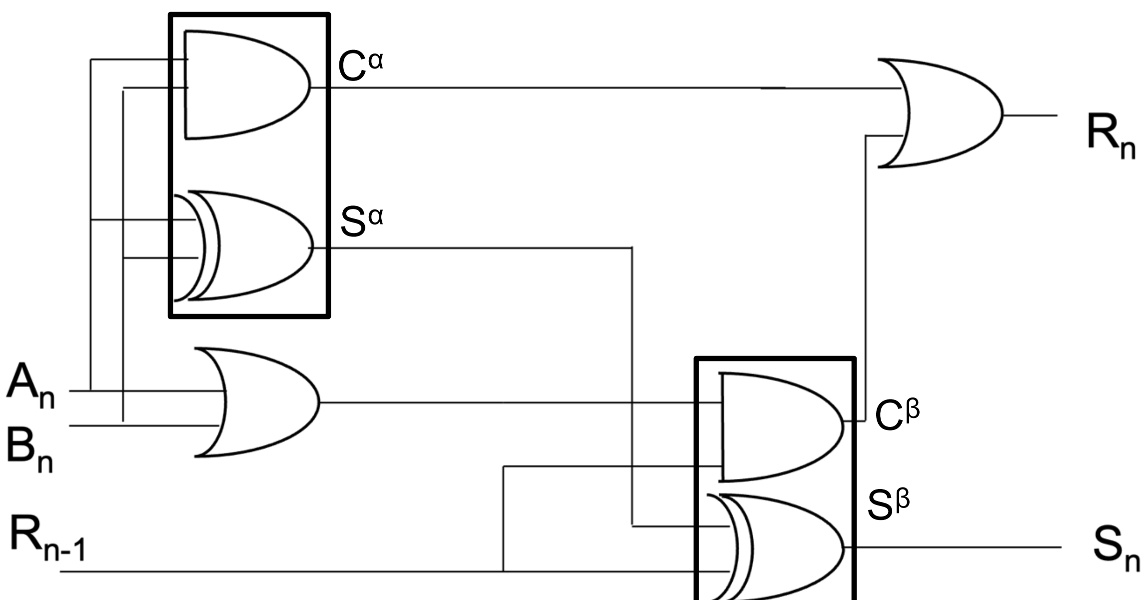
(cioè $A_n \oplus B_n$)

e poi C^β va in OR con

C^α

(cioè A_nB_n)

vs.



Nella AND del secondo HA (HA^β) entrano:

C_{n-1}

e

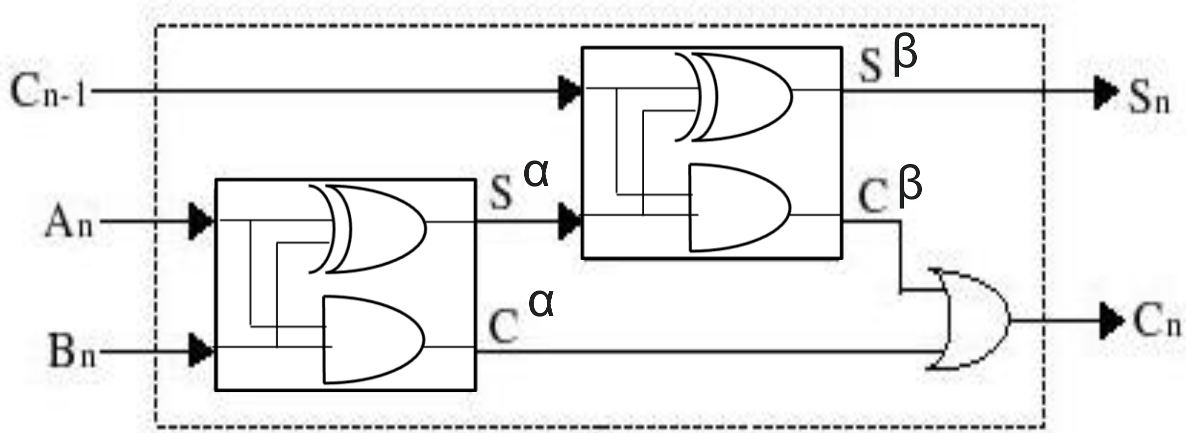
$A_n + B_n$

e poi C^β va in OR con

C^α

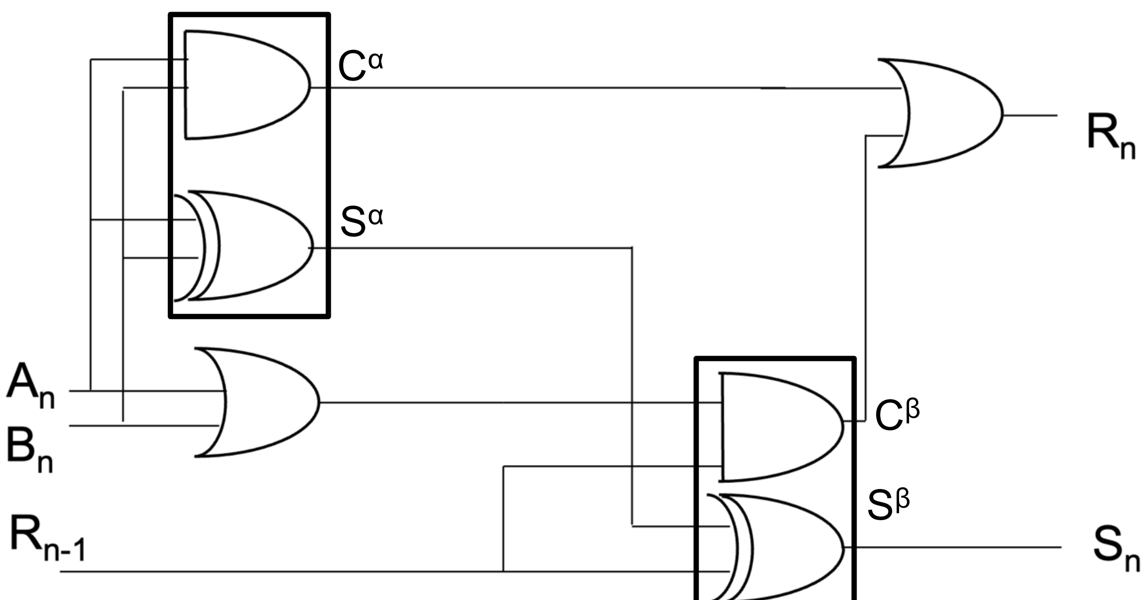
(cioè A_nB_n)

Full Adder



$C_{n-1}(A_n \oplus B_n) + A_nB_n$
=?
$C_{n-1}(A_n + B_n) + A_nB_n$

vs.

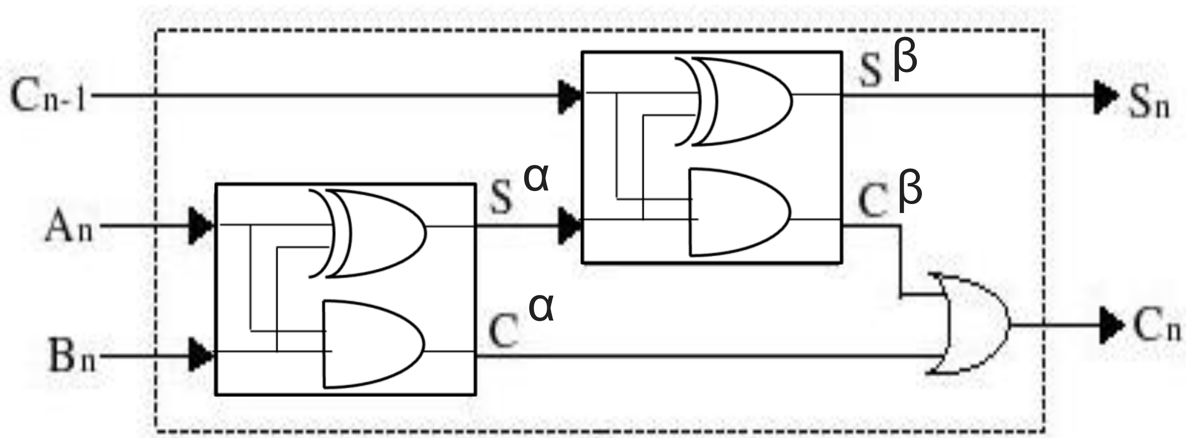


l'unica differenza potrebbe esserci se A_n e B_n sono entrambi 1.
In questo caso:

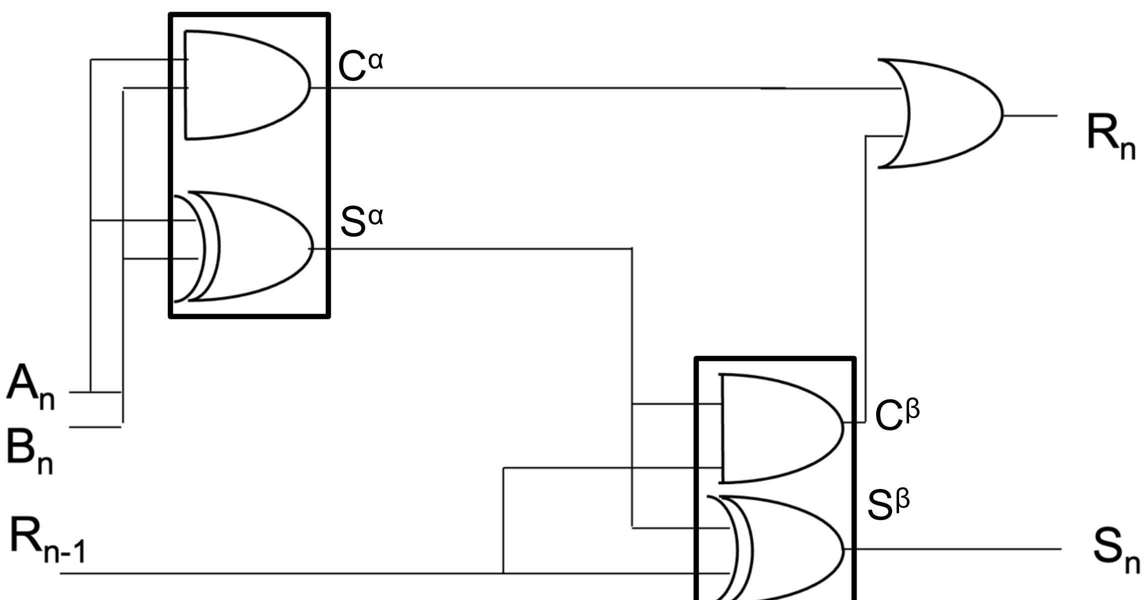
$$A_nB_n = C^{\alpha} = 1$$

a questo punto, dato che questo è in OR non è importante con cosa lo sia

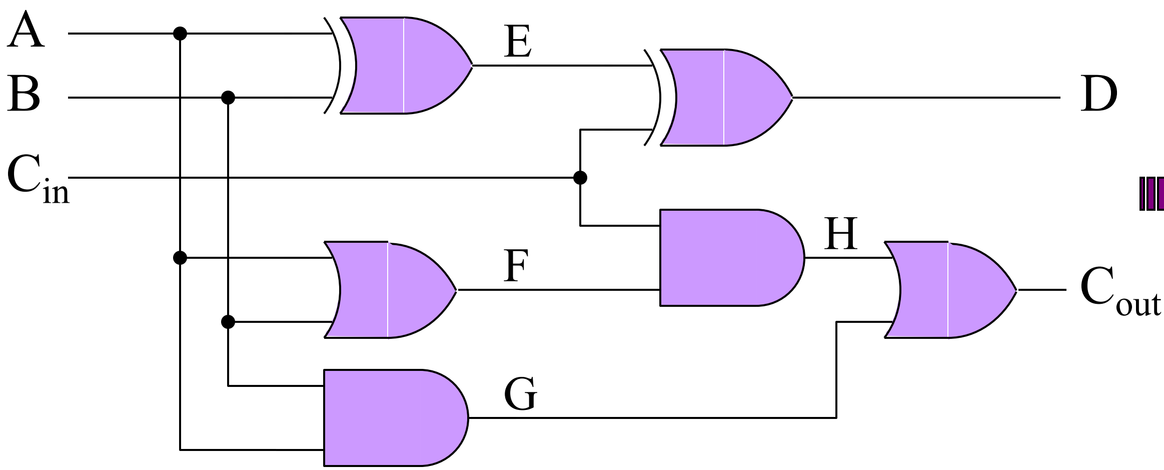
Full Adder

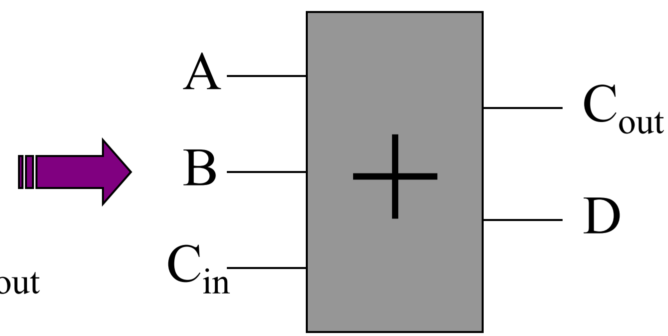


vs.



Aritmetica binaria vs transistor



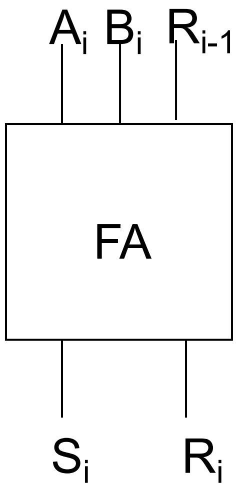


Chip “integrato”

Input			Intermedi				Output	
A	B	C_{in}	E	F	H	G	D	C_{out}
0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0
1	0	0	1	1	0	0	1	0
1	1	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0
0	1	1	1	1	1	0	0	1
1	0	1	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1

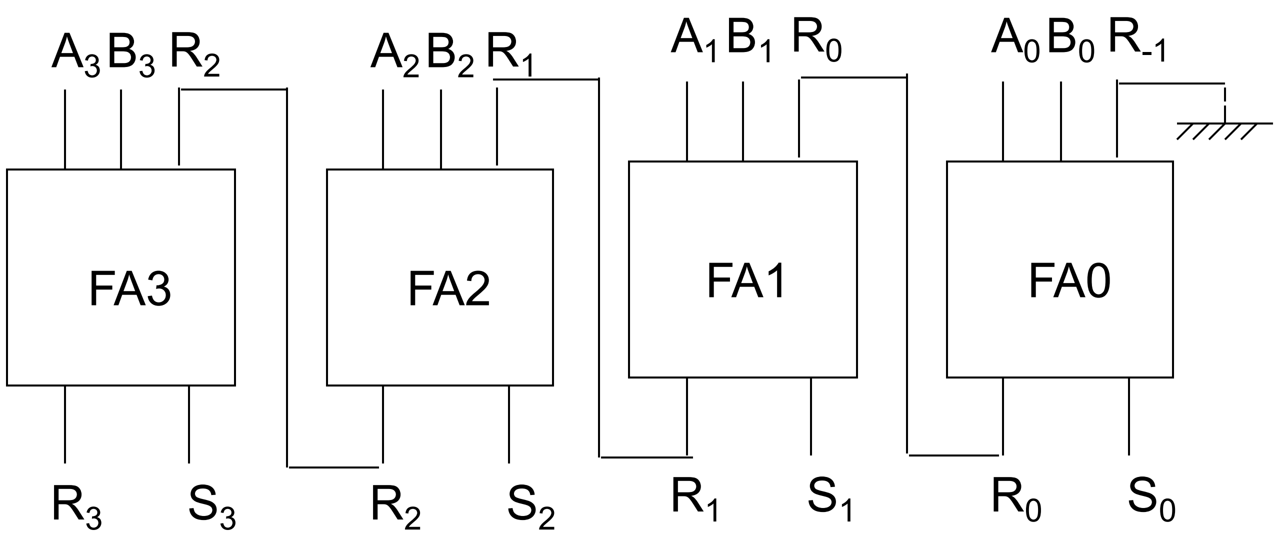
- ogni cifra richiede 6 porte
- ogni porta ha ~ 6 transistor
- ~ 36 transistor per cifra

Full Adder

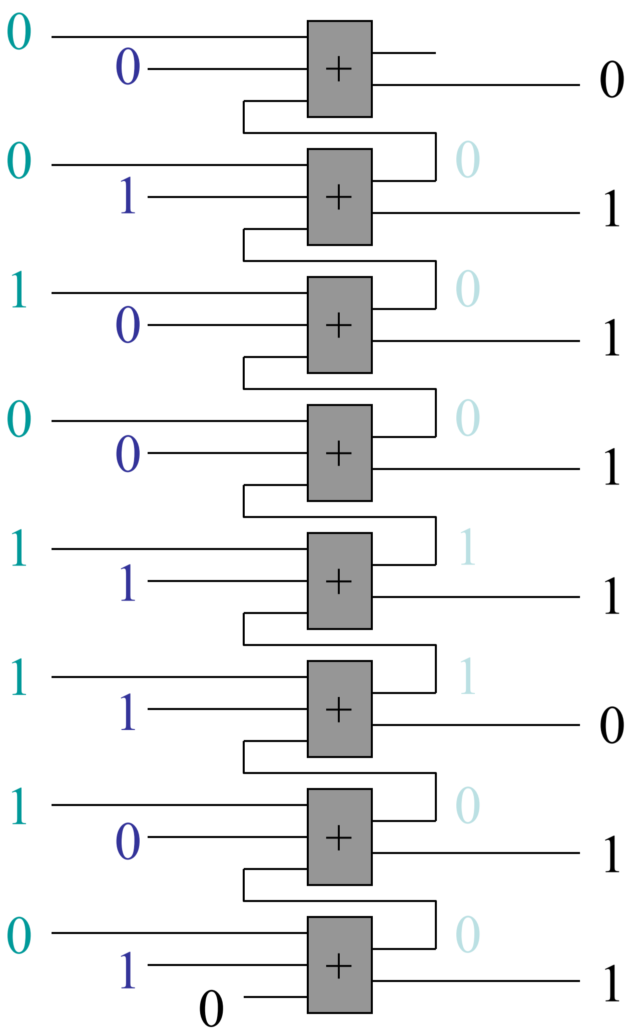


3 input e 2 output

Una somma di 4 bit può essere eseguita in parallelo usando 4 Full Adders



Aritmetica binaria a 8 bit (in cascata)



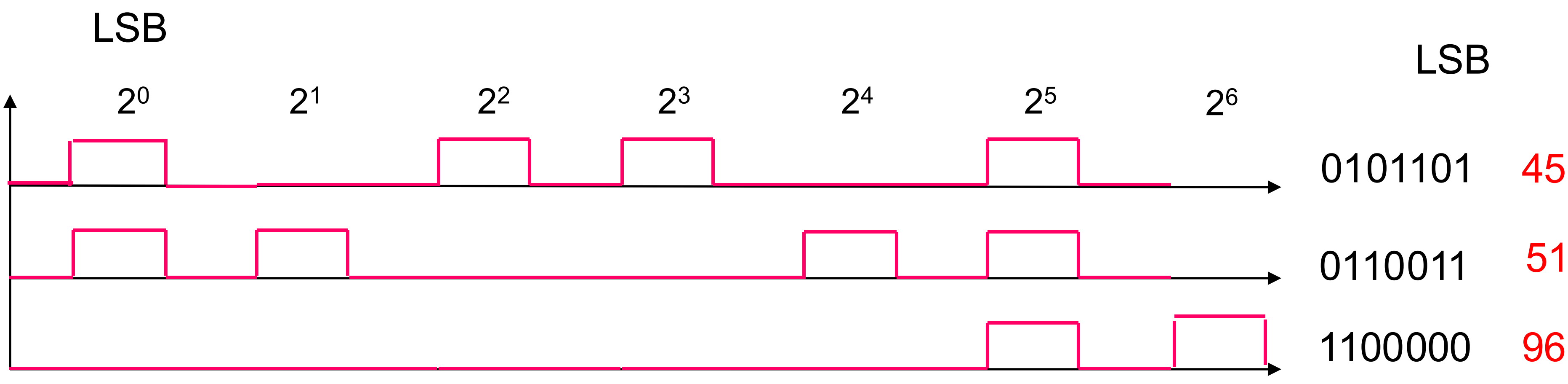
MSB = Most Significant Bit

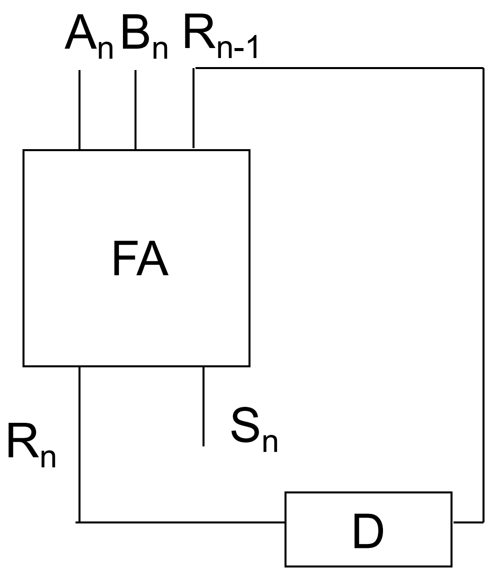
$$
\begin{array}{r}
{\scriptstyle 11} \\
00101110 \;=\; 46 \\
+\,\underline{01001101 \;=\; 77} \\
01111011 \;=\; 123
\end{array}
$$

- il riporto è utilizzato nel successivo stadio
- somma di due numeri a 8 bit
- ~300 transistor per fare questa operazione di base (+). Poi però ci sono anche –, ×, /, etc…

LSB = Least Significant Bit

Somma seriale





Una unità di ritardo in più
D = T fra gli impulsi

impulso di riporto in tempo
con i bit da sommare

Famiglie logiche

Famiglie logiche più diffuse e usate

- **CMOS** (Complementary MOS)
- **NMOS** (MOSFET a canale n)
- **TTL** (Transistor-Transistor Logic)
- **ECL** (Emitter Coupled Logic)

transistor **FET**

transistor **BJT**

Le porte logiche possono essere fabbricate con le varie tecnologie in un singolo chip con stesse funzioni, compatibili

numero di porte →

SSI small scale integration (1-10 gates)
MSI medium scale integration (10-100 gates)
LSI large scale integration (~ 10^3)
VLSI very large scale integration (~ 10^6)
ULSI ultra large scale integration (> 10^6)

Famiglie logiche (le due più comuni)

- TTL: Transistor-Transistor Logic, basato sul BJT
  - output: ‘1’ logico: $V_{OH} > 3.3$ V; ‘0’ logico: $V_{OL} < 0.35$ V
  - input: ‘1’ logico: $V_{IH} > 2.0$ V; ‘0’ logico: $V_{IL} < 0.8$ V
  - zona “morta” fra 0.8V e 2.0 V

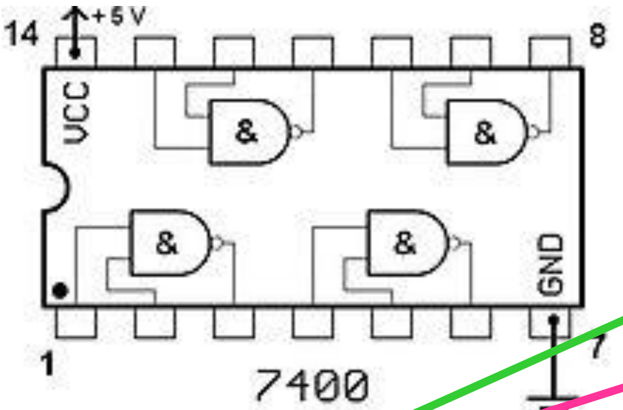
- CMOS: Complimentary MOSFET, basato su FET
  - output: ‘1’ logico: $V_{OH} > 4.7$ V; ‘0’ logico: $V_{OL} < 0.2$ V
  - input: ‘1’ logico: $V_{IH} > 3.7$ V; ‘0’ logico: $V_{IL} < 1.3$ V
  - zona “morta” fra 1.3V e 3.7 V

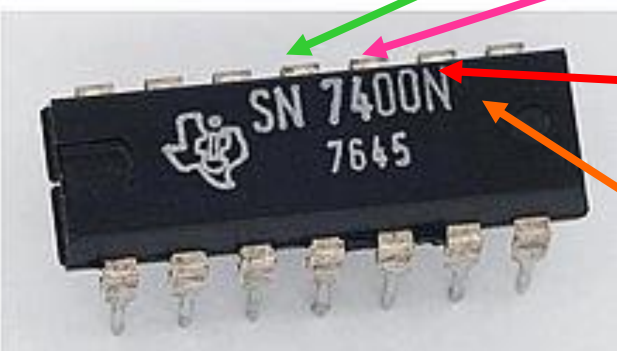
L’uscita di un CMOS è TTL-compatibile

Confronto famiglie logiche

	TTL (V)	CMOS (V)	ECL (V)
tensione massima di alimentazione	5	5	-5.2
valore massimo V_{in} identificato come 0	0.8	1.3	-1.4
valore minimo V_{in} identificato come 1	2.0	3.7	-1.2
valore massimo V_{out} identificato come 0	0.35	0.2	-1.7
valore minimo V_{out} identificato come 1	3.3	4.7	-0.9

Nomenclatura circuiti





AA 74 AAA XXX P

due lettere indicano la casa costruttrice

74, sempre uguale (porte logiche)

tre lettere che indicano la sottofamiglia

numeri indicano la funzione del circuito

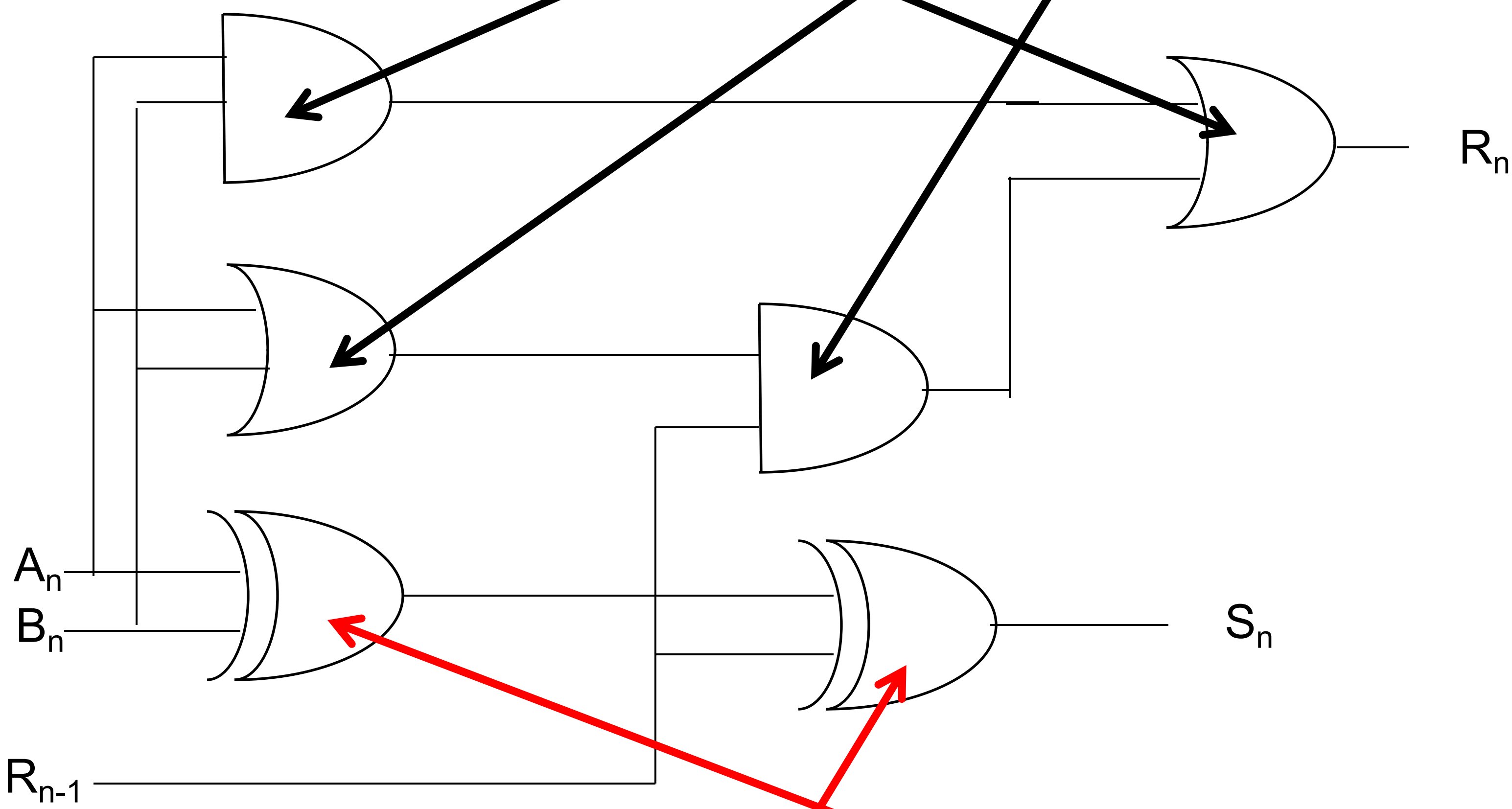
lettere che identificano il contenitore (packaging)

SN74ALS245N

significa che è fatto dalla Texas Instruments (SN), è una porta logica TTL con range di temperatura commerciale (74), è della famiglia “Advanced Low-power Schottky” (ALS), ed è un buffer bi-direzionale a 8 bit, in un package plastico di tipo through-hole DIP (N).

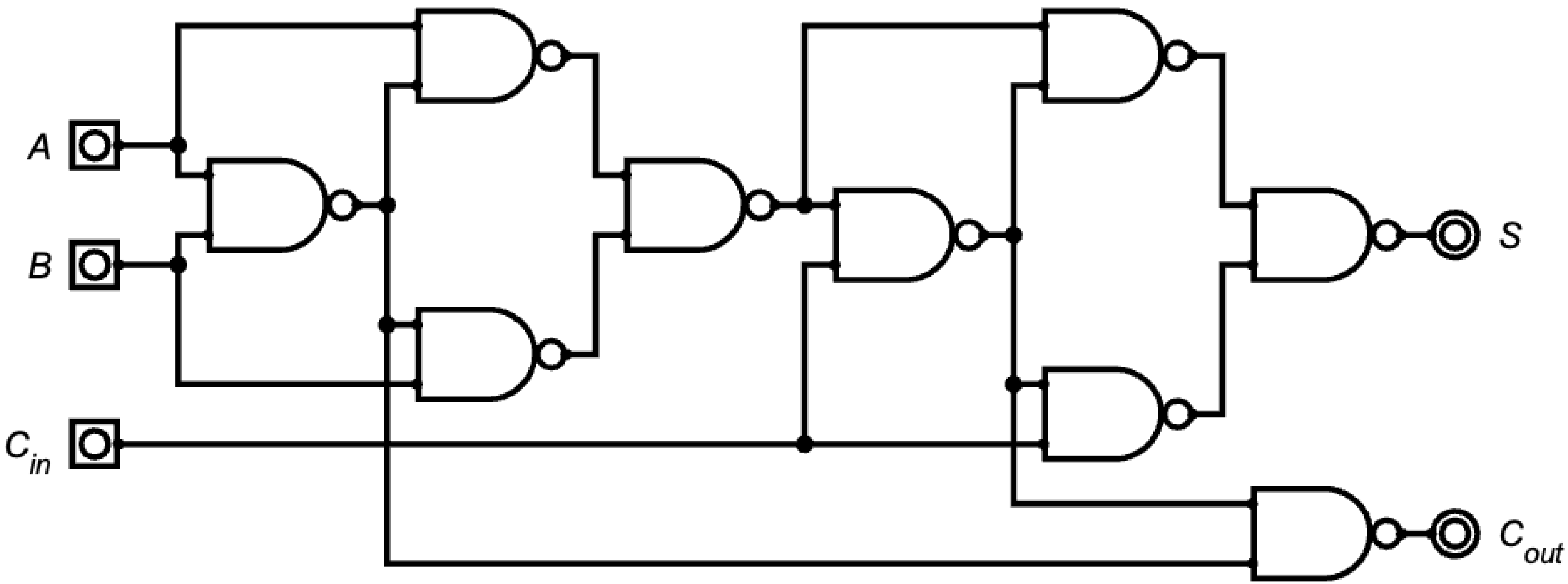
Full Adder

$$\boxed{R_n = A_n B_n + (A_n + B_n) R_{n-1}}$$

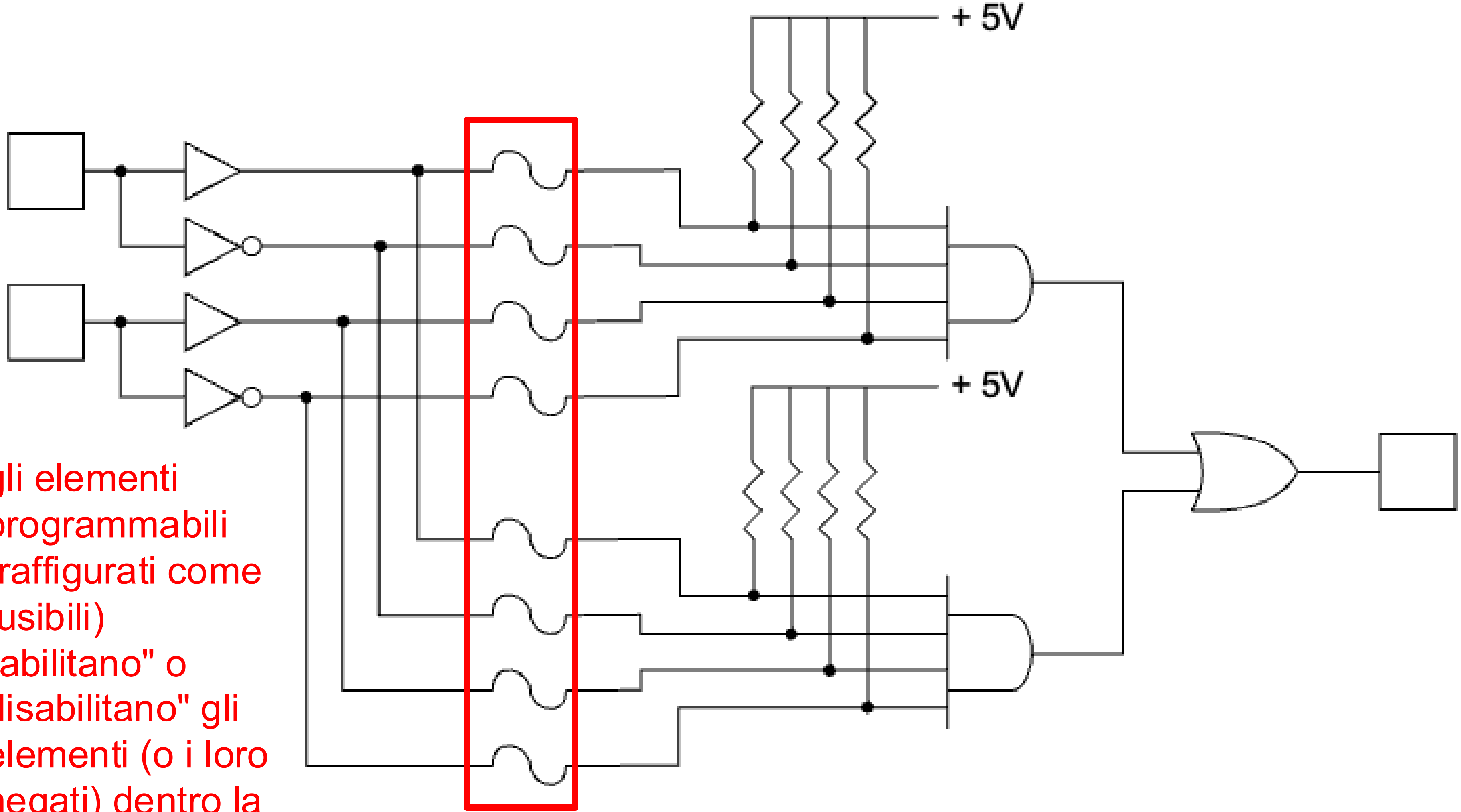


$$\boxed{S_n = R_{n-1} \oplus (A_n \oplus B_n)}$$

Full Adder - NAND



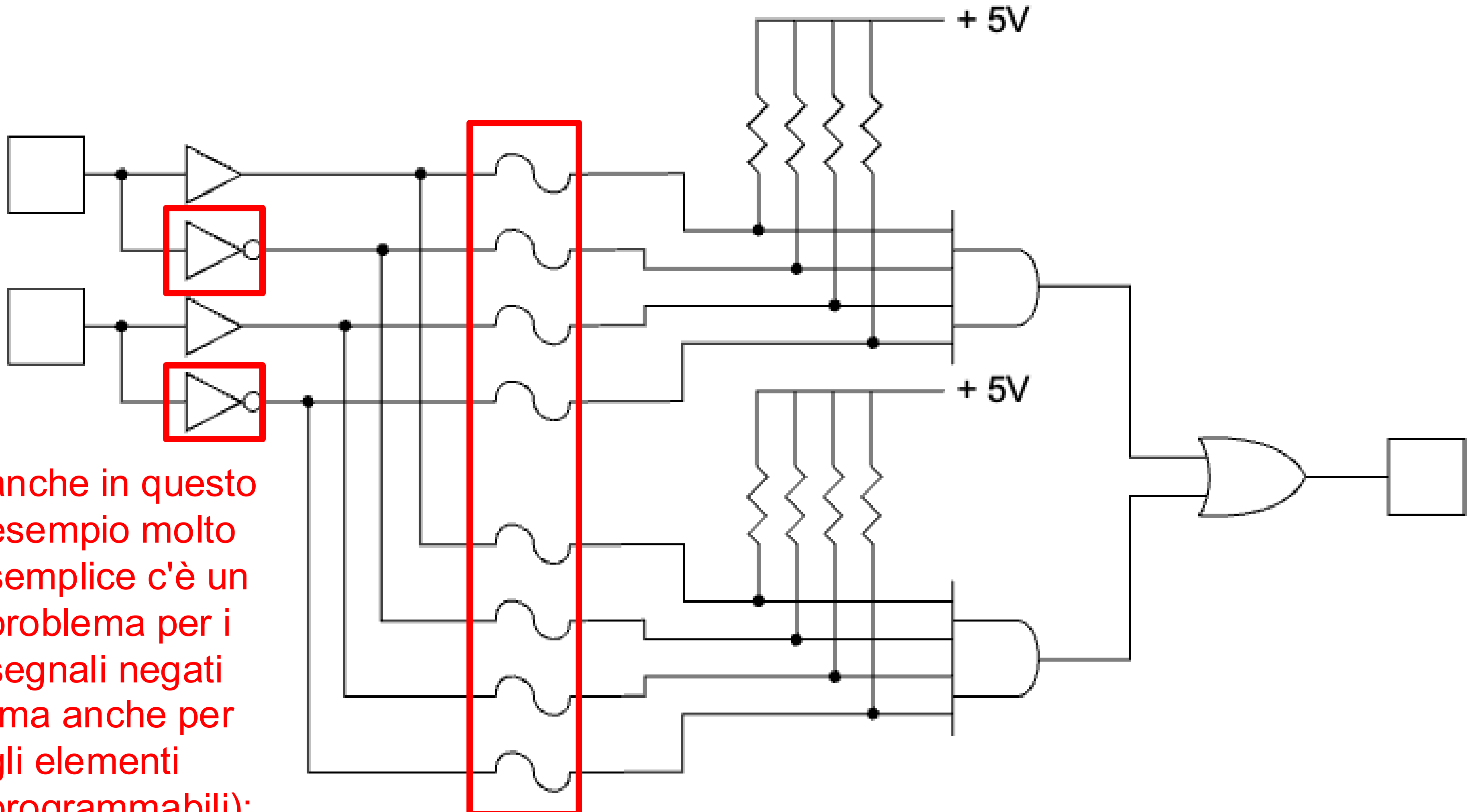
Logica programmabile



gli elementi programmabili (raffigurati come fusibili) "abilitano" o disabilitano" gli elementi (o i loro negati) dentro la AND (poi in OR fra loro)

Simplified programmable logic device

Logica programmabile

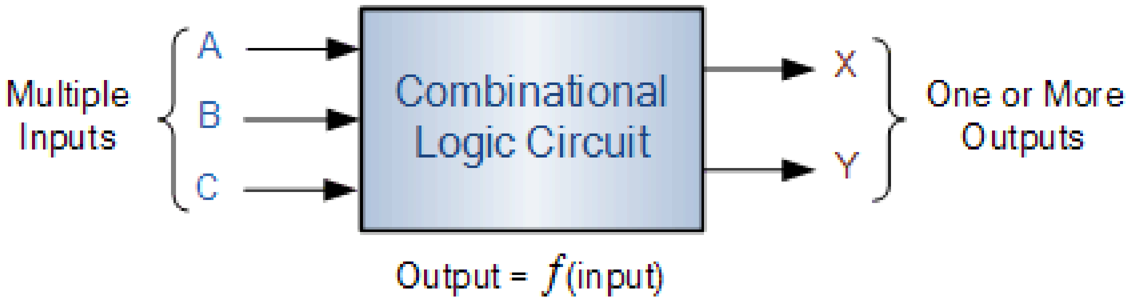


anche in questo esempio molto semplice c'è un problema per i segnali negati (ma anche per gli elementi programmabili): arrivano leggermente dopo gli altri

Simplified programmable logic device

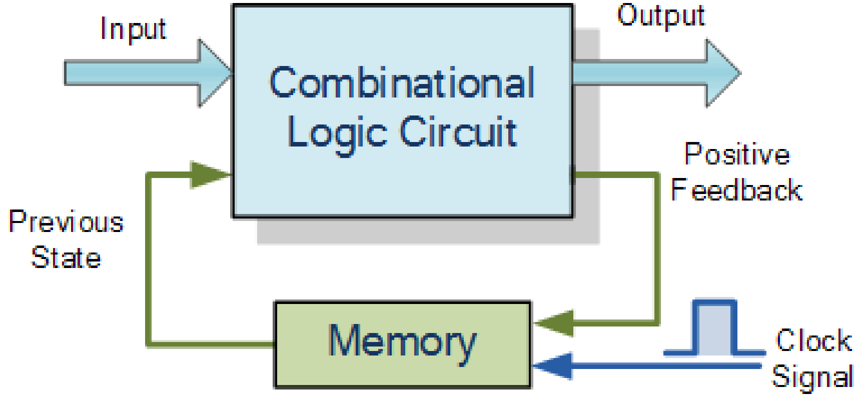
Reti Logiche e Sequenziali

Reti Logiche



- No feedback, in ogni istante l'output è funzione degli input
- Tutti i circuiti analizzati fino adesso sono reti logiche

Rete Sequenziale

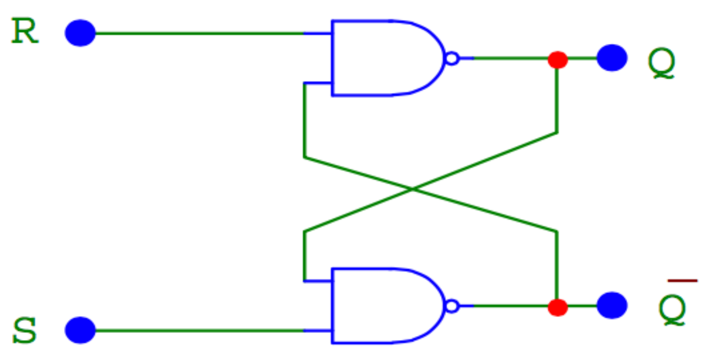


- L'output è funzione dell'input corrente e dell'output precedente
- I (possibili) cambi di output sono definiti dal segnale di **clock**
- Il circuito ha "**memoria**" dei suoi stati precedenti

Flip Flop SR (NAND)

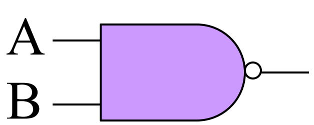
- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato "set"
1	0	0	1	stato "reset"
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

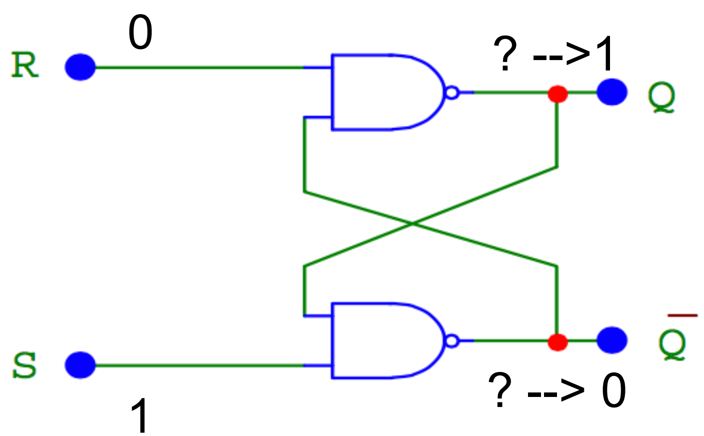


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Flip Flop SR (NAND)

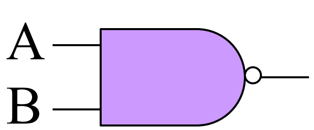
- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato "set"
1	0	0	1	stato "reset"
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

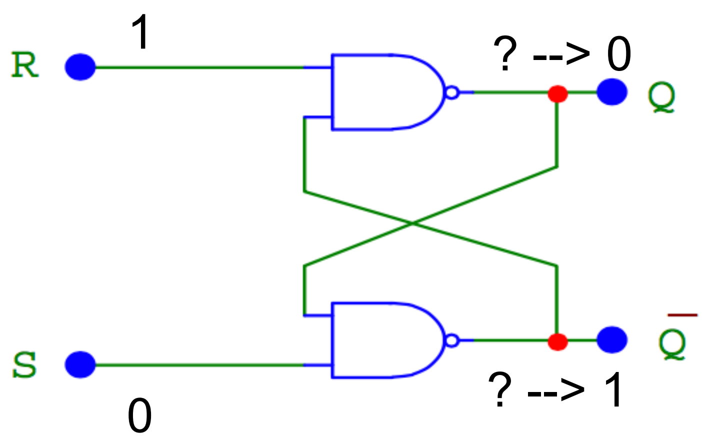


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Flip Flop SR (NAND)

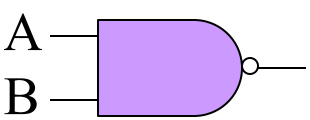
- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato “set”
1	0	0	1	stato “reset”
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

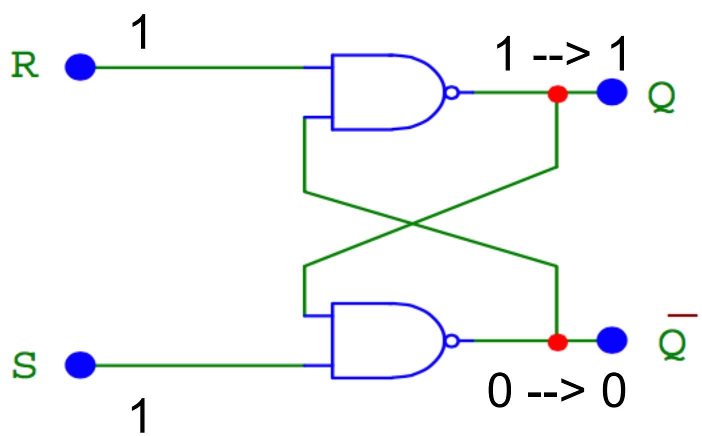


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Flip Flop SR (NAND)

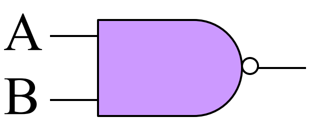
- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato "set"
1	0	0	1	stato "reset"
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

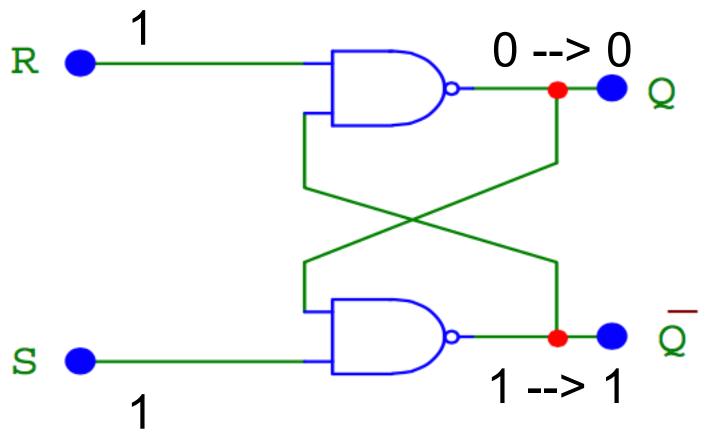


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Flip Flop SR (NAND)

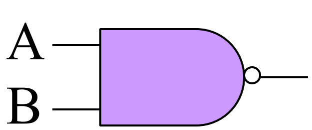
- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato "set"
1	0	0	1	stato "reset"
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

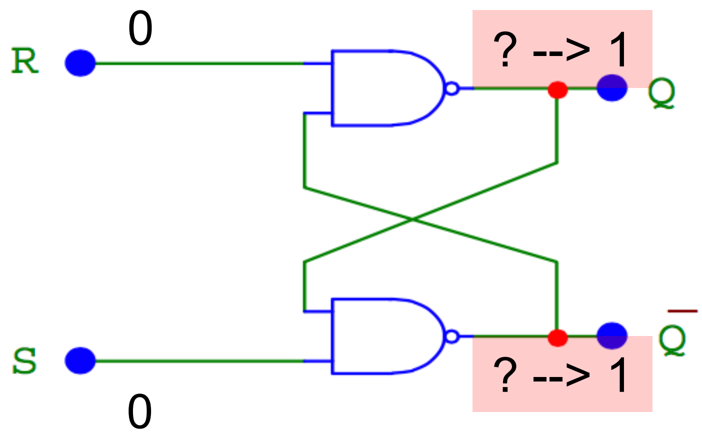


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Flip Flop SR (NAND)

- **FLIP FLOP**: unità di memoria fondamentale dei circuiti digitali. Elemento di base dei circuiti sequenziali. È un circuito che immagazzina l'informazione di base, (bit, 0 o 1)

SR Flip Flop (Set – Reset)



R	S	Q_n	$\overline{Q}_n$	
0	0	non consentito		
0	1	1	0	stato "set"
1	0	0	1	stato "reset"
1	1	Q_{n-1}	$\overline{Q}_{n-1}$	

- Asincrono: il cambio di stato dell'output avviene in corrispondenza al cambio di stato degli input

- Stato R=0, S=0 proibito:
$Q_n = 1 \;\; \overline{Q_n} = 1$ (i.e. non è vero che "*Q*" *!=* "*!Q*"): situazione anomala, è necessario evitare che il FF sia in questo stato

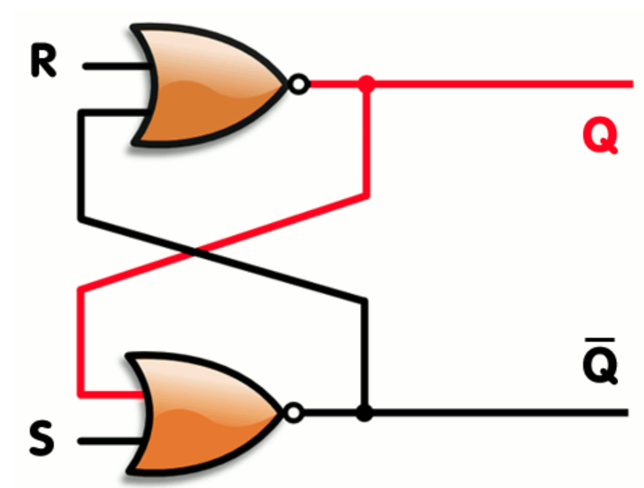
Flip Flop SR (NOR)

SR NOR latch [edit]

While the R and S inputs are both low, feedback maintains the Q and $\overline{Q}$ outputs in a constant state, with $\overline{Q}$ the complement of Q. If S (*Set*) is pulsed high while R (*Reset*) is held low, then the Q output is forced high, and stays high when S returns to low; similarly, if R is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns to low.

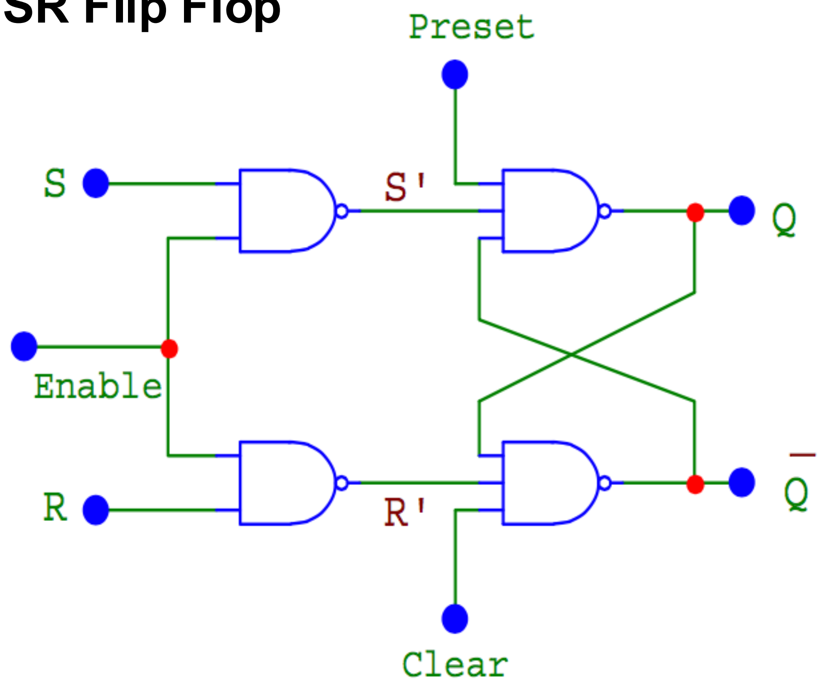
SR latch operation[3]

Characteristic table				Excitation table			
S	**R**	**Q_{next}**	**Action**	**Q**	**Q_{next}**	**S**	**R**
0	0	Q	Hold state	0	0	0	X
0	1	0	Reset	0	1	1	0
1	0	1	Set	1	0	0	1
1	1	X	Not allowed	1	1	X	0



Flip Flop SR - gated

SR Flip Flop

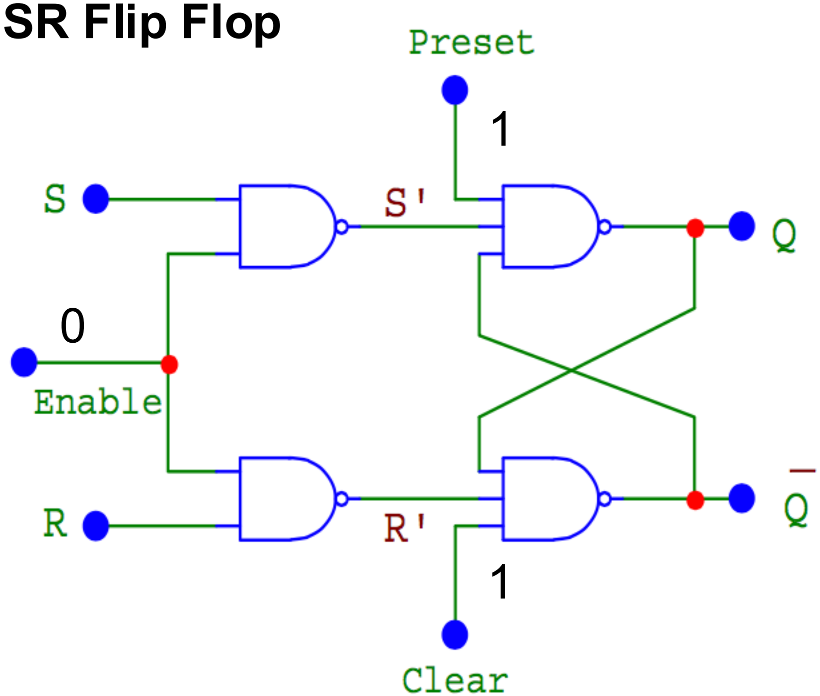


R	S	R'	S'	Q_n	$\overline{Q}_n$
x	x	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

ENABLE: gate che abilita la porta:
PRESET e **CLEAR**: gate per definire lo stato iniziale del FF

Flip Flop SR - gated

SR Flip Flop

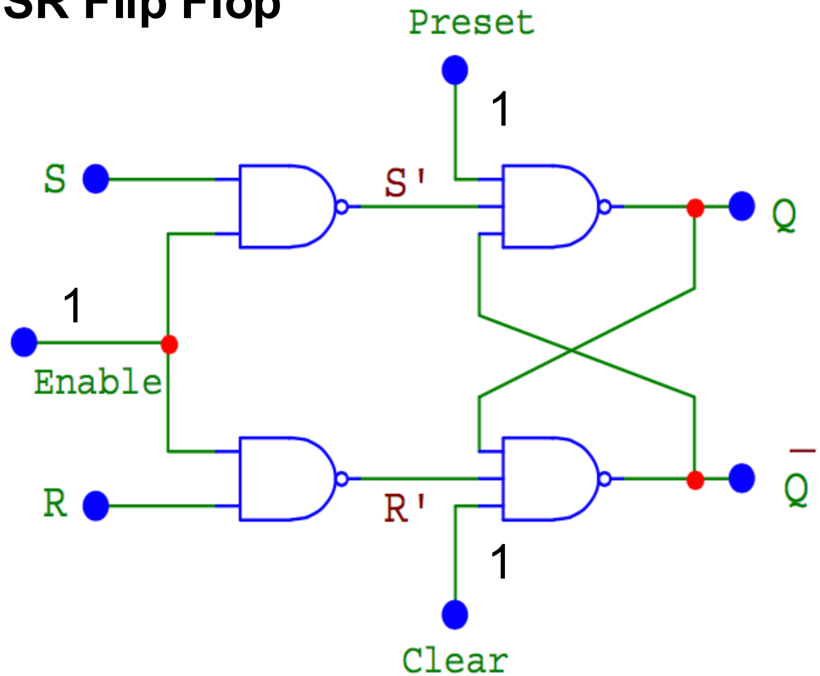


R	S	R’	S’	Q_n	$\overline{Q}_n$
x	x	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

ENABLE = 0: S’=R’=1: FF Mantiene lo stato attuale, non risponde a variazioni di S e R

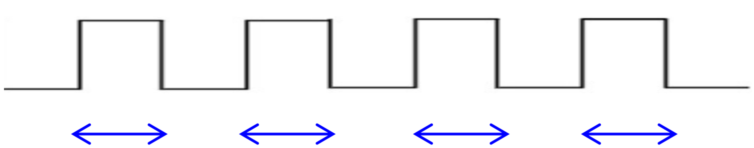
Flip Flop SR - gated

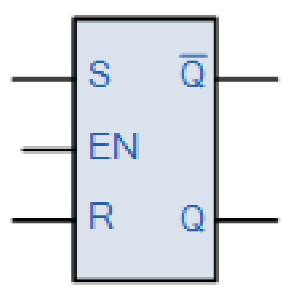
SR Flip Flop



R	S	R'	S'	Q_n	$\overline{Q}_n$
0	0	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

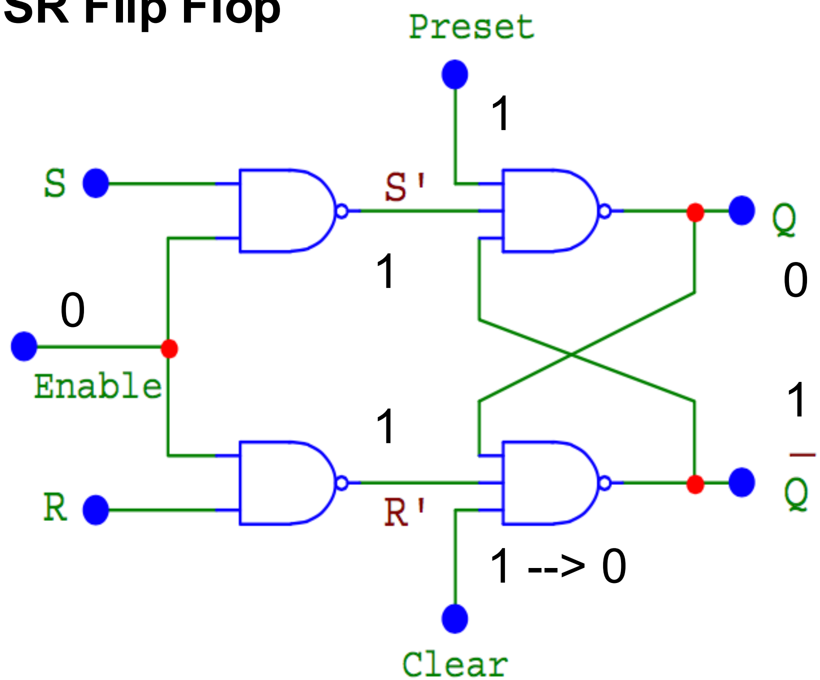
- Il gate EN permette di controllare quando il FF può cambiare stato: quando EN=0, l'uscita del FF "**memorizza**" l'output definito nel tempo in cui EN=1
- **Sincrono**: il cambio di stato di Q e !Q avviene solamente quando il segnale di enable è positivo





Flip Flop SR - gated

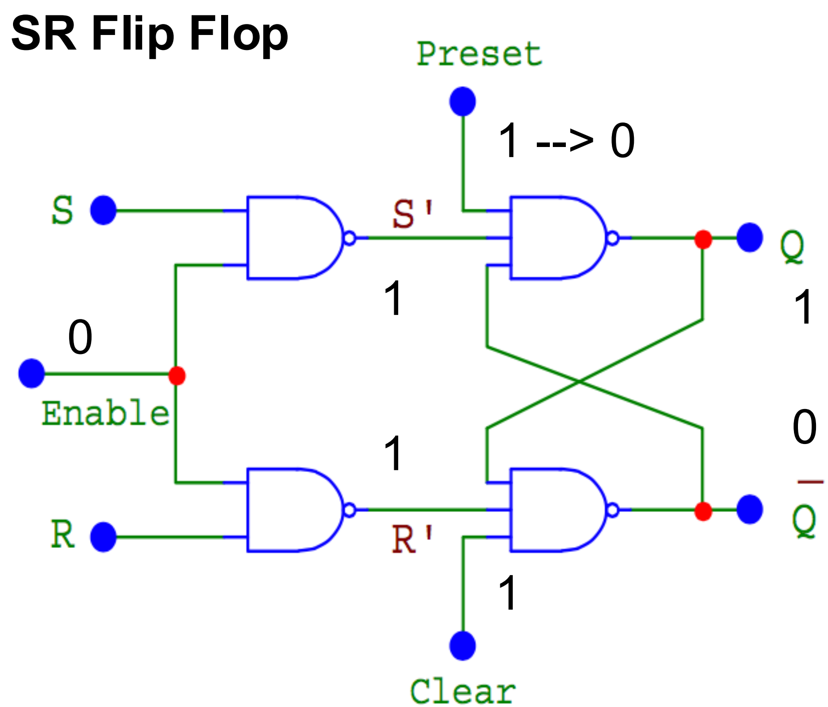
SR Flip Flop



R	S	R’	S’	Q_n	$\overline{Q}_n$
0	0	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

- Il gate EN permette di controllare quando il FF può cambiare stato: quando EN=0, l’uscita del FF “**memorizza**” l’output definito nel tempo in cui EN=1
- **Sincrono**: il cambio di stato di Q e !Q avviene solamente quando il segnale di enable è positivo

Flip Flop SR - gated

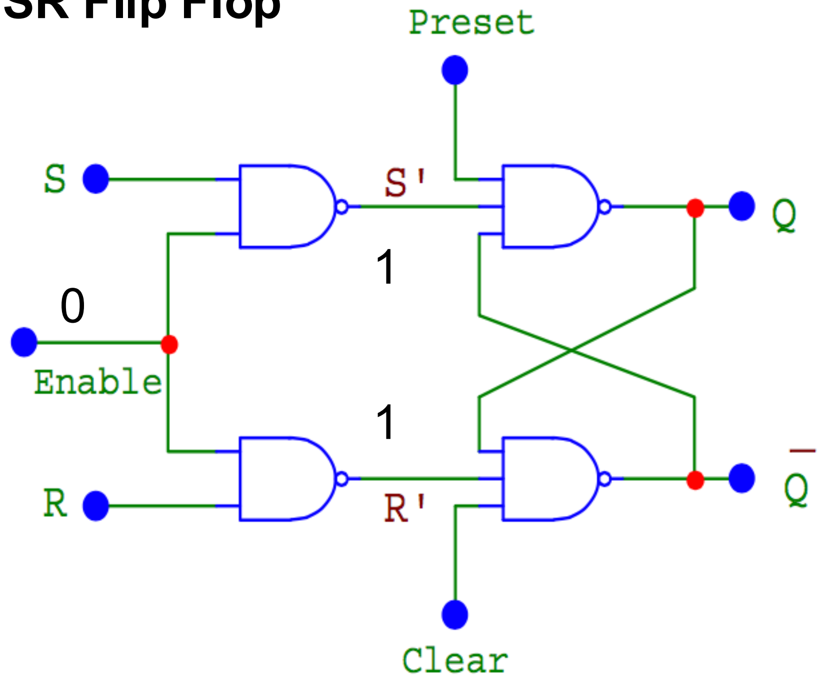


R	S	R’	S’	Q_n	$\overline{Q}_n$
0	0	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

- Il gate EN permette di controllare quando il FF può cambiare stato: quando EN=0, l’uscita del FF “**memorizza**” l’output definito nel tempo in cui EN=1
- **Sincrono**: il cambio di stato di Q e !Q avviene solamente quando il segnale di enable è positivo

Flip Flop SR - gated

SR Flip Flop



R	S	R'	S'	Q_n	$\overline{Q}_n$
0	0	1	1	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

- Il gate EN permette di controllare quando il FF può cambiare stato: quando EN=0, l'uscita del FF "**memorizza**" l'output definito nel tempo in cui EN=1
- **Sincrono**: il cambio di stato di Q e !Q avviene solamente quando il segnale di enable è positivo
- Gli ingressi di **Preset** e **Clear** devono essere tenuti alti durante il funzionamento. Possono essere usati per definire lo stato iniziale del FF quando il segnale EN è basso (EN=0)

Flip Flop D

D Flip Flop

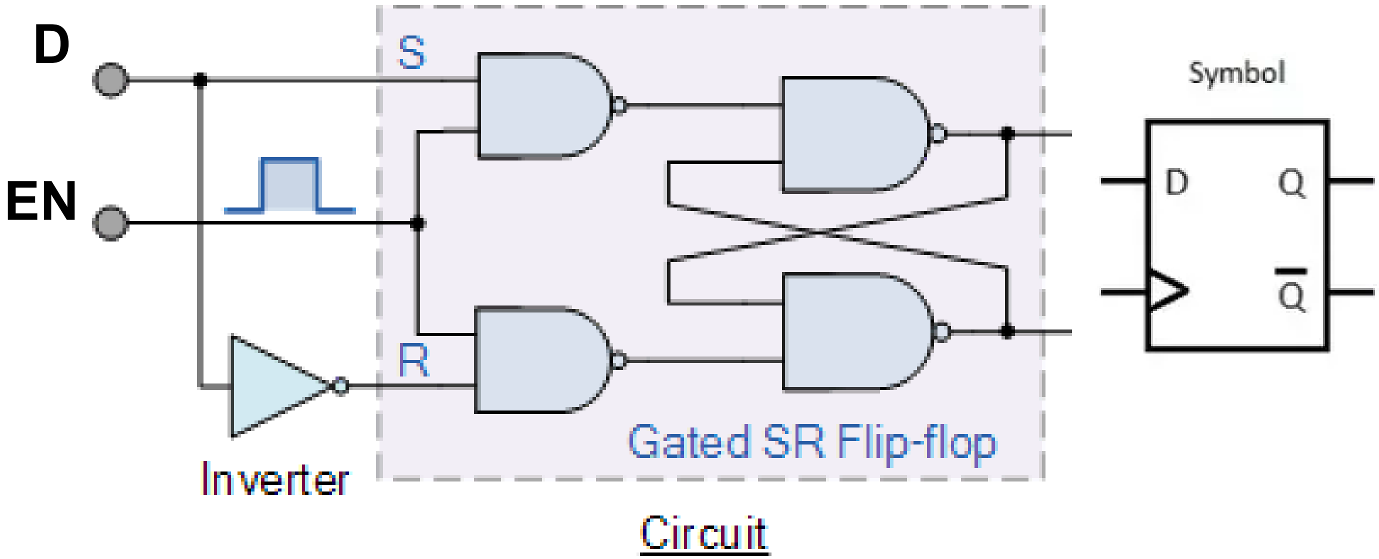


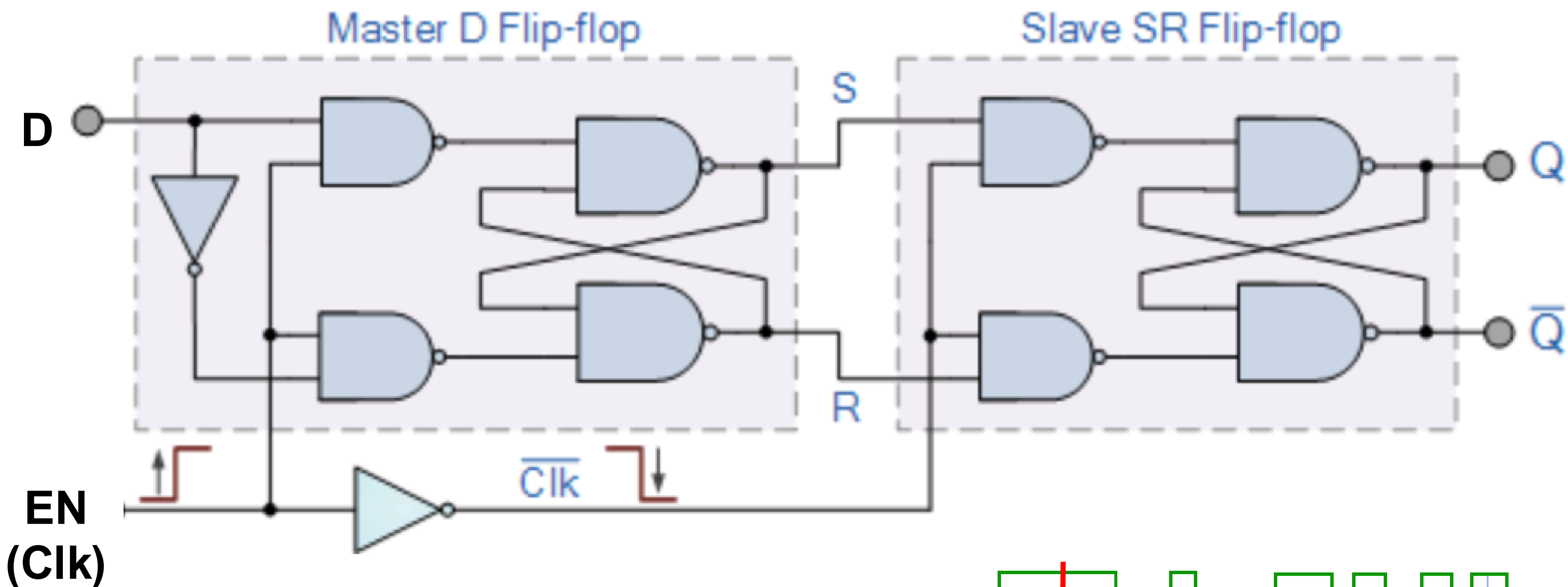
Table of truth:

clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

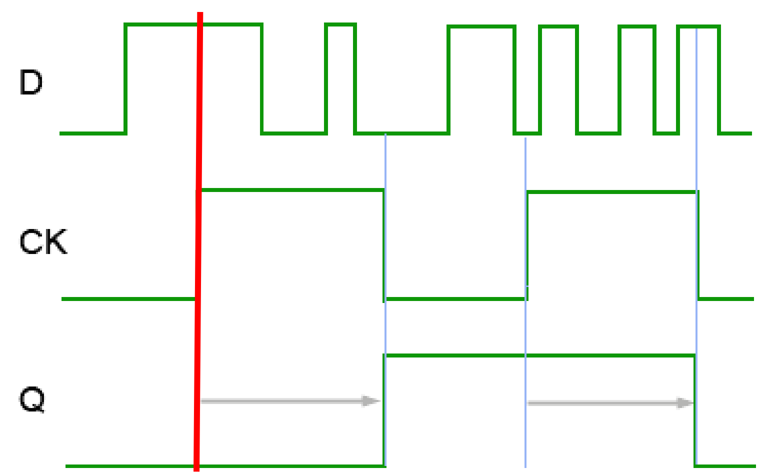
- Grazie all’invertitore, si ha solamente S=1, R=0 oppure S=0, R=1 -> assimilabile a un unico input “D”
- Il DATO (D) viene trasferito su Q solo se il segnale di enable è alto
    - Latch FF, **Level Triggered**

Master Slave Flip Flop D

MASTER-SLAVE D Flip Flop



- Clk=1 → S e R sono settati da D.
- Clk=0 → Q e !Q sono settati da S e R.
- D è trasferito a Q in un intero ciclo di clock
- **Edge Triggered**

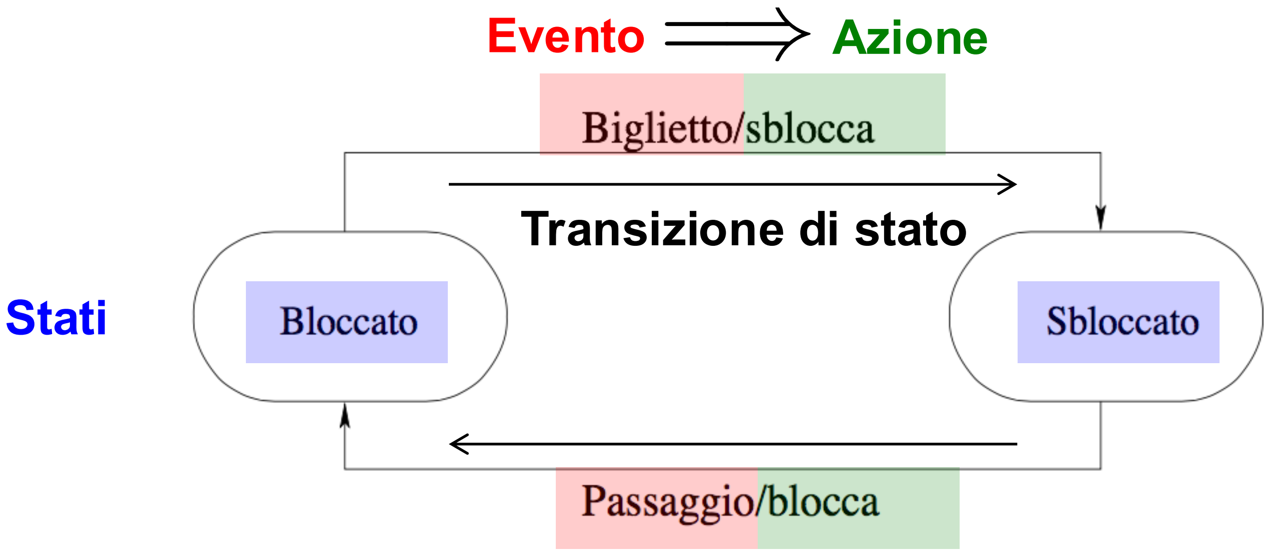


Backup

Sistemi Logici Complessi

porte logiche + flip flop + memorie/registri --> Sistemi logici complessi

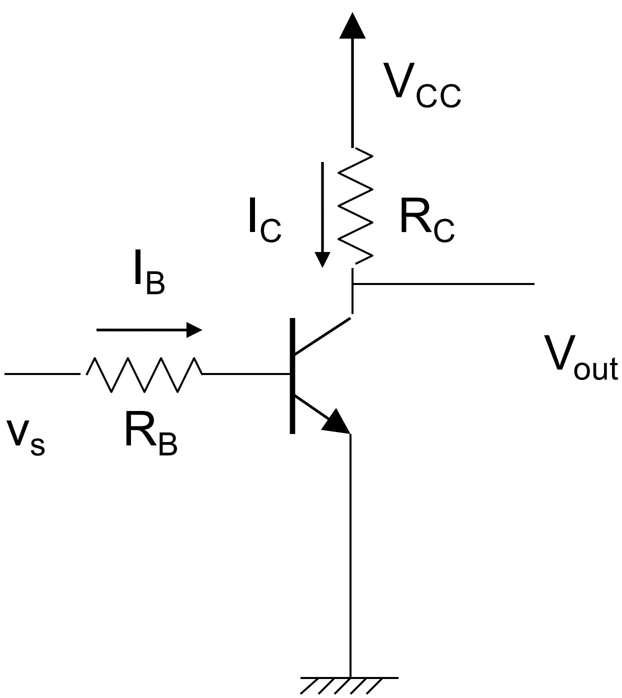
FSM (Finite State Machine): sistema che può trovarsi in un numero finito di **stati** che può cambiare mediante **transizioni** triggerate da **eventi** esterni



Invertitore (NOT)

Realizzazione: è di fatto un interruttore

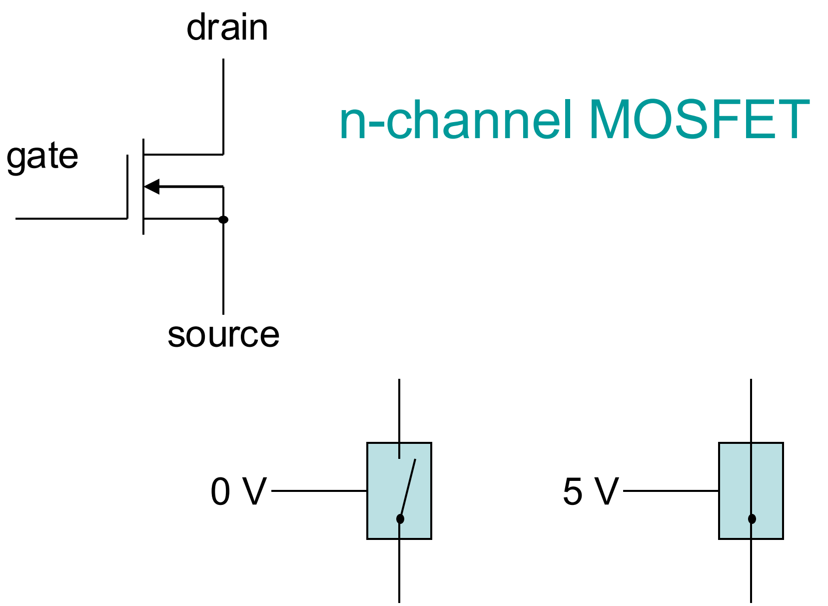
logica TTL (BJT)

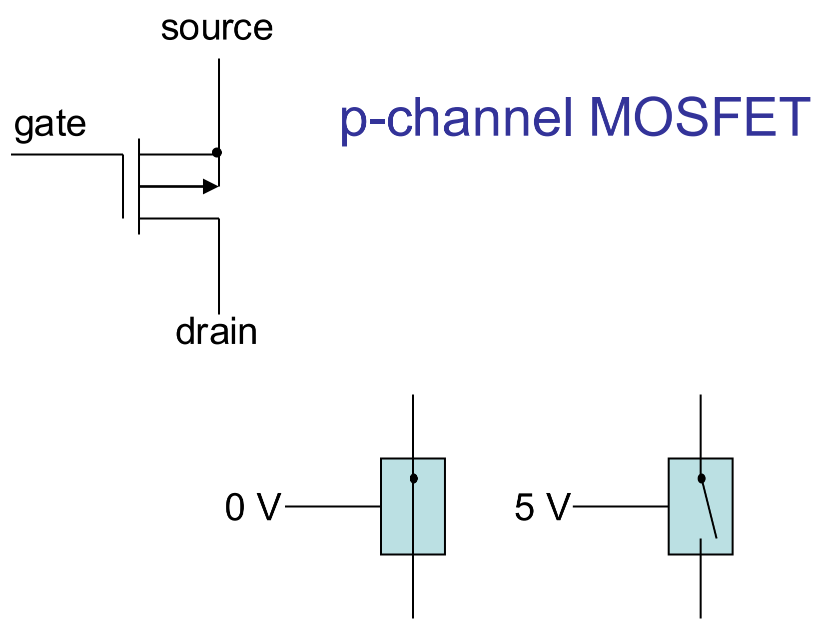


- quando V_s è ~ 0 il transitor è in cut-off
  → I_B~0
  → I_C~0
  → V_{out} è "pulled up" verso V_{CC}
- quando V_s è "grande" il transitor va in saturazione
  → I_C è massima
  → V_{out}~0
  (dato che $V_{CC}-V_{out} = R_C * I_C$)

Interruttori MOSFET

- i MOSFET, utilizzati nei circuiti di logica, agiscono come interruttori controllati con un voltaggio
  - n-channel MOSFET è chiuso (conduce) quando è applicato un voltaggio positivo (+5V), aperto quando il voltaggio è nullo
  - p-channel MOSFET è aperto quando è applicato un voltaggio positivo (+5V), chiuso (conduce) quando il voltaggio è nullo

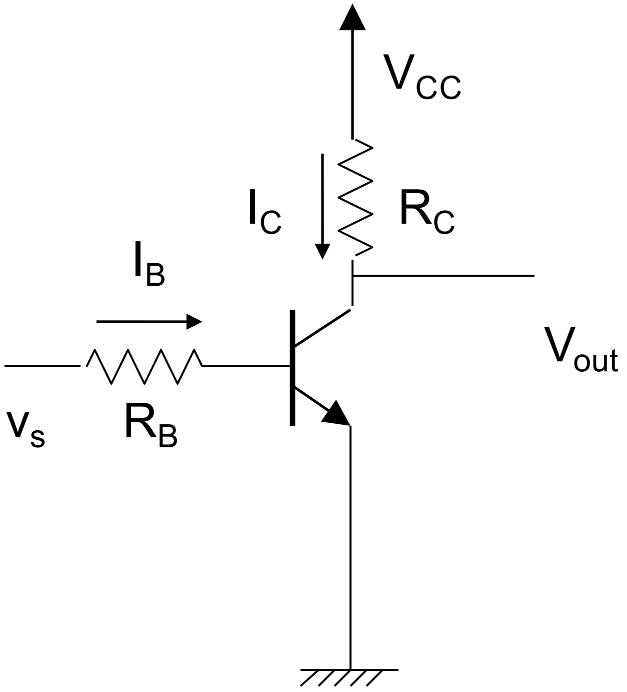




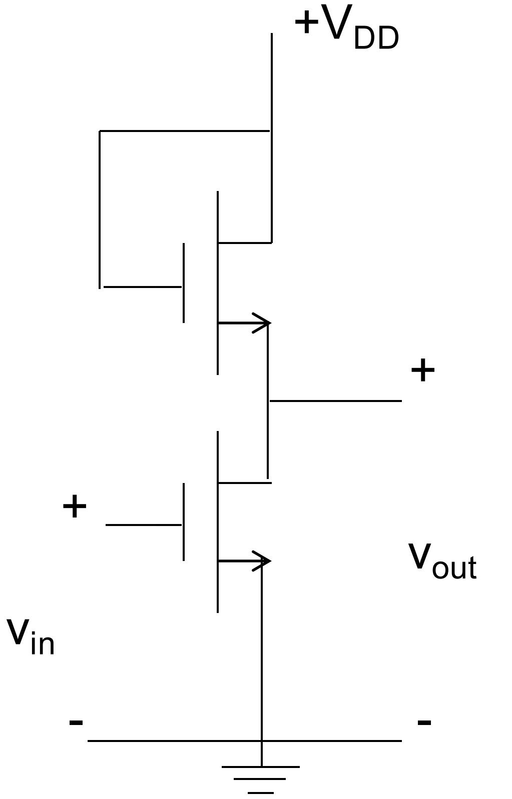
Invertitore (NOT)

Realizzazione: è di fatto un interruttore

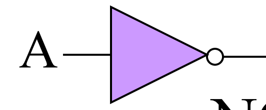
logica TTL (BJT)



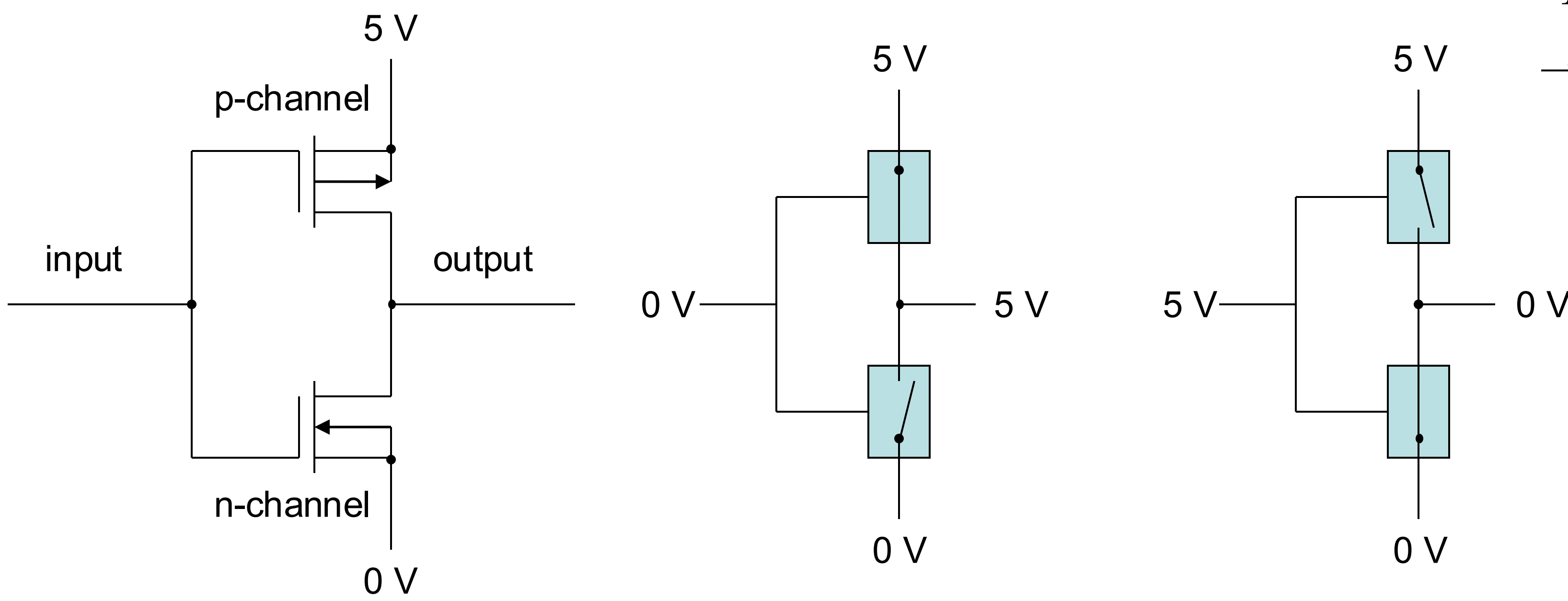
logica NMOS (MOSFET)



Invertitore (NOT) MOSFET:



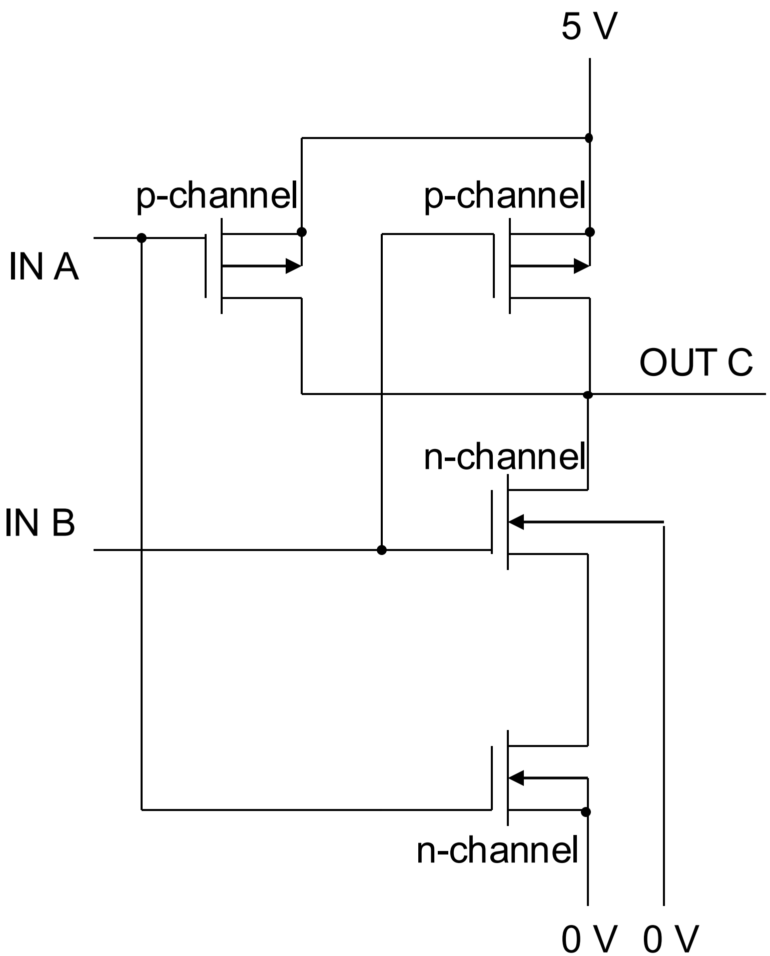
A	C
0	1
1	0



- 0V come input “apre” il FET in basso (n-channel) ma “chiude” quello in alto (p-channel) → l’output è a +5V
- 5V come input “chiude”” il FET in basso (n-channel) ma “apre” quello in alto (p-channel) → l’output è a 0V

→ l’effetto netto è l’inversione logica: 0 $\rightarrow$ 5; 5 $\rightarrow$ 0

NAND MOSFET:



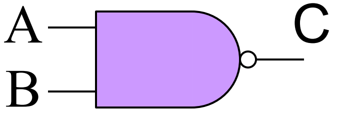
- Entrambe gli input a 0V:
  - i due FET in basso OFF, i due in alto ON

  → uscita "alta"
- Entrambe gli input a 5V:
  - i due FET in basso ON, i due in alto OFF

  → uscita "bassa"
- IN A a 5V, IN B a 0V:
  - alto a sinistra OFF, più basso ON
  - alto a destra ON, in mezzo OFF

  → uscita "alta"
- IN A a 0V, IN B a 5V:
  - opposto rispetto a prima

  → uscita "alta"

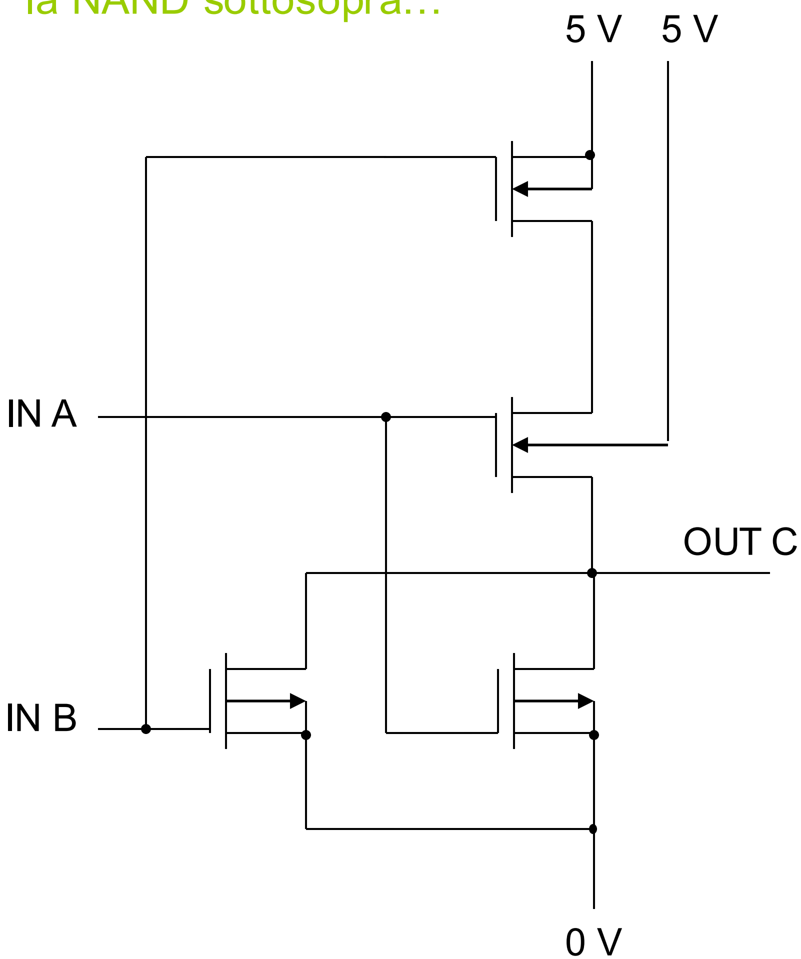


NAND

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR MOSFET:

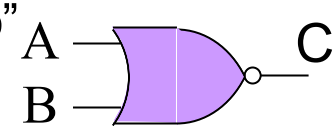
la NAND sottosopra…



- Entrambe gli input a 0V:
  – i due FET in basso OFF, i due in alto ON
  → output "alto"

→ Entrambe gli input a 5V:
  – i due FET in basso ON, i due in alto OFF
  → output "basso"

- IN A a 5V, IN B a 0V:
  – basso a sinistra OFF, basso destra ON
  – più alto ON, in mezzo OFF
  → output "basso"
- IN A a 0V, IN B a 5V:
  – opposto rispetto a prima
  → output "basso"



NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Sottofamiglie TTL