

# Reti combinatorie

## **Argomenti:**

- Elementi base: porte logiche, registri, bus
- Analisi e sintesi di reti combinatorie

## **Materiale di studio:**

- Capitoli 11.1 e 11.2
- Capitoli 11.3, 11.5 (no sezioni "The QuineMccluskey method", "synchronous counters", "Field-Programmable Gate Array")

# Elementi base

# Valori logici: convenzione

I **valori logici** sono 2.

Per indicarli useremo i nomi:

1  
VERO  
ALTO  
+5 V

0  
FALSO  
BASSO  
0 V

I nomi in ciascuna colonna sono equivalenti.

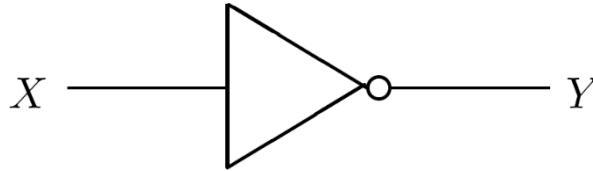
I nomi sono solo una convenzione!

# Definizione di porta logica

Una **porta logica** è un dispositivo con  $N$  ingressi ed 1 uscita, che realizza un legame tra il valore presente all'uscita e quelli presenti agli ingressi, esprimibile con una funzione logica elementare

# Porta NOT

Primo esempio di porta logica: la **porta NOT**

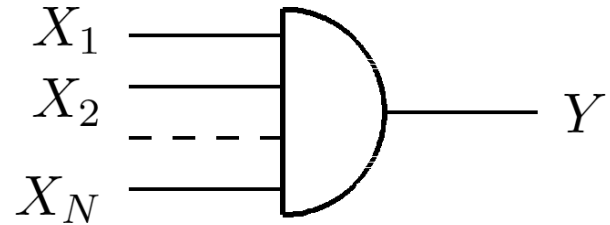


Produce in uscita un valore logico opposto a quello presente all'ingresso.

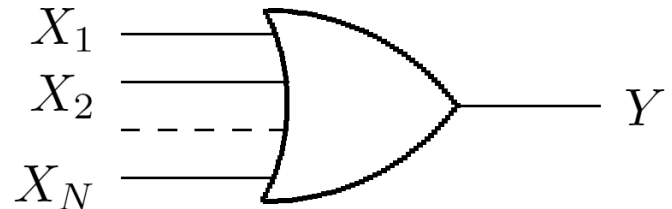
- $N = 1$  ingresso

# Porte logiche AND e OR

La porta **AND** fa assumere all'uscita il valore logico 1 se e solo se tutti gli ingressi si trovano ad avere il valore 1

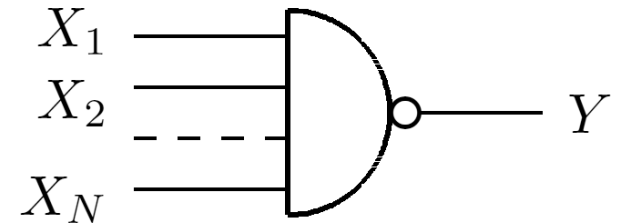


La porta **OR** fa assumere all'uscita il valore logico 1 se e solo se ad almeno uno degli ingressi è presente un valore logico 1

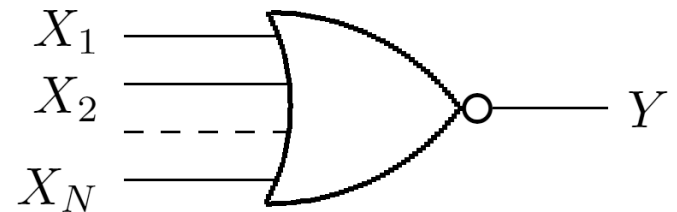


# Porte logiche NAND e NOR

La porta **NAND** fa assumere all'uscita il valore logico 0 se e solo se tutti gli ingressi si trovano ad avere il valore 1



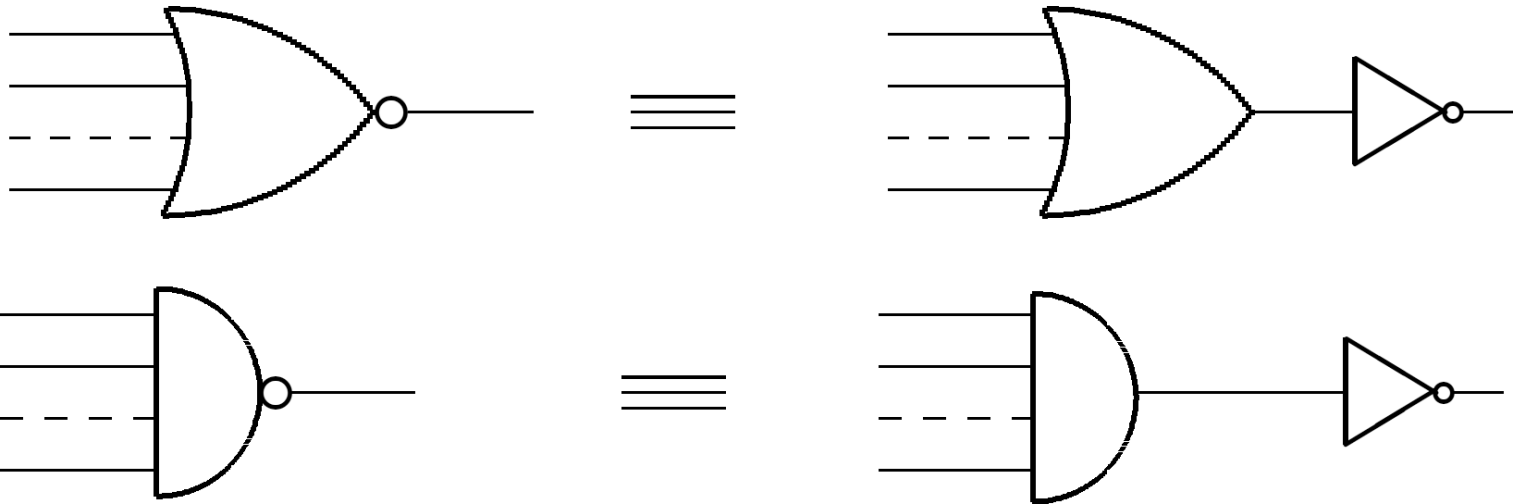
La porta **NOR** fa assumere all'uscita il valore logico 0 se e solo se ad almeno uno degli ingressi è presente un valore logico 1



# Equivalenze tra porte logiche (1 di 3)

La funzione realizzata da una porta logica può essere ottenuta mediante opportune sequenze di altre porte logiche:

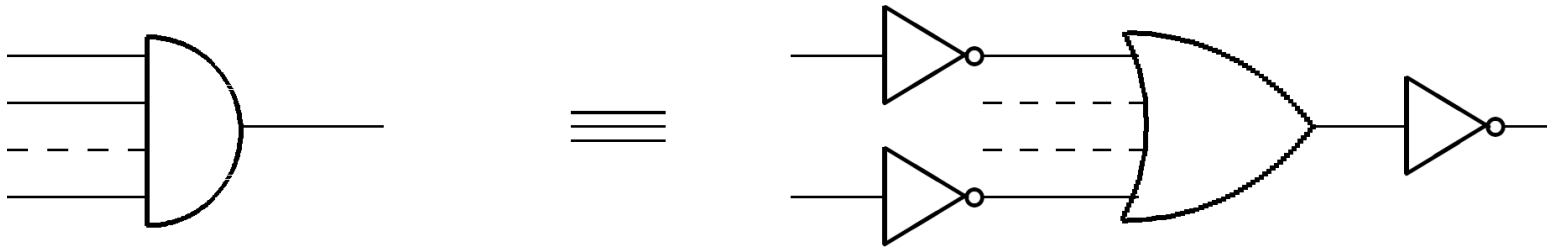
- Le porte NOR/NAND possono essere ottenute mediante una porta OR/AND e una NOT collegate in sequenza



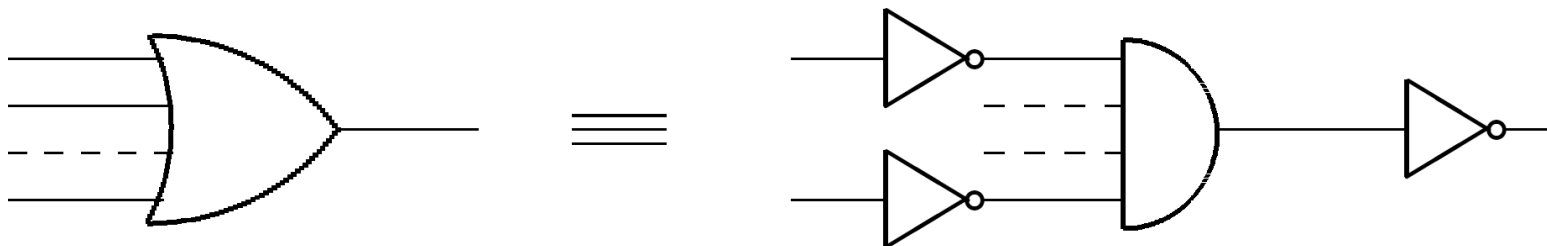


# Equivalenze tra porte logiche (2 di 3)

**Ogni funzione logica** può essere ottenuta impiegando solo porte **OR e NOT**

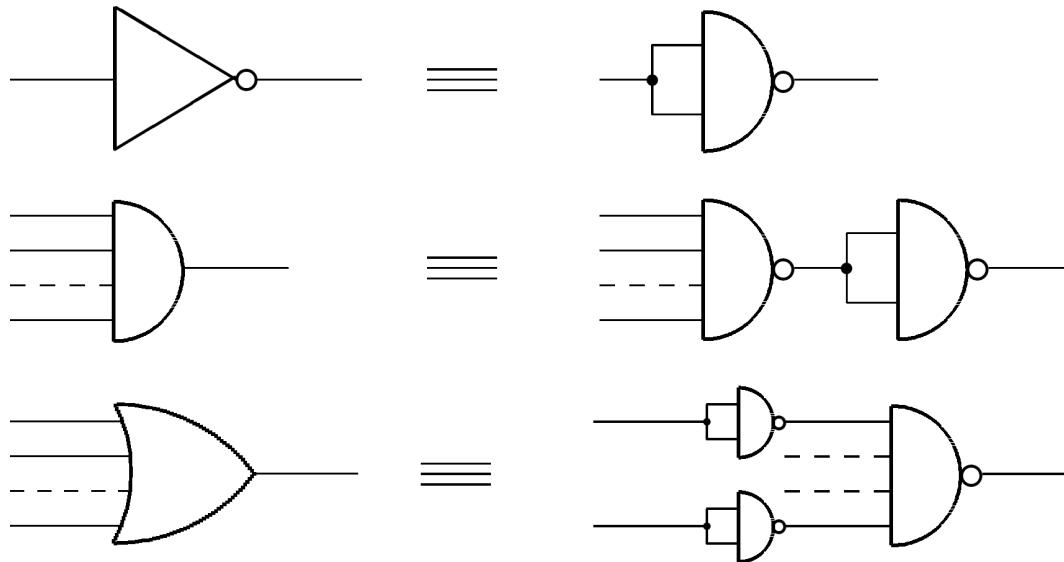


**Ogni funzione logica** può essere ottenuta impiegando solo porte **AND e NOT**



# Equivalenze tra porte logiche (3 di 3)

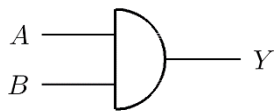
**Ogni funzione logica** può essere ottenuta impiegando solo porte **NAND**



**Ogni funzione logica** può essere ottenuta impiegando solo porte **NOR** (provarlo per esercizio)

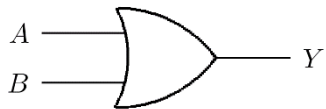
# Tabelle di verità

Un secondo modo di rappresentare una porta logica è mediante la sua **tabella di verità**, che specifica il valore dell'uscita per ciascuna possibile combinazione dei valori in ingresso.



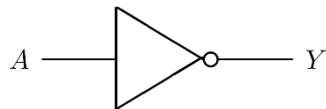
AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



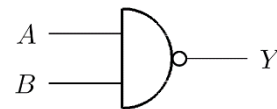
OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



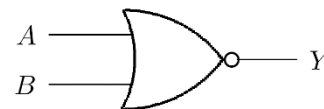
NOT

—	A	Y
	0	1
	1	0



NAND

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



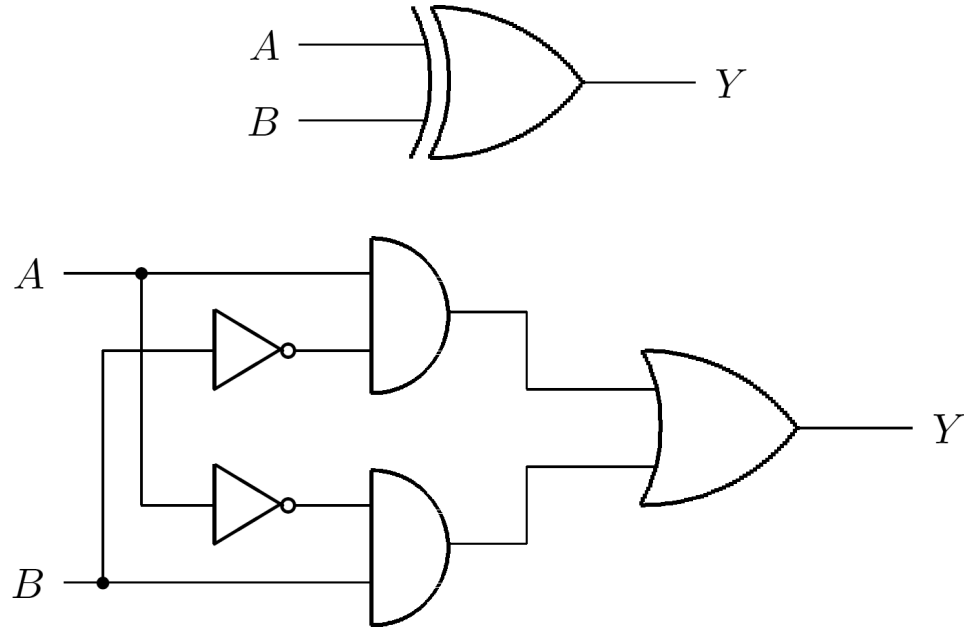
NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0


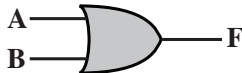
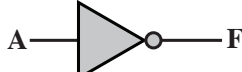



# OR esclusivo

Un esempio più complesso è dato dalla porta logica che realizza la funzione di **OR esclusivo**: l'uscita Y assume il valore 1 se e solo se ai 2 ingressi sono presenti valori logici diversi

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

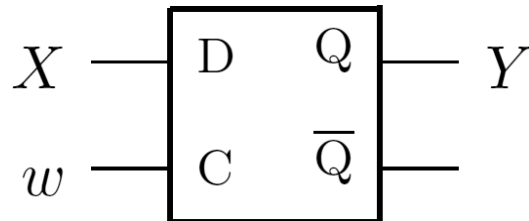


# Notazione algebrica

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

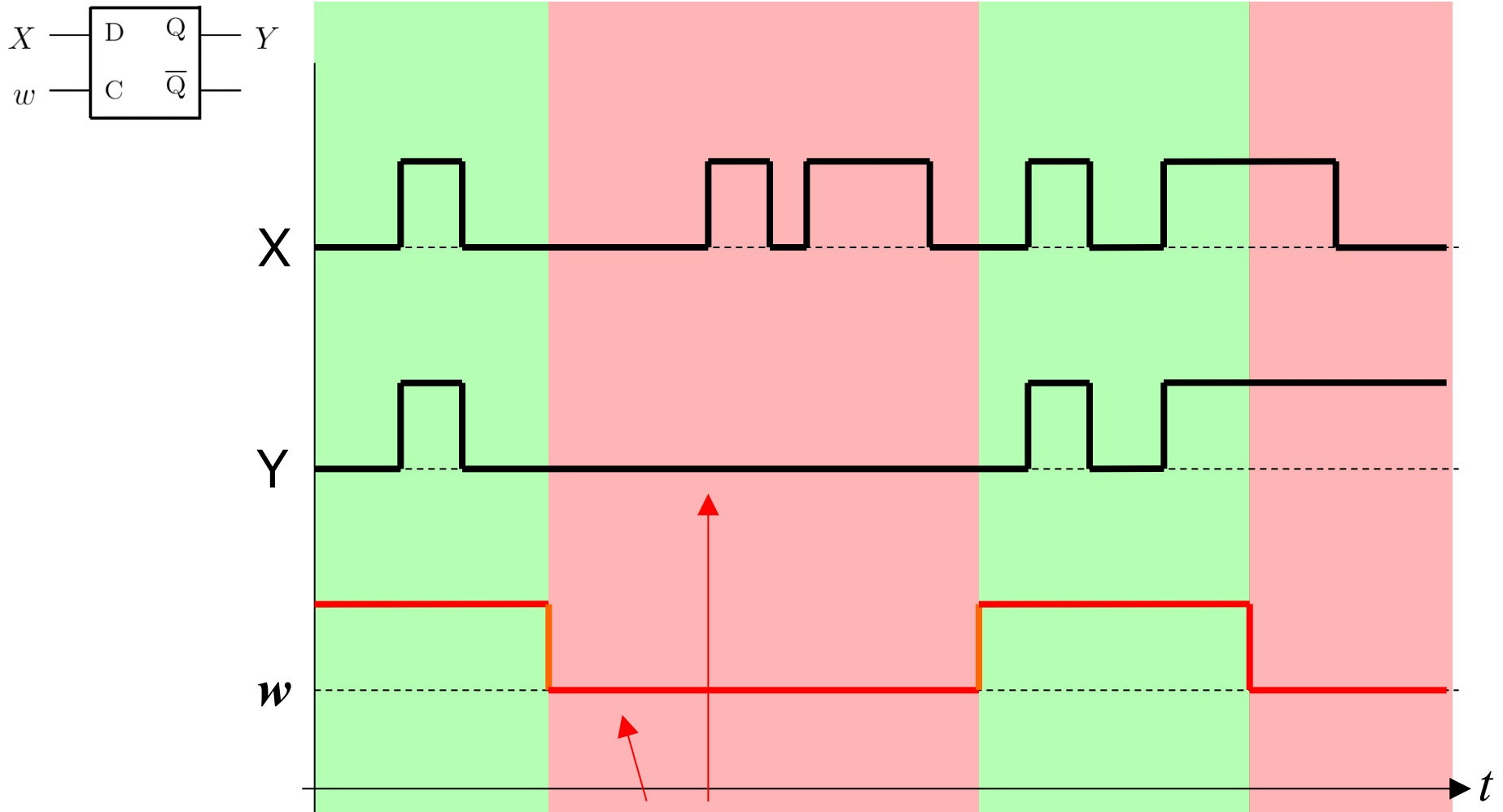
# Registro da un bit

- Un **registro** è un dispositivo in grado di memorizzare (cioè conservare nel tempo) un valore logico.
  - Tale capacità distingue i registri dalle porte logiche.
  - E' un circuito sequenziale!



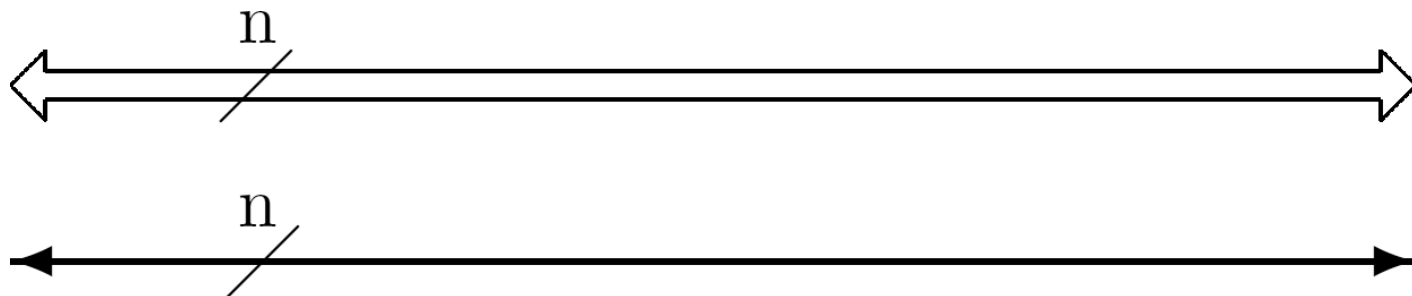
- $w$  è il **segnale di controllo**
  - $w=1$ : il valore di  $X$  viene trasferito (memorizzato) nel registro;  $Y=X$
  - $w=0$ :  $Y$  rimane al valore memorizzato, senza risentire di eventuali variazioni di  $X$

# Registro da un bit: diagramma temporale



# Bus

- Un **bus** è un collegamento elettrico tra parti diverse di un elaboratore che consente il trasferimento di informazione.
- Un bus è composto da una o più linee; ogni linea consente di trasferire un bit.
- Per rappresentare un bus di  $n$  bit si usano i seguenti simboli grafici:





# Gruppi di Bit

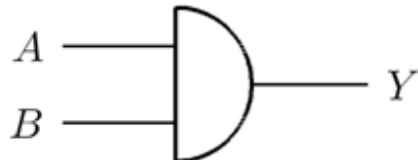
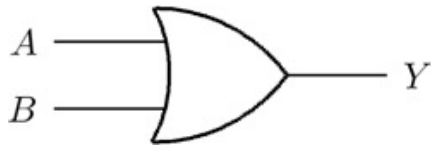
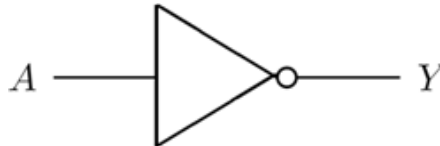
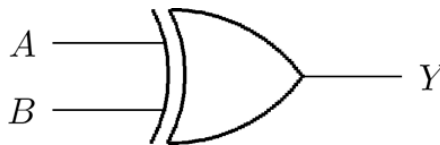
Un po' di nomenclatura...

- 4 bit: "**nibble**" (o nybble)
- 8 bit: "**byte**"
- 16 bit: "**half-word**"
- 32 bit: "**word**"
- 64 bit: "**double-word**"

La definizione di "word" dipende comunque dalla taglia della parola di memoria della macchina che si sta considerando.

# Reti Logiche Combinatorie

# Porte logiche: notazione algebrica

Nome	Simbolo grafico	Tabella di verità	Notazione algebrica															
AND		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	$Y = A \cdot B$
A	B	Y																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$Y = A + B$
A	B	Y																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		<table><tr><th>-</th><th>A</th><th>Y</th></tr><tr><td></td><td>0</td><td>1</td></tr><tr><td></td><td>1</td><td>0</td></tr></table>	-	A	Y		0	1		1	0	$Y = \overline{A}$						
-	A	Y																
	0	1																
	1	0																
XOR		<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$Y = A \oplus B$
A	B	Y																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

# Rete logica combinatoria: definizione

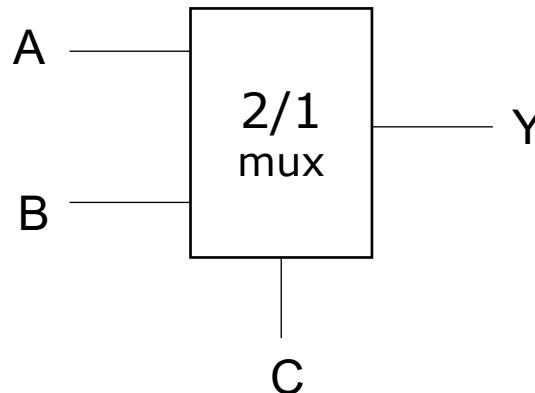
**Rete logica combinatoria:** è una rete logica nella quale, in ogni istante, i valori presenti alle uscite sono determinati unicamente dai valori presenti agli ingressi nel medesimo istante.

Una rete logica combinatoria è quindi:

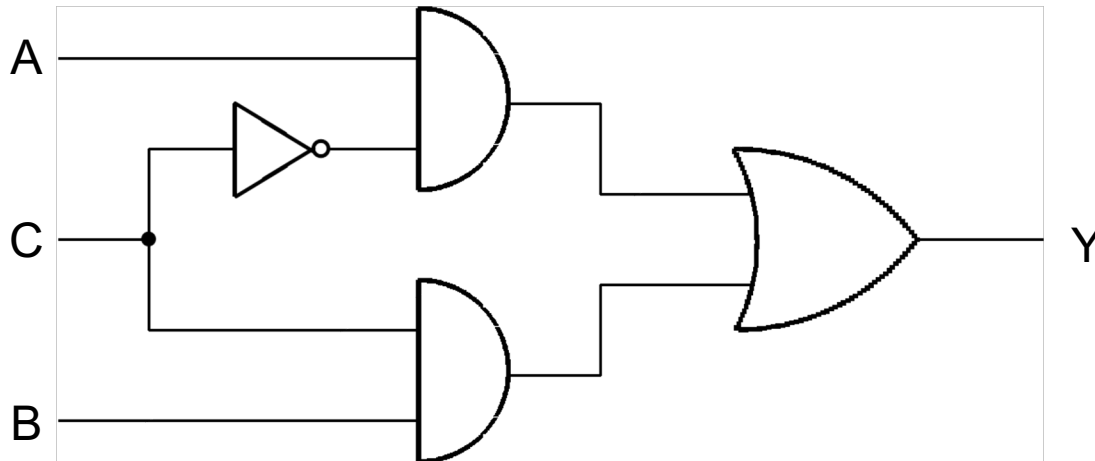
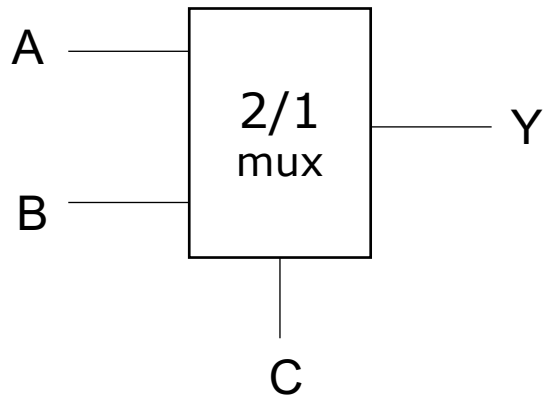
- **priva di stato:** non contiene elementi di memoria;
- **interamente descritta dalla sua tabella di verità**, che definisce, in altrettante colonne, le funzioni logiche delle variabili di ingresso prodotte alle uscite.

## Esempio: Multiplexer $n/1$

- Il **multiplexer  $n/1$**  è una rete combinatoria con ( $n$  potenza di 2) :
  - $n + \log_2 n$  bit di input
  - 1 bit di output
- I  $\log_2 n$  codificano quale delle  $n$  linee di input propagare verso l'unica linea di output.



# Esempio: Multiplexer 2/1



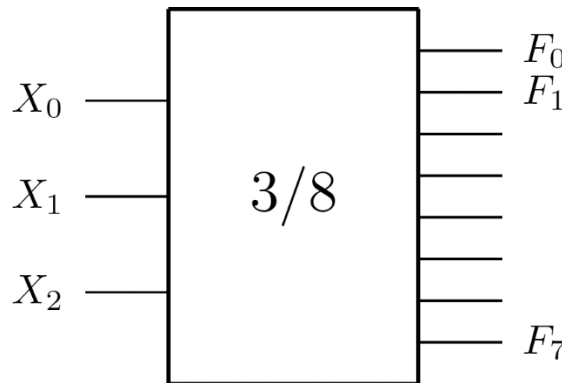
<b>A</b>	<b>B</b>	<b>C</b>	<b>Y</b>
0	0	0	<b>0</b>
0	1	0	<b>0</b>
1	0	0	<b>1</b>
1	1	0	<b>1</b>
0	0	1	<b>0</b>
0	1	1	<b>1</b>
1	0	1	<b>0</b>
1	1	1	<b>1</b>

$Y = A$  se  $C = 0$

$Y = B$  se  $C = 1$

## Esempio 2: il decodificatore $n/2^n$

- Un **decodificatore**  $n/2^n$  è una rete combinatoria con:
  - $n$  bit di input
  - $2^n$  bit di output
- Il decodificatore attiva l' $i$ -esima uscita se e solo se il valore binario codificato dagli ingressi è  $i$ .



## Esempio 2: il decodificatore 3/8

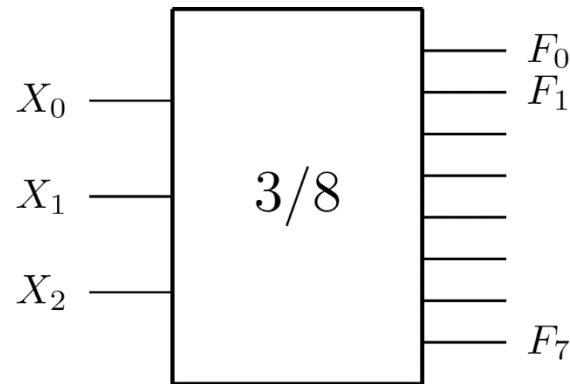
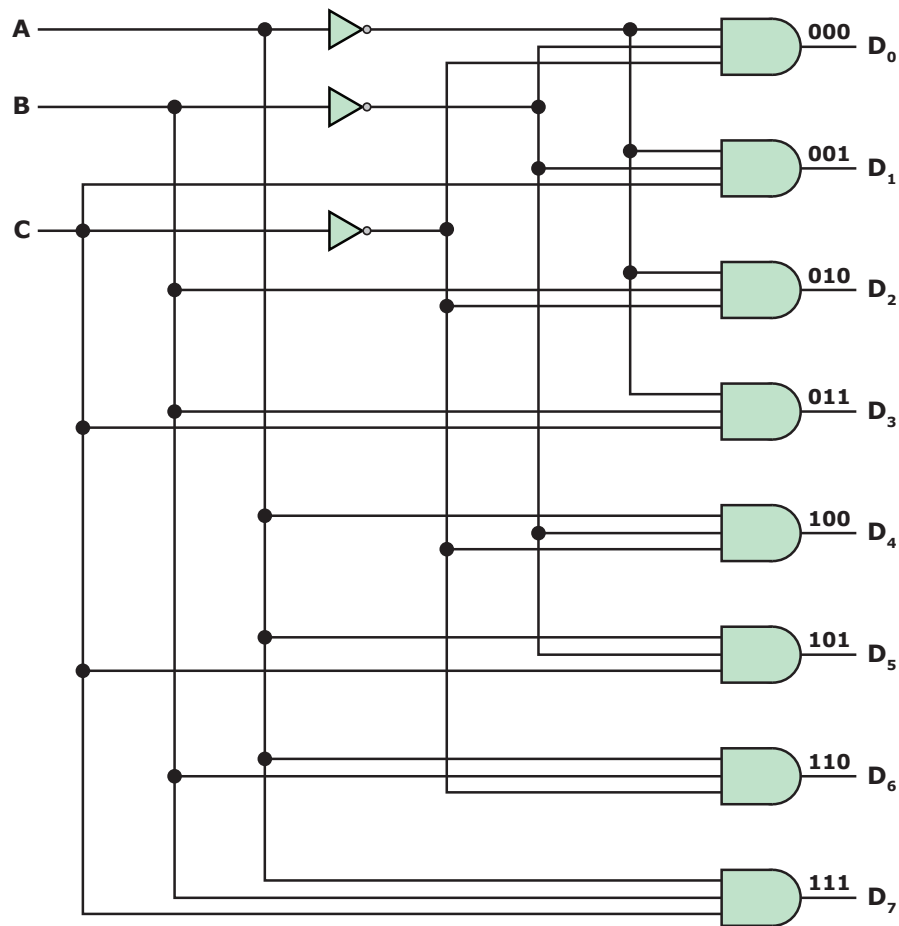


Tabella di verità per un  
decodificatore con 3  
ingressi e  $2^3=8$  uscite:

$X_2$	$X_1$	$X_0$	$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



# Decodificatore: realizzazione



# Funzioni logiche n/1

- Il numero delle diverse possibili funzioni logiche di  $n$  input e 1 output è un **numero finito**.
- Le funzioni logiche di  $n = 2$  input sono tante quante sono le diverse possibili tabelle di verità che le definiscono, cioè 16
  - Ci sono  $2^2 = 4$  possibili input e ogni input può assumere 2 valori:  $2^4 = 16$ .
- Le funzioni logiche di  $n = 3$  input sono  $2^8 = 256$ .

# Funzioni logiche (n/m)

- Le funzioni logiche di ***n*** variabili di input e una variabile di output sono  $2^{2^n}$ .
- Le funzioni logiche di ***n*** variabili di input e ***m*** variabili di output sono  $2^m 2^n$ .

# Rappresentazioni di reti logiche

Le reti logiche possono essere rappresentate in tre modi **equivalenti**:

- **schema grafico**
  - **tabella di verità**
  - **espressione algebrica**
- 
- Scegliremo in ciascun caso la rappresentazione più opportuna

# Somma di prodotti

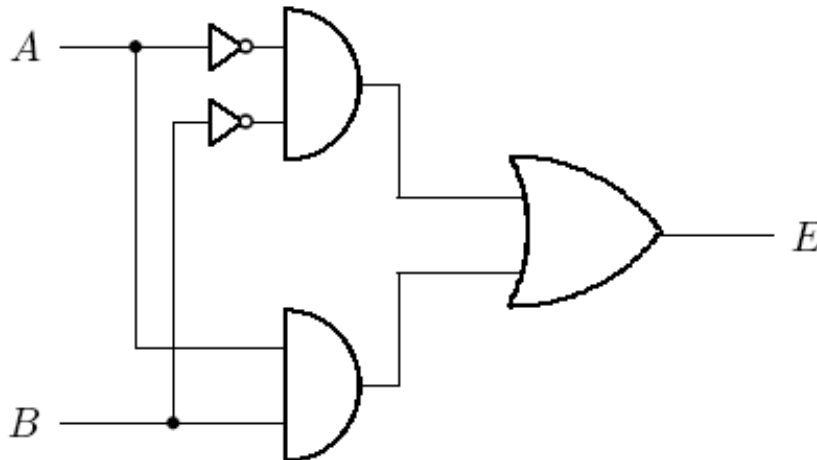


A	B	E
0	0	1
0	1	0
1	0	0
1	1	1

E=1 se A=0 e B=0 ( $\bar{A} \cdot \bar{B} = 1$ )

E=1 se A=1 e B=1 ( $A \cdot B = 1$ )

È possibile ottenere E come "**somma di prodotti**":

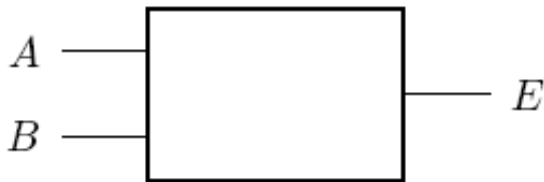


$$E = \bar{A} \cdot \bar{B} + A \cdot B$$

La funzione che vale 1 quando è uguale a 1 uno o l'altro dei suoi due ingressi e l'OR

# Prodotto di somme

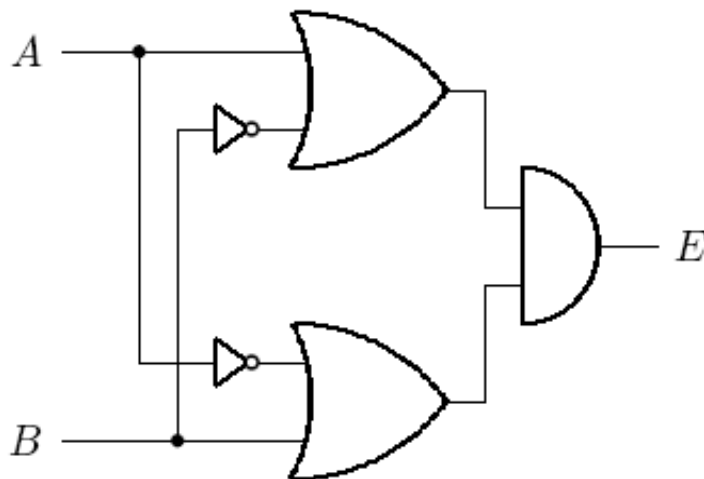
E si può ottenere anche come “**prodotto di somme**”:



<i>A</i>	<i>B</i>	<i>E</i>
0	0	1
0	1	0
1	0	0
1	1	1

E=0 se A=0 e B=1 ( $A+\bar{B}=0$ )

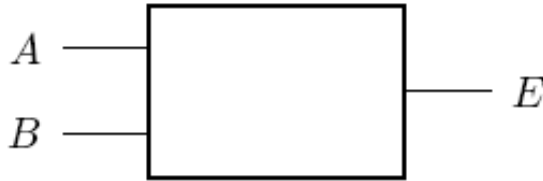
E=0 se A=1 e B=0 ( $\bar{A}+B=0$ )



$$E = (A + \bar{B}) \cdot (\bar{A} + B)$$

La funzione che vale 0 quando è uguale a 0 uno o l'altro dei suoi due ingressi e l'AND

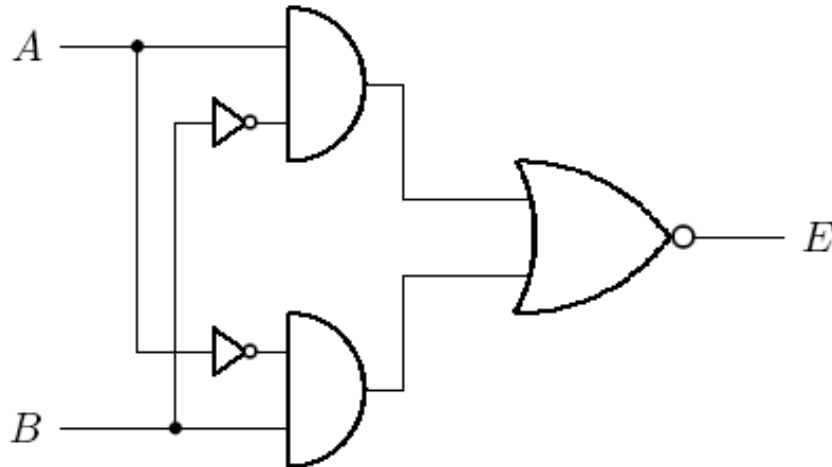
# Equivalenza tra circuiti



A	B	E
0	0	1
0	1	0
1	0	0
1	1	1

$$E = \overline{A \oplus B}$$

Diversi circuiti logici **equivalenti** realizzano la stessa funzione logica:



(E = OR esclusivo negato)

# Algebra di Boole o booleana

L'analisi delle proprietà delle espressioni algebriche costruite da variabili binarie e operatori logici, si deve al matematico G. Boole (1815-1864), ed è nota come **algebra booleana**.

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



# Proprietà base dell'algebra booleana

## Basic Postulates

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

$$A + B = B + A$$

Commutative Laws

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

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

Distributive Laws

$$1 \cdot A = A$$

$$0 + A = A$$

Identity Elements

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

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

Inverse Elements

## Other Identities

$$0 \cdot A = 0$$

$$1 + A = 1$$

$$A \cdot A = A$$

$$A + A = A$$

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

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

Associative Laws

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

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

DeMorgan's Theorem

# Legge di De Morgan

Legge di De Morgan:

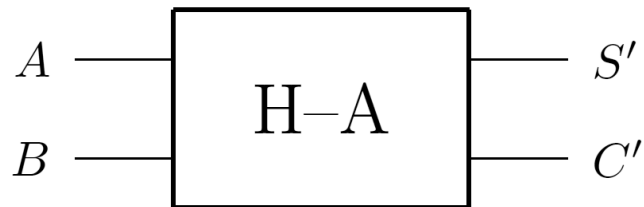
$$\overline{A + B} = \overline{A} \cdot \overline{B} \quad \text{oppure} \quad \overline{\overline{A} + \overline{B}} = A \cdot B$$

$$\overline{A \cdot B} = \overline{A} + \overline{B} \quad \text{oppure} \quad \overline{\overline{A} \cdot \overline{B}} = A + B$$

ES:



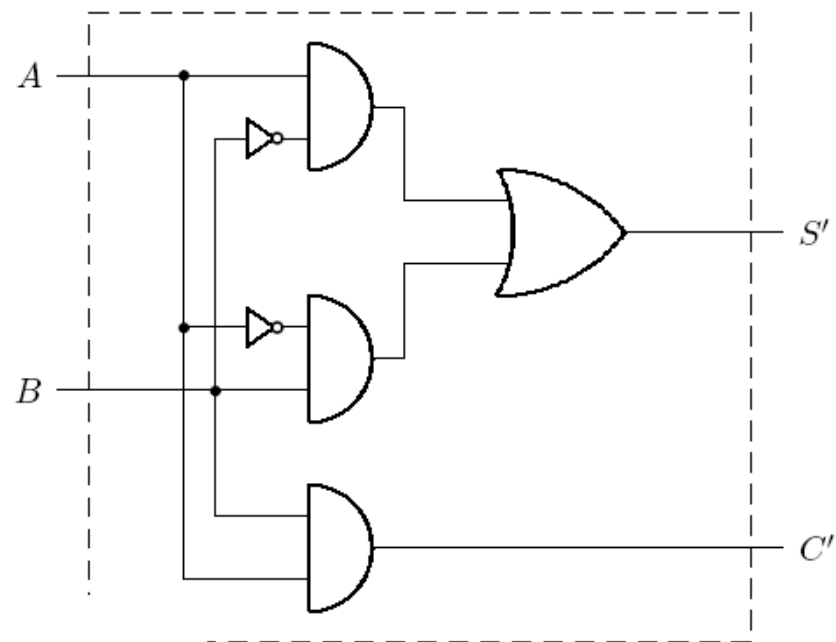
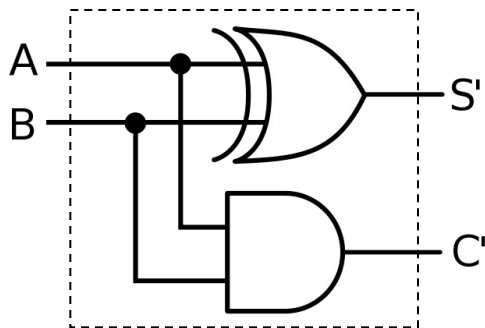
# Sintesi di un half-adder



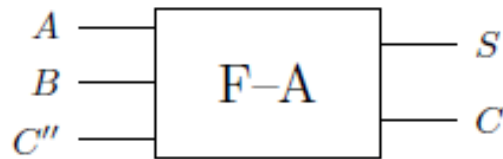
$A$	$B$	$S'$	$C'$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$C' = A \cdot B$$



# Sintesi di un full-adder (1 di 2)



A	B	C''	S	C
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

$$S = \overline{A} \cdot \overline{B} \cdot C'' + \overline{A} \cdot B \cdot \overline{C}'' + A \cdot \overline{B} \cdot \overline{C}'' + A \cdot B \cdot C''$$

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

$$S = (A \oplus B) \cdot \overline{C}'' + (\overline{A \oplus B}) \cdot C'' = (A \oplus B) \oplus C''$$

$$S = S' \oplus C''$$

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

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

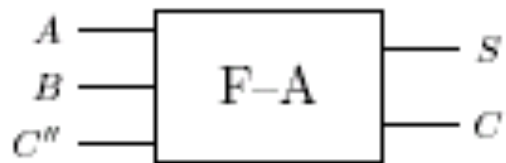
$$C = S' \cdot C'' + C'$$

Half-Adder

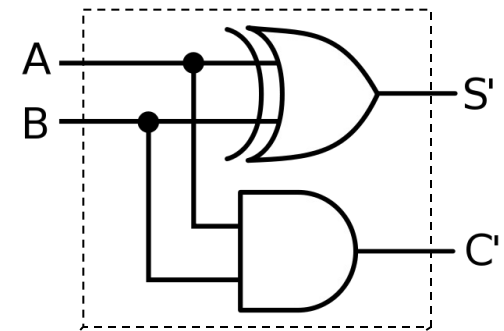
$$S' = (A \oplus B)$$

$$C' = A \cdot B$$

# Sintesi di un full-adder (2 di 2)

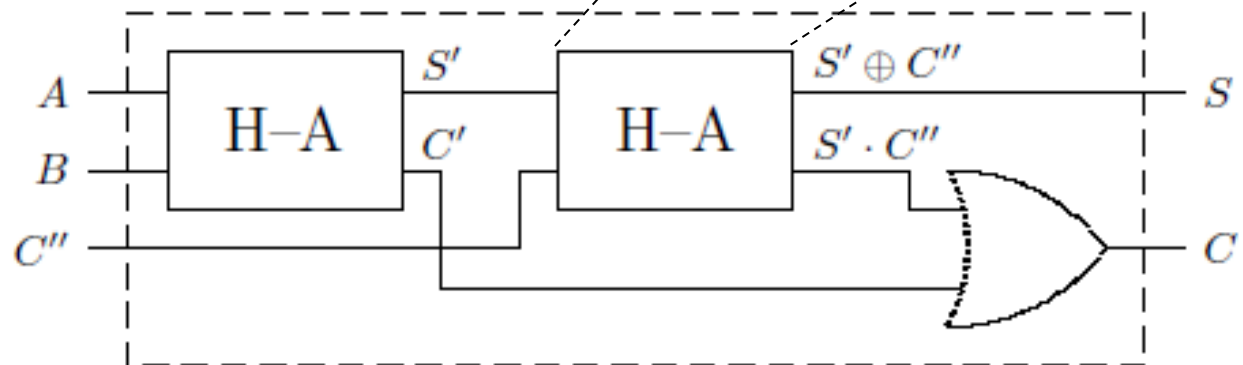


A	B	C''	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$S = S' \oplus C''$$

$$C = S' \cdot C'' + C'$$



Schema di un *full-adder*.

# Sintesi di un half-adder con porte NAND

Utilizziamo l'algebra booleana e le sue proprietà per riscrivere  $S'$  utilizzando solo porte NAND:

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

$$S' = \overline{\overline{\bar{A} \cdot B} \cdot \overline{A \cdot \bar{B}}}$$

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

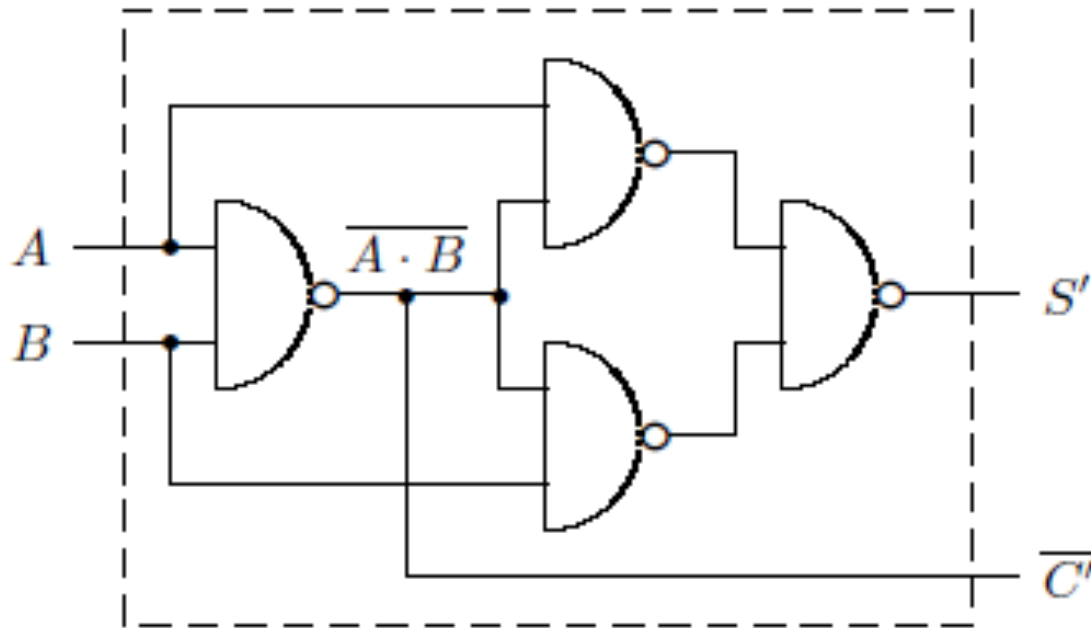
$$S' = \overline{B \cdot (\overline{\bar{A} + \bar{B}}) \cdot A \cdot (\overline{\bar{A} + \bar{B}})}$$

$$S' = \overline{B \cdot (\overline{A \cdot B}) \cdot A \cdot (\overline{A \cdot B})}$$

# Half-adder con sole porte NAND

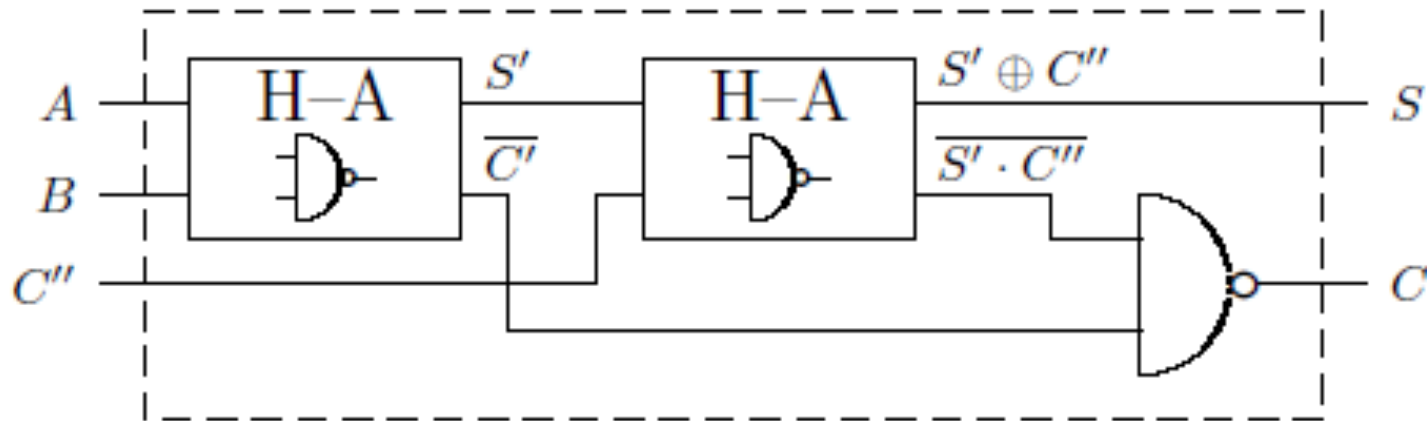
$$S' = \overline{\overline{B \cdot (\overline{A \cdot B})} \cdot \overline{A \cdot (\overline{A \cdot B})}}$$

$$C' = A \cdot B$$



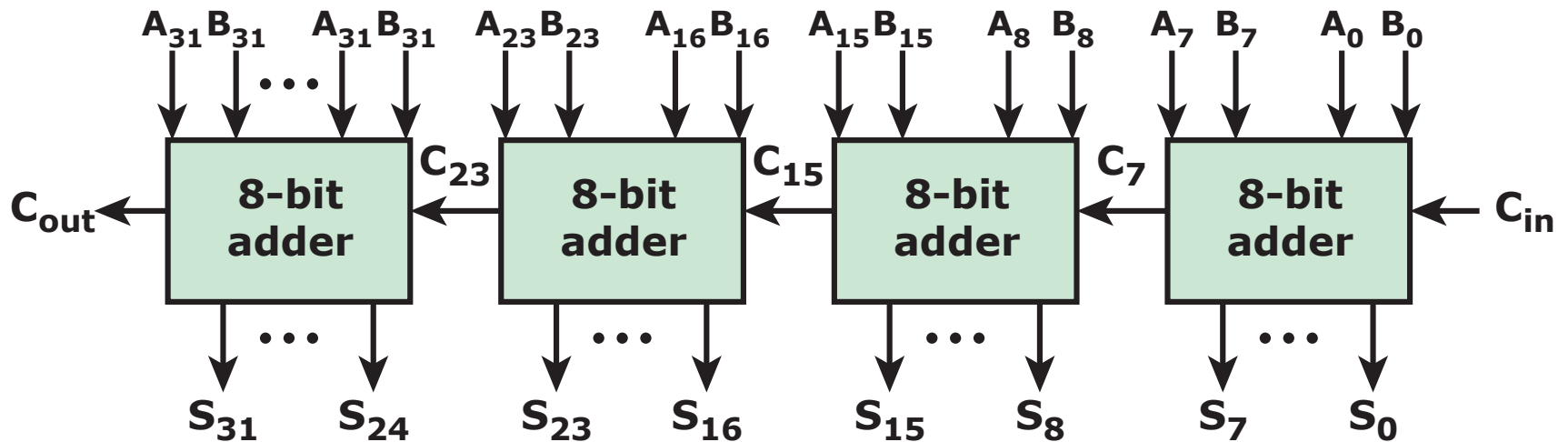
# Full-adder con sole porte NAND

$$C = S' \cdot C'' + C' = \overline{\overline{S'} \cdot \overline{C''}} \cdot \overline{C'}$$





# Sommatore binario da 32 bit



# Minimizzazione con Mappe di Karnaugh

Le seguenti proprietà dell'algebra di Boole consentono di semplificare notevolmente le espressioni booleane:

$$A \bullet B + A \bullet \bar{B} = A \bullet (B + \bar{B}) = A$$

$$A \bullet (B \bullet C + B \bullet \bar{C} + \bar{B} \bullet C + \bar{B} \bullet \bar{C}) = A$$

Le mappe di Karnaugh sono una particolare forma di tabella di verità, che consente di individuare immediatamente la possibilità di fare queste semplificazioni.

# Mappe di Karnaugh (1)

Ad esempio, la seguente tabella di verità della funzione  $Y=Y(A,B,C)$

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

può essere ridisegnata così:

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

Mappa di Karnaugh della funzione  $Y$

Nelle mappe di K. i valori della funzione sono scritti dentro le caselle.

# Mappe di Karnaugh (2)

<b>A \ B</b>	0		1		
	0	1	1	0	
<b>C</b>	0	0	0	1	0
	1	0	1	1	1

Dalla tabella di verità o dalla mappa di Karnaugh è immediato ottenere l'espressione booleana della funzione **Y** come "somma di prodotti", cioè come OR di tanti termini AND quante sono le caselle in cui la funzione vale 1; ciascuno di questi termini AND (detti *minterm*) è costituito dall'AND delle variabili di ingresso, negate oppure no a seconda che il valore della variabile associato a quella casella sia 0 oppure 1:

$$Y = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

# Mappe di Karnaugh (3)

Nel caso di funzioni di 4 variabili, ad es.  $Z=Z(A,B,C,D)$ , la mappa di Karnaugh ha 4 righe e quattro colonne:

CD \ AB	00	01	11	10
	0	1	1	0
00	0	0	1	0
01	1	0	1	1
11	1	1	1	1
10	1	1	1	0

Mappa di Karnaugh della funzione  $Z$

# Mappe di Karnaugh (4)

		<b>A</b>			
		0	0	1	1
<b>CD</b>	<b>B</b>	0	1	1	0
00		0	0	1	0
01		1	0	1	1
11		1	1	1	1
10		1	1	1	0

I valori delle variabili **A,B,C,D** individuano le “coordinate” delle caselle: le coppie di valori di **A** e **B** (di **C** e **D**) associate alle colonne (alle righe) sono ordinate in modo che tra due caselle adiacenti (della medesima riga o della medesima colonna) cambi il valore di una sola delle variabili, mentre quello di tutte le altre rimane lo stesso; ciò vale anche tra le caselle estreme di ciascuna riga e di ciascuna colonna (che possono quindi essere considerate “adiacenti”, in senso circolare).

# Mappe di Karnaugh (5)

In questo modo a ciascuna coppia di caselle adiacenti contrassegnate con il valore 1 corrispondono, nella espressione booleana, due termini "prodotto" (minterm) nei quali una variabile è presente negata in uno e non negata nell'altro, mentre tutte le altre variabili hanno lo stesso valore.

E' allora possibile semplificare l'espressione sostituendo quei due termini con un unico termine nel quale non è più presente la variabile che cambia valore.

Ad esempio le ultime due caselle della seconda riga nella mappa della funzione **Y** portano alla seguente semplificazione:

$$\mathbf{A \bullet B \bullet C} + \mathbf{A \bullet \bar{B} \bullet C} = \mathbf{A \bullet C}$$

# Mappe di Karnaugh (6)

Allo stesso modo, quaterne di caselle adiacenti tutte con il valore 1 (sulla stessa riga o sulla stessa colonna) corrispondono a quattro termini che si riducono ad uno; ad esempio le quattro caselle della terza riga nella mappa della funzione **Z** portano alla seguente semplificazione:

$$\mathbf{C \bullet D} \bullet (\bar{A} \bullet \bar{B} + \bar{A} \bullet B + A \bullet \bar{B} + A \bullet B) = \mathbf{C \bullet D}$$

le quattro caselle della terza colonna nella mappa della funzione **Z** portano alla seguente semplificazione:

$$\mathbf{A \bullet B} \bullet (\bar{C} \bullet \bar{D} + \bar{C} \bullet D + C \bullet \bar{D} + C \bullet D) = \mathbf{A \bullet B}$$



# Mappe di Karnaugh (7)

Così pure quaterne adiacenti disposte secondo un quadrato producono un unico termine; ad esempio le quattro caselle in basso a sinistra nella mappa della funzione **Z** portano alla seguente semplificazione:

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

Analogo discorso vale per gruppi di otto caselle adiacenti tutte con il valore 1.

# Mappe di Karnaugh (8)

Per semplificare l'espressione di una funzione, si individuano, nella mappa di K., i gruppi di (2 o 4 o 8) caselle adiacenti con il valore 1. Spesso conviene sfruttare la proprietà  $\mathbf{A+A=A}$ , che consente di utilizzare più volte la stessa casella (lo stesso minterm), per formare gruppi diversi e ottenere il maggior numero di semplificazioni possibile.

Individuando un insieme di gruppi (da 1, 2, 4 o 8) che copra tutte le caselle in cui compare il valore 1, si ottiene una espressione semplificata, costituita dall'OR dei termini corrispondenti a ciascun gruppo.

# Mappe di Karnaugh (9)

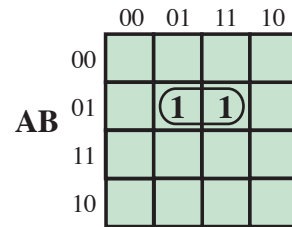
Ad es. per la funzione **Z**, si possono individuare i gruppi segnati in figura:

CD \ AB	00	01	11	10
00	0	0	1	0
01	1	0	1	1
11	1	1	1	1
10	1	1	1	0

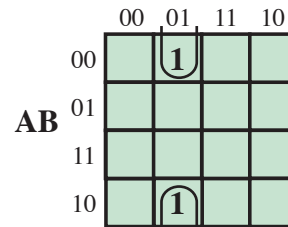
$\bar{A} \cdot C$        $A \cdot B$        $\bar{B} \cdot D$

Si ottiene, immediatamente, l'espressione semplificata:  **$Z = \bar{A} \cdot C + A \cdot B + \bar{B} \cdot D$**

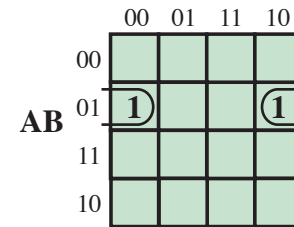
# Esempi di mappe di Karnaugh



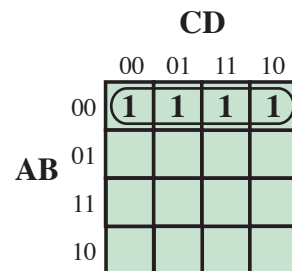
(a)  $\bar{A}BD$



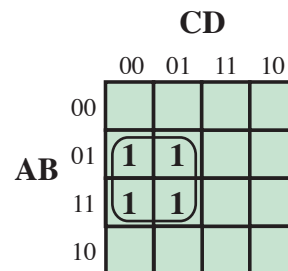
(b)  $\bar{B}\bar{C}D$



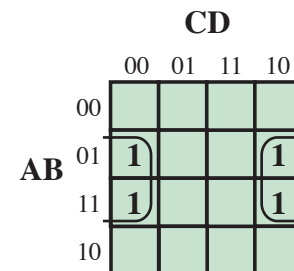
(c)  $\bar{A}B\bar{D}$



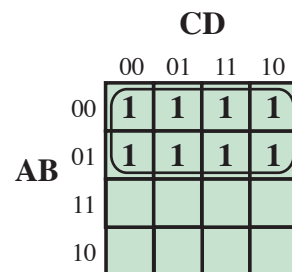
(d)  $\bar{A}\bar{B}$



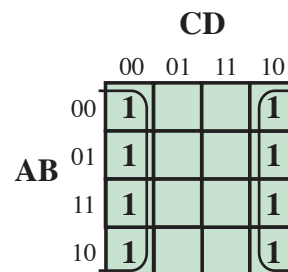
(e)  $B\bar{C}$



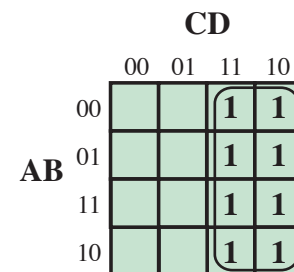
(f)  $BD$



(g)  $\bar{A}$



(h)  $\bar{D}$



(i)  $C$

# Esempi di mappe di Karnaugh

		BC			
		00	01	11	10
A	0			1	1
	1				1

(a)  $F = \bar{A}B + B\bar{C}$

		CD			
		00	01	11	10
AB	00				
	01		1		
	11		1	1	
	10			1	

(b)  $F = B\bar{C}D + ACD$

# Mappe di Karnaugh con don't care

**Funzioni booleane parzialmente definite:** il loro valore è specificato solo per alcune combinazioni dei valori delle variabili.

Le altre combinazioni o non si verificano mai o il valore della funzione non interessa: *don't care conditions* (d.c.c.).

In una mappa di K. è spesso utile inserire un valore 1 al posto di d.c.c. (per formare ulteriori raggruppamenti).

# Mappe di Karnaugh con don't care (2)

Es. Funzione parzialmente definita **W** (i trattini individuano d.c.c.):

A	B	C	W
0	0	0	-
0	0	1	1
0	1	0	-
0	1	1	-
1	0	0	1
1	0	1	-
1	1	0	-
1	1	1	0

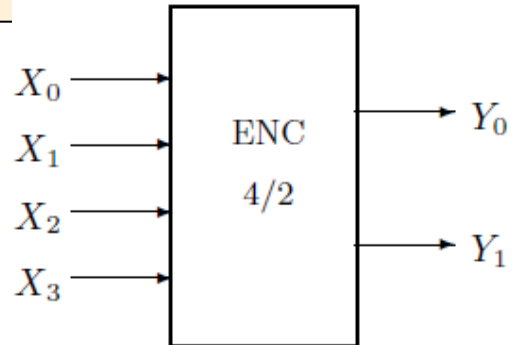
Si possono sostituire due  
d.c.c. con altrettanti 1:

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

A	0	0	1	1
B	0	1	1	0
C				
0	<u>1</u>	-	-	<u>1</u>
1	<u>1</u>	-	0	<u>1</u>

si forma la quaterna con cui si ottiene l'espressione semplificata:  **$W = \bar{B}$**

# Encoder



$X_0$	$X_1$	$X_2$	$X_3$	$Y_1$	$Y_0$
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

**$Y_0$**

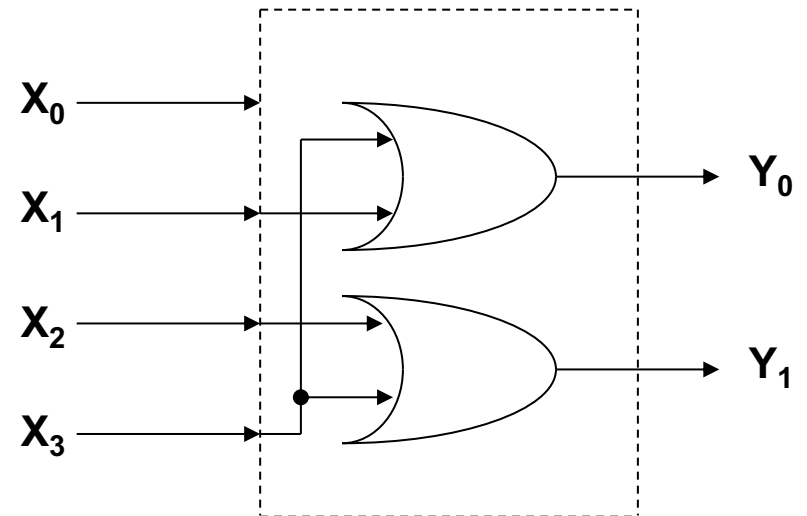
		$X_0$			
$X_2 \backslash X_3$	$X_1$	0	1	1	0
	0	-	1	-	0
0	1	1	-	-	-
1	1	-	-	-	-
1	0	0	-	-	-

$$Y_0 = X_1 + X_3$$

Analogamente:

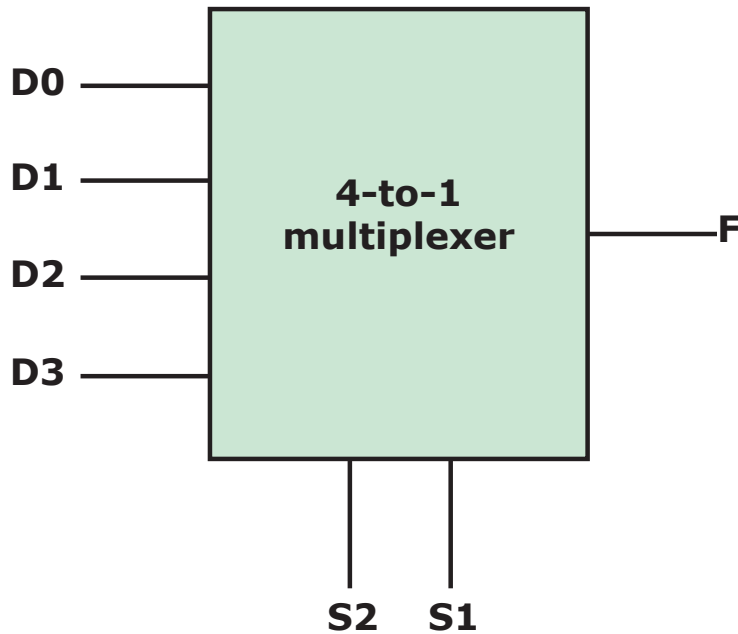
$$Y_1 = X_2 + X_3$$

*Encoder 4/2.*





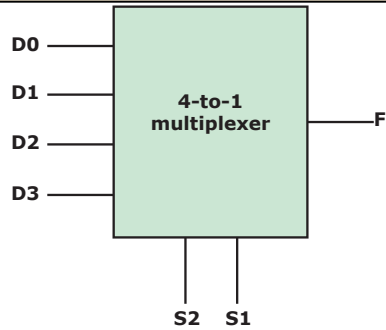
# Multiplexer



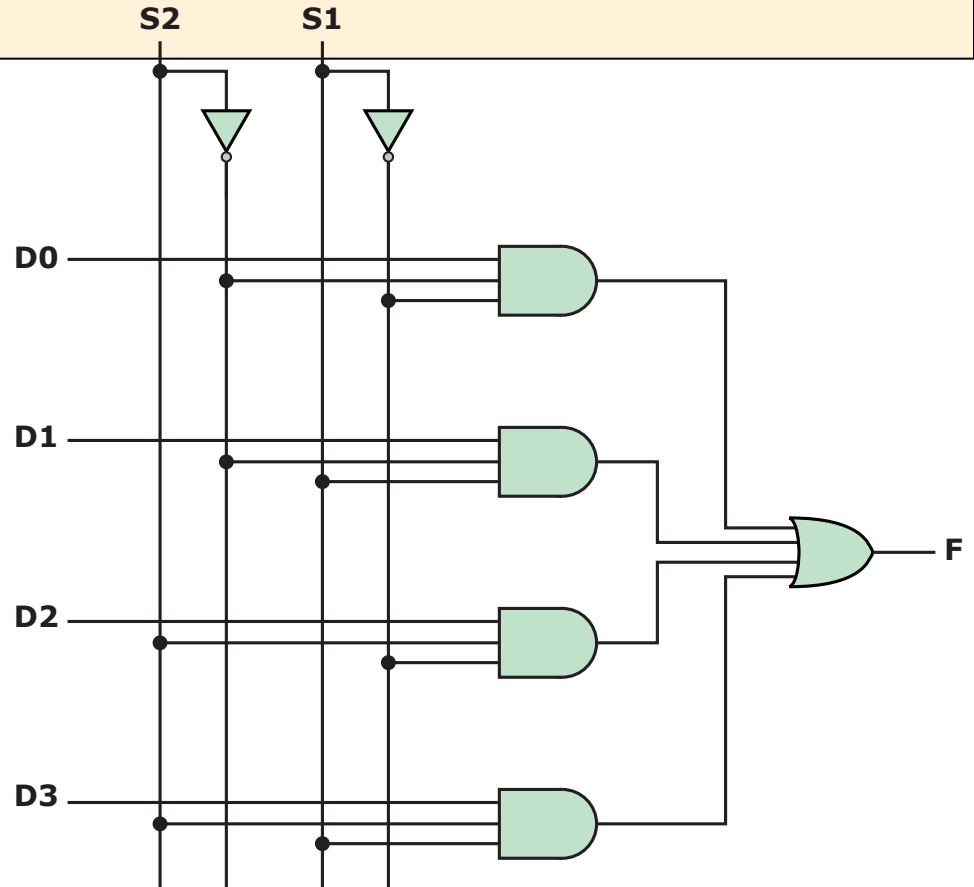
Invia all'uscita F il valore dell'ingresso  $D_i$  selezionato dagli ingressi  $S_i$

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

# Multiplexer



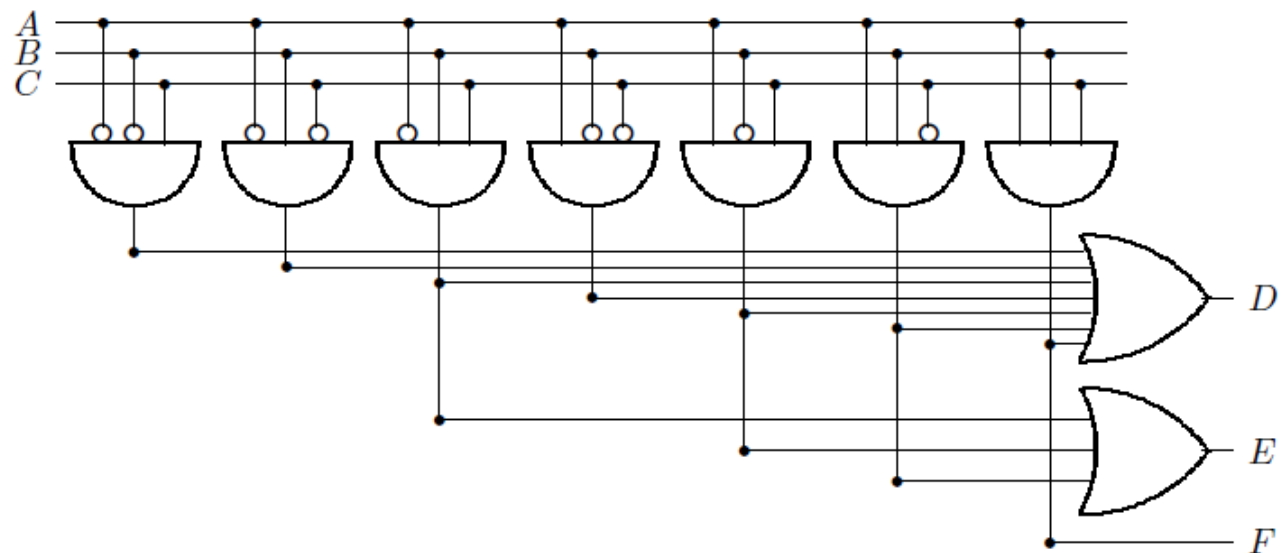
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



# Sintesi a due livelli

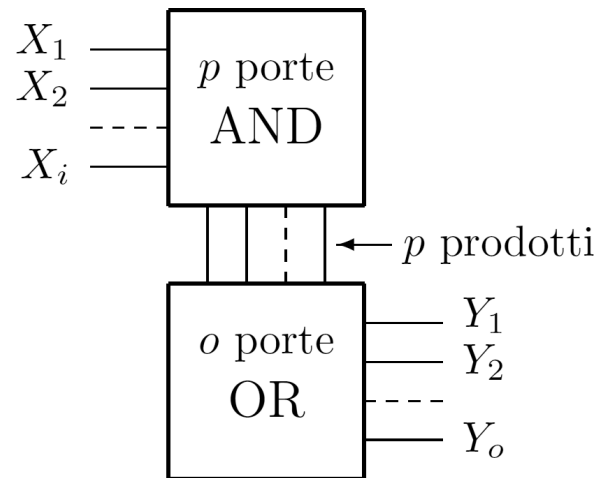
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Sintesi come "somma di prodotti"

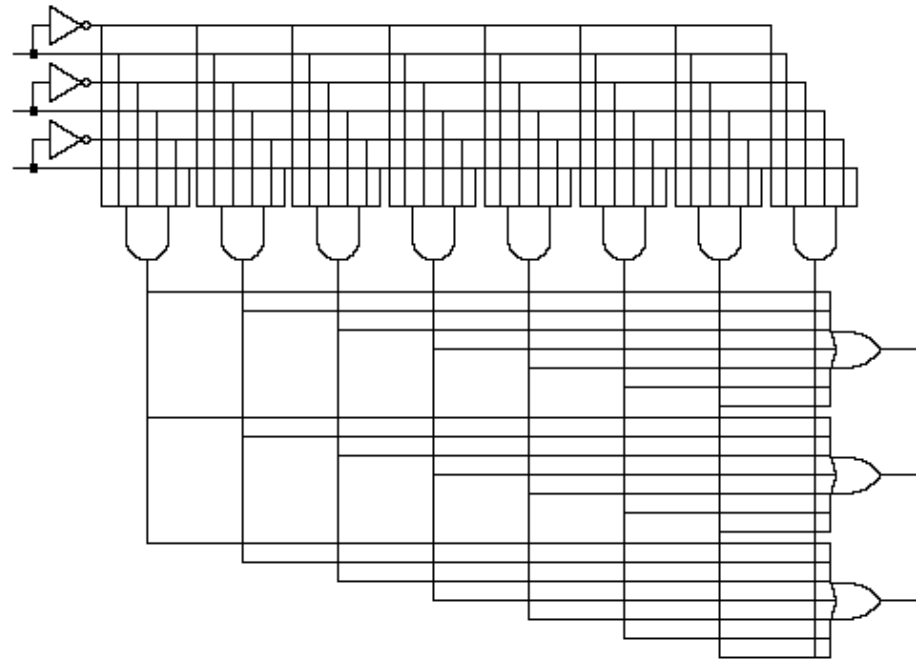


# Sintesi tramite PLA

La sintesi a due livelli è alla base della sintesi tramite PLA = "**P**rogrammable **L**ogic **A**rray"



# Sintesi tramite PLA



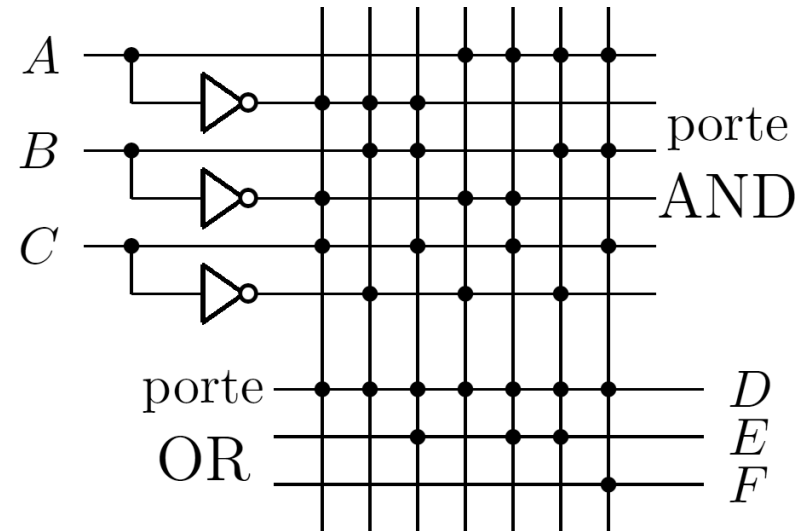
All'interno ci sono collegamenti che possono essere impostati tra gli ingressi (negati e diretti) e le porte AND e da queste agli OR

# Sintesi tramite PLA

Il numero  $p$  di prodotti (porte AND) che servono è uguale al numero di righe della tabella di verità in cui una funzione di uscita vale 1

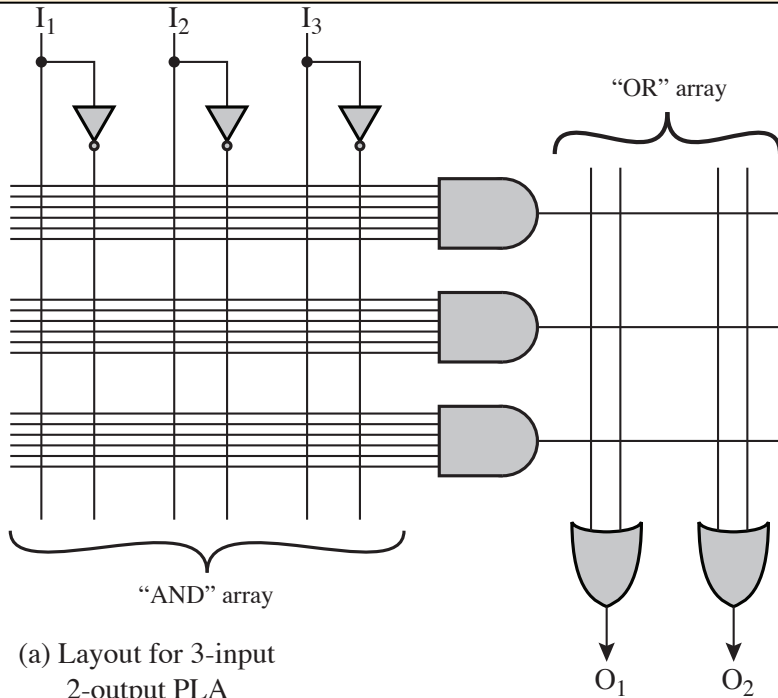
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Es:  $i = 3$ ,  $o = 3$ ,  $p = 7$

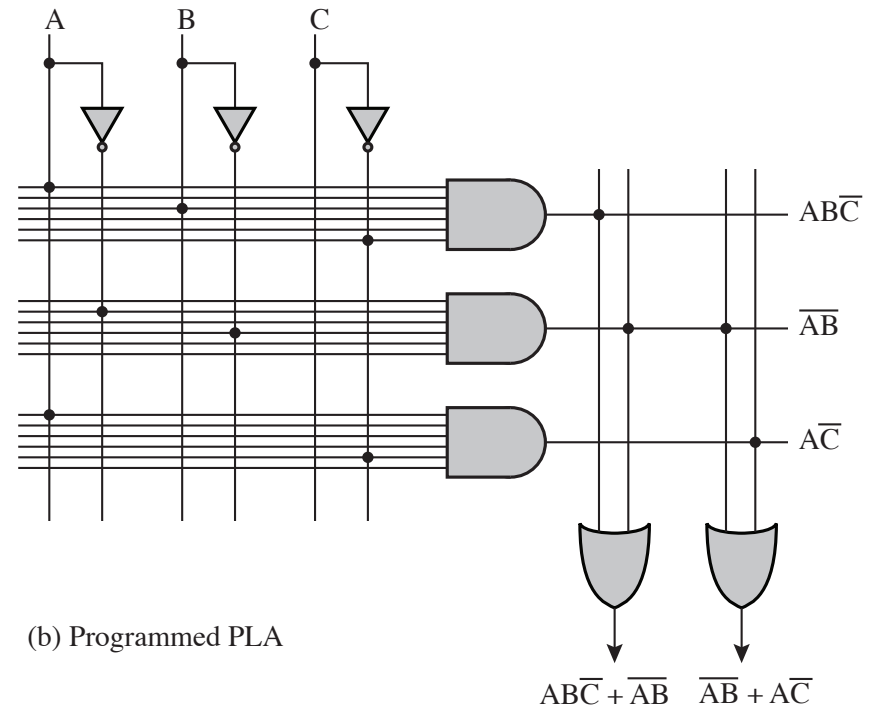


Un'evoluzione di questi dispositivi programmabili sono gli FPGA (Field Programmable Gate Array), che contengono anche elementi di memoria

# PLA : Esempio



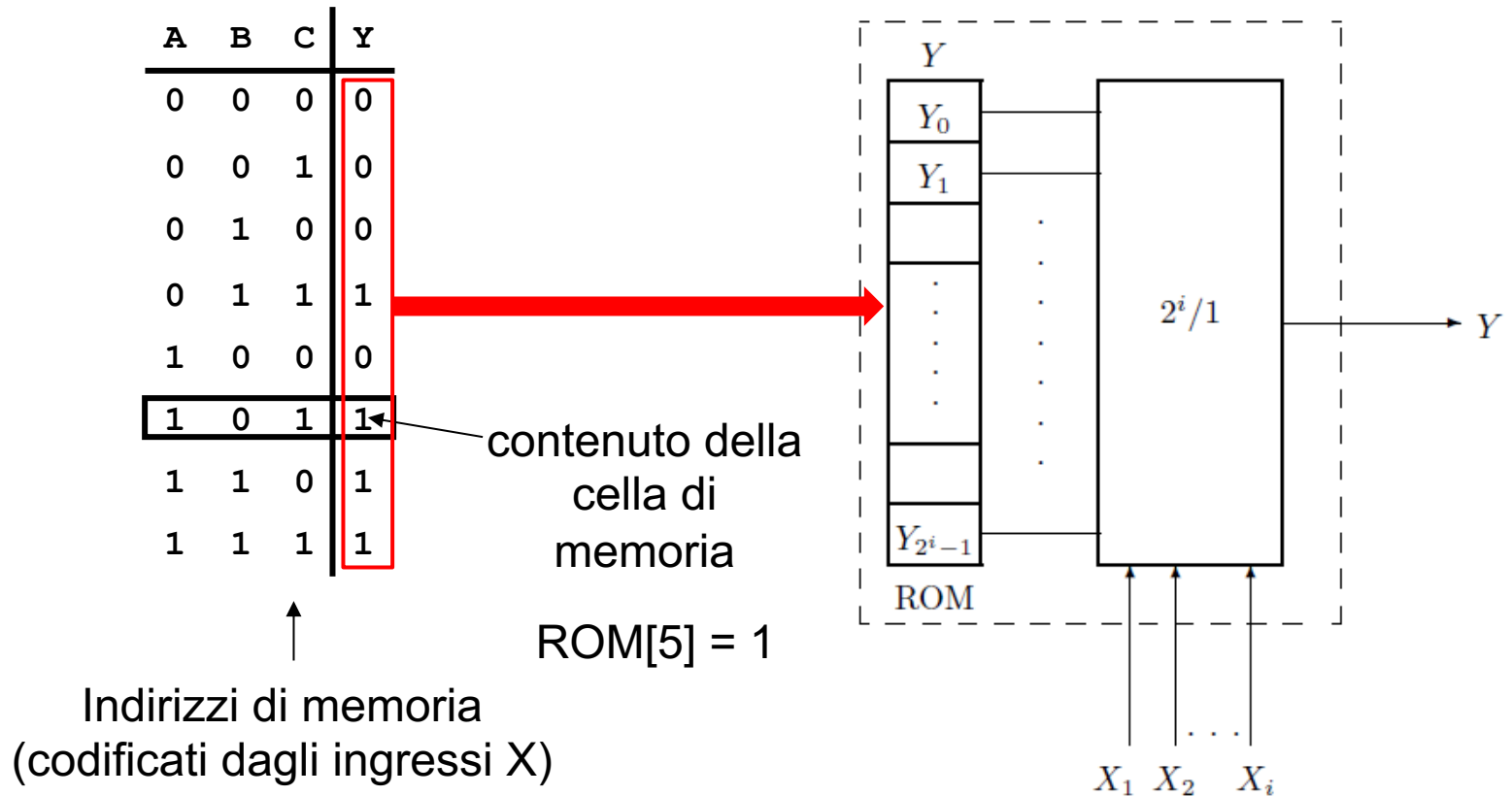
(a) Layout for 3-input  
2-output PLA



(b) Programmed PLA

# Sintesi tramite ROM (1 di 2)

Si può sintetizzare una funzione logica scrivendo in una memoria ROM i valori che la definiscono nella tabella di verità





# Sintesi tramite ROM (2 di 2)

Nella ROM vi sono  $2^i$  righe: ogni riga, individuata dagli  $i$  ingressi, contiene 1 bit per ciascuna delle  $o$  funzioni di uscita

Per una rete con  $i$  input e  $o$  output, serve una ROM di  $o \times 2^i$  bit

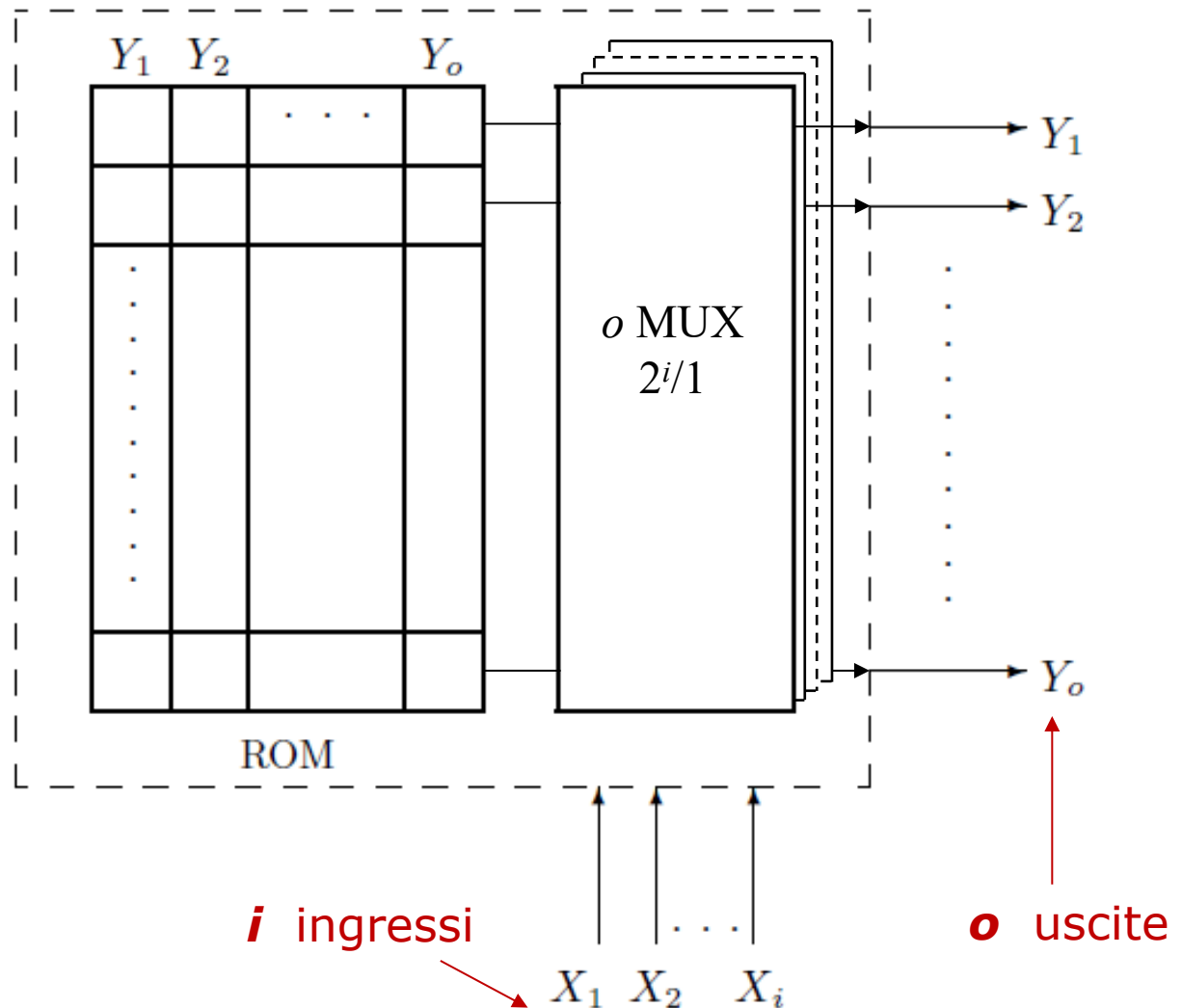


Figura 2.40: Sintesi di più funzioni logiche tramite ROM.

# Esercizio: Sintesi tramite ROM

Realizzare tramite una ROM la seguente tabella di verità

A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1