Reti Combinatorie

Corso di Calcolatori Elettronici A

2007/2008

Sito Web:http://prometeo.ing.unibs.it/quarella

Prof. G. Quarella

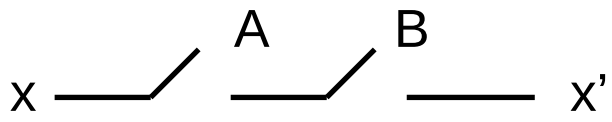
prof@quarella.net

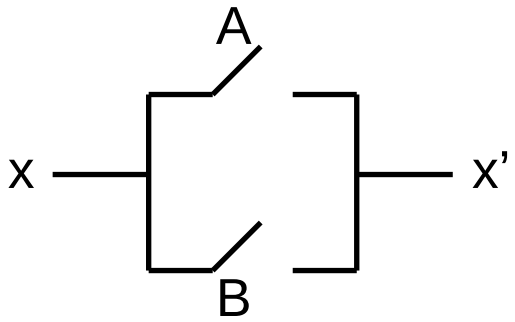
Il livello hardware

- Necessità di un linguaggio preciso per descrivere i fenomeni nel calcolatore
- Molti modelli
  - Fisici: cosa accade nei componenti
  - Elettronici: cosa accade nei circuiti
  - Logici: descrivono le funzioni che si calcolano
- Modelli logici: il calcolatore è descritto ad alto livello di astrazione. Ad esempio: ogni elemento di memoria come un interruttore che si apre o si chiude (es. transistor Mos)

Circuiti, tabelle, funzioni

Il mondo dei segnali





Il mondo delle descrizioni

A	***B***	***circuito***
a	a	a
a	c	a
c	a	a
c	c	c

A	***B***	***circuito***
a	a	a
a	c	c
c	a	c
c	c	c

Ogni circuito ‘calcola ’ una funzione

Convenzione 1

a $\rightarrow$ 0

c $\rightarrow$ 1

A	*B*	$f_1(A,B)$	$f_2(A,B)$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Convenzione 2

a $\rightarrow$ 1

c $\rightarrow$ 0

A	*B*	$f_1'(A,B)$	$f_2'(A,B)$
1	1	1	1
1	0	1	0
0	1	1	0
0	0	0	0

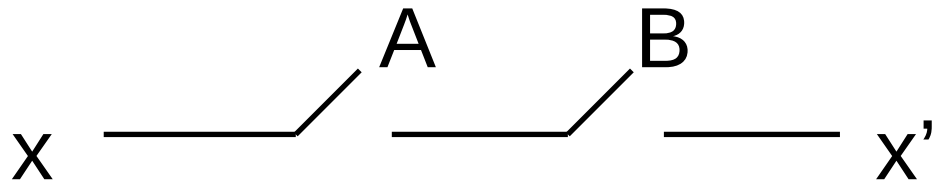
Logica positiva vs. Logica negativa

Funzioni e Formule

- Descrivere una rete in formule: sintetizzare le connessioni con formule
  - Indico con nomi di variabili gli interruttori (connessioni)
  - 0 => passaggio aperto
  - 1 => filo = passaggio chiuso

  } costanti
- Le variabili sono entità matematiche: possono assumere uno dei due valori
- Indico con nomi (simboli) le funzioni (variabili) il cui valore dipende da altri valori

Esempio: funzione di trasmissione



- Indico con f_{AB} la connessione x – x'
- Se assumo la convenzione
  - aperto => 0 e chiuso =>1 (logica positiva)
- Il circuito calcola A ∧ B = f_{AB} *(*detta *funzione di trasmissione)*

Funzione di trasmissione

- Data la formula $A \wedge B$, dove $\wedge$ indica una certa funzione
- Nota la convenzione
- Osservo lo stato fisico
- Deduco valore della funzione
  - $A \rightarrow 1$
  - $B \rightarrow 0$
  - $1 \wedge 0 \rightarrow 0$
- Verifico: $0 \rightarrow$ non passa ok

Funzioni booleane

- Funzione booleana $f(x_0, \ldots, x_{n-1})$
- $f : \{0,1\}^n \rightarrow \{0,1\}$
- Una funzione booleana di ***n*** variabili può anche essere descritta da una ***tabella*** (***di verità***) che elenca, per ognuna delle 2^n possibili assegnazioni di valori, il corrispondente valore della funzione
- ***n*** è sempre **finito**
- Esistono 2^{2^n} funzioni booleane di *n* variabili distinte

Perché 2^{2^n} funzioni booleane di n variabili?

Esempio con $n = 2$

		and							**xor**	**or**	**nor**	**≡**				**nand**
x_1	x_0	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

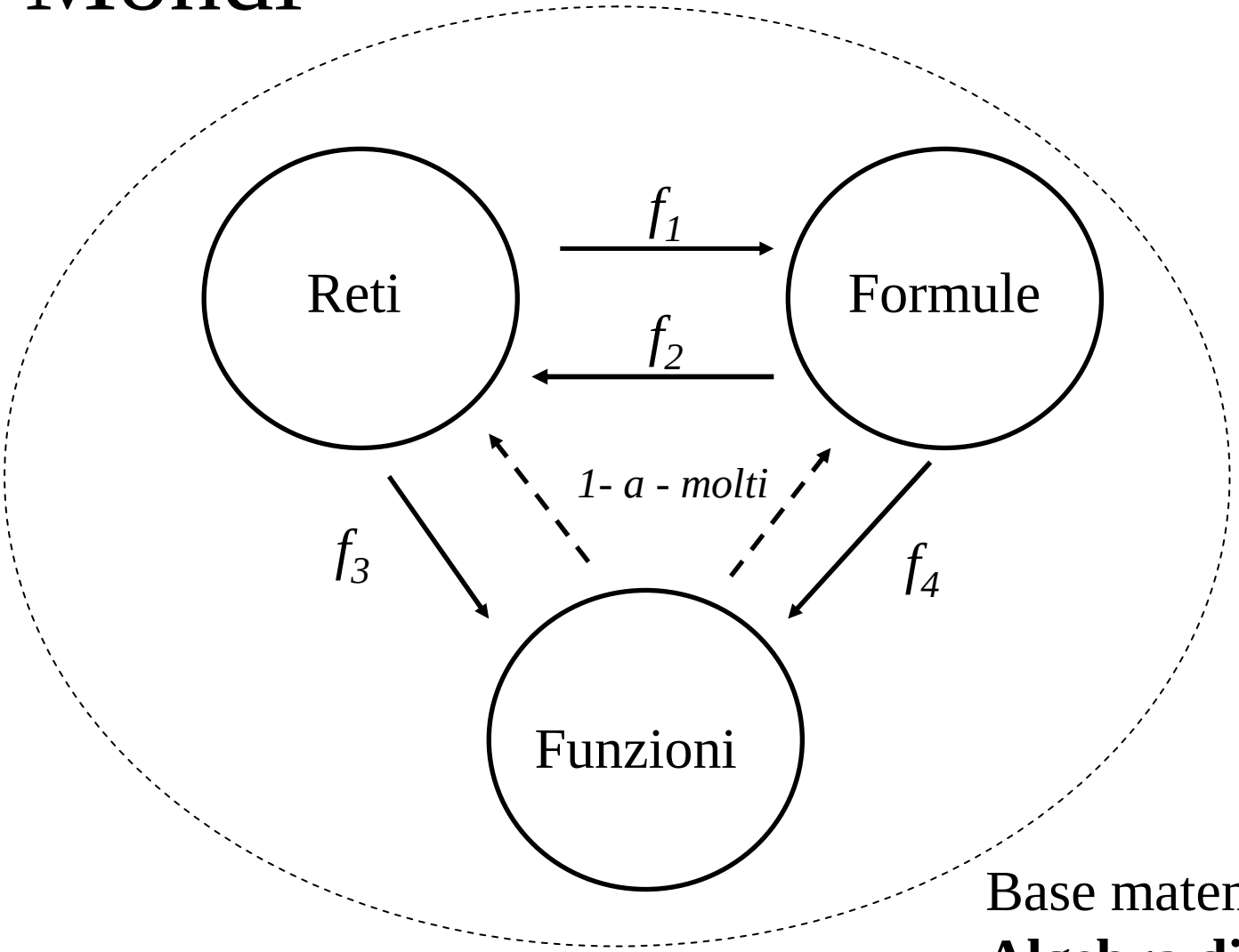
2^n configurazioni distinte

16 funzioni (2^4)

Circuiti, Funzioni, Formule

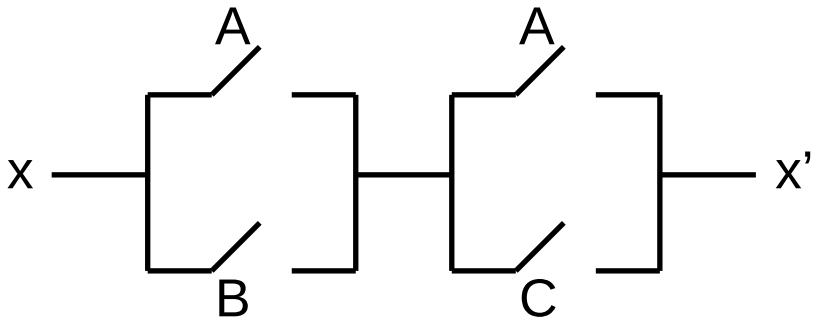
- I circuiti sono descritti da funzioni (tabelle) e da formule
- Le tabelle e le formule sono realizzate da circuiti (che ne calcolano il valore, dati i valori delle variabili di ingresso e date le convenzioni di corrispondenza)

3 Mondi

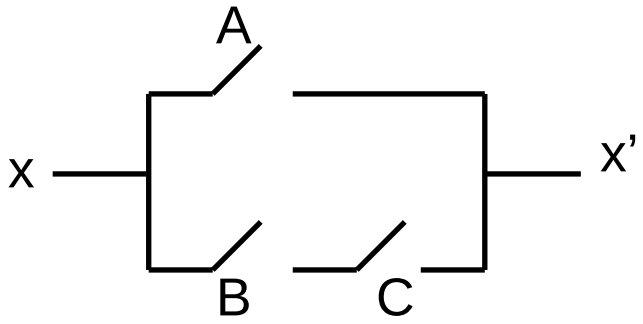


Base matematica:
Algebra di Boole

Esempio: 2 reti, 2 formule, 1 funzione



$f_{xx'} = (A+B)\bullet(A+C)$



$f_{xx'} = A+B\bullet C$

rappresentano la stessa funzione

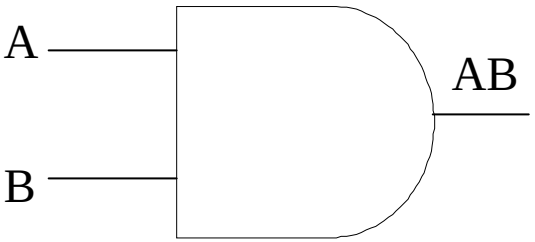
Porte logiche

- Porta: dispositivo fisico che realizza un elemento di calcolo binario
  - I segnali applicati e prodotti possono assumere due soli valori: segnale alto e segnale basso
- Caratteristiche:
  - Numero prefissato di linee di ingresso
  - Una linea di uscita
- Le linee di ingresso/uscita possono essere collegate a più linee di circuito
  - fan in: max numero di ingressi
  - fan out: max numero di uscite

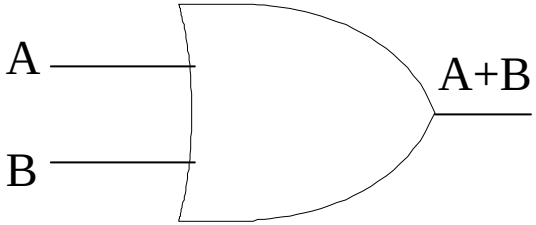
Porte logiche

- Le porte sono realizzate con tecnologie diverse
- Parametri di una porta:
  - Assorbimento di energia (frazioni di watt)
  - Ritardo introdotto
  - Immunità dal rumore
- Le porte logiche sono dispositivi fisici la cui attività ad un opportuno livello di astrazione può essere interpretata come il calcolo di operatori booleani o funzioni booleane più complesse

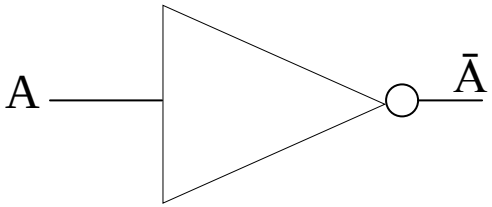
Porte logiche



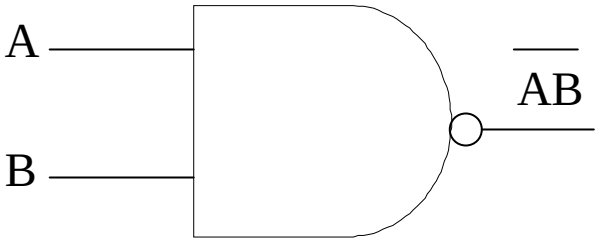
AND



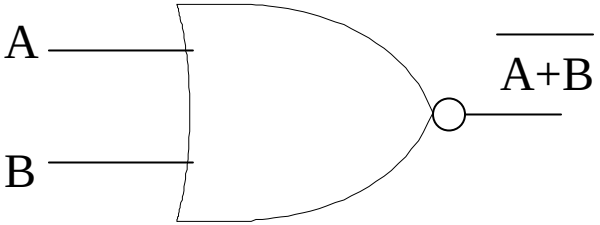
OR



NOT



NAND



NOR

Algebra di Boole

- È lo **strumento matematico** usato per lo studio delle reti combinatorie
- È un particolare tipo di algebra che include:
  - un insieme di supporto **A** (l'insieme **{0,1}** nel nostro caso)
  - degli operatori binari: **AND** (·) e **OR** (+)
  - un operatore complemento: **NOT** ($\bar{}$)
- Gli operatori soddisfano certe **proprietà** che si deducono da un insieme di **assiomi**

Assiomi e alcune proprietà dell'Algebra di Boole

		AND	OR
Assiomi	Commutatività	$AB = BA$	$A+B = B+A$
	Distributività	$A+BC=(A+B)(A+C)$	$A(B+C)=AB+AC$
	Identità	$1A = A$	$0+A = A$
	Inverso	$A\bar{A} = 0$	$A+\bar{A} = 1$
		dualità	
Proprietà	Elem. nullo	$0A = 0$	$1+A = 1$
	Idempotenza	$AA = A$	$A+A = A$
	Assorbimento	$A(A+B) = A$	$A+AB=A$
	Associatività	$(AB)C = A(BC)$	$(A+B)+C=A+(B+C)$
	De Morgan	$\overline{AB} = \bar{A}+\bar{B}$	$\overline{A+B} = \bar{A}\,\bar{B}$

Formule (espressioni) booleane

1. Le **costanti 0** e **1** e le **variabili** (simboli a cui possono essere associati i valori 0 e 1) sono espressioni booleane
2. Se E, E_1 ed E_2 sono espressioni booleane lo sono anche $\mathbf{(E_1+E_2)}$, $\mathbf{(E_1 \cdot E_2)}$ e $\mathbf{(\overline{E})}$
3. Non esistono altre espressioni oltre a quelle che possono essere generate da un numero finito di applicazioni delle regole 1 e 2

Esempi di formule booleane

- $(\overline{(x_1+x_2)}\cdot x_3)$
- $((x_1\cdot\overline{x_2})+(x_3\cdot(x_4+\overline{x_5})))$

Valgono le regole classiche di semplificazione delle parentesi e di priorità degli operatori:

$((x_1\cdot x_2)+(x_3\cdot(x_4+x_5))) \Rightarrow x_1\cdot x_2+x_3\cdot(x_4+x_5)$

… e il simbolo "$\cdot$" di solito si omette

Equivalenza fra formule booleane Esempi

- $x_1x_2 + x_1\overline{x}_2x_3 = x_1\,(x_2+\overline{x}_2x_3)$
- $x_1 + x_2 + x_2x_3 + \overline{x}_2x_3 = x_1 + x_2 + x_3(x_2+\overline{x}_2) = x_1 + x_2 + x_3$
- $x_1x_2 + x_1x_2x_3 + x_1x_2 = x_1x_2 + x_1x_2x_3 = x_1x_2\,(1 + x_3) = x_1x_2$

Equivalenza fra espressioni booleane Esempio

$$x_1 + x_2 + x_2x_3 + \overline{x}_2x_3 = x_1 + x_2 + x_3$$

x_3	x_2	x_1	$\overline{x}_2$	x_2x_3	$\overline{x}_2x_3$	$x_1+x_2+x_2x_3+\overline{x}_2x_3$	$x_1+x_2+x_3$
0	0	0	1	0	0	0	0
0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	1	1	1
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	1

Le 3 funzioni di base: Tabelle di verità

x_1	x_0	$x_1 \bullet x_0$
0	0	0
0	1	0
1	0	0
1	1	1

AND

x_1	x_0	$x_1 + x_0$
0	0	0
0	1	1
1	0	1
1	1	1

OR

x	$\overline{x}$
0	1
1	0

NOT

(Assumendo di adottare la logica positiva)

Altre funzioni tipiche

x_1	x_0	$\overline{x_1 \bullet x_0}$
0	0	1
0	1	1
1	0	1
1	1	0

NAND

x_1	x_0	$\overline{x_1 + x_0}$
0	0	1
0	1	0
1	0	0
1	1	0

NOR

Tabelle di verità e proprietà dell'Algebra di Boole: Esempio

Proprietà di De Morgan: $\overline{x_1 x_0} = \overline{x_1} + \overline{x_0}$

x_1	x_0	$x_1 x_0$	$\overline{x_1 x_0}$	$\overline{x_1}$	$\overline{x_0}$	$\overline{x_1} + \overline{x_0}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Espressioni e funzioni booleane

- Ad una espressione booleana di *n* variabili corrisponde un'**unica** funzione booleana di *n* variabili (cioè una tabella)
- Viceversa, ad una funzione booleana di *n* variabili corrispondono **infinite** espressioni booleane di *n* variabili (cioè se partiamo da una tabella scopriamo che vi sono infinite espressioni tra loro equivalenti)

Rappresentazioni canoniche

- Tra le diverse espressioni booleane equivalenti corrispondenti a una stessa funzione se ne individuano due, chiamate:
  - Forma canonica disgiuntiva (o “**somma di prodotti**”)
  - Forma canonica congiuntiva (o “**prodotto di somme**”)
- Nel seguito, faremo ampio uso della prima

Esempio
Funzione booleana a 3 variabili

x_2	x_1	x_0	$f(x_0, x_1, x_2)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$f = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1\overline{x}_0 + x_2\overline{x}_1x_0$$

Mintermini

(funzioni mintermine)

Esempio con 3 variabili

$\bar{x}_2\bar{x}_1\bar{x}_0$	m_0	(000)
$\bar{x}_2\bar{x}_1 x_0$	m_1	(001)
$\bar{x}_2 x_1\bar{x}_0$	m_2	(010)
$\bar{x}_2 x_1 x_0$	m_3	(011)
$x_2\bar{x}_1\bar{x}_0$	m_4	(100)
$x_2\bar{x}_1 x_0$	m_5	(101)
$x_2 x_1\bar{x}_0$	m_6	(110)
$x_2 x_1 x_0$	m_7	(111)

Forma canonica disgiuntiva “somma di prodotti”

Quindi, la formula precedente

$$f = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1\overline{x}_0 + x_2\overline{x}_1x_0$$

si può scrivere come

$$\boldsymbol{f = m_1 + m_2 + m_5}$$

Reti combinatorie: definizione

- Una rete combinatoria è un circuito elettronico in grado di ***calcolare in modo automatico funzioni binarie di una o più variabili binarie***
- Le **uscite** di una rete combinatoria dipendono unicamente dai **valori di ingresso**

Reti combinatorie: elementi costitutivi

Una **rete combinatoria** è costituita da

- un insieme di elementi attivi: le **porte logiche**
- un insieme di elementi passivi:
  - **linee di ingresso**: solo un'estremità della linea è collegata all'ingresso di una porta
  - **linee di uscita**: solo un'estremità della linea è collegata all'uscita di una porta
  - **linee di circuito**: collegano l'uscita di una porta con l'ingresso di una diversa porta

Reti combinatorie: regole di costruzione

- Una rete combinatoria **ben strutturata** è quindi un collegamento di porte tramite linee tale che:
  - **ogni ingresso** di ogni porta deve essere collegato a una linea di ingresso o a una linea di circuito
  - **ogni uscita** di ogni porta deve essere collegata una linea di uscita o a una linea di circuito
  - il collegamento tra porte non deve dare luogo a cicli

Dalle funzioni ai circuiti: sintesi di reti combinatorie

Sintesi: come si fa a progettare un circuito che calcola una o più funzioni?

1. Individuare le **variabili** di ingresso e di uscita e la **tabella** di verità
2. Derivare dalla tabella un'**espressione booleana** (ad esempio quella in forma canonica disgiuntiva) per ognuna delle *m* linee di uscita
3. Costruire la **rete** combinatoria associata alle *m* espressioni booleane

Esempio
Funzione di maggioranza

restituisce 1 se la maggioranza degli input è 1, 0 altrimenti

x_2	x_1	x_0	$f(x_0, x_1, x_2)$
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

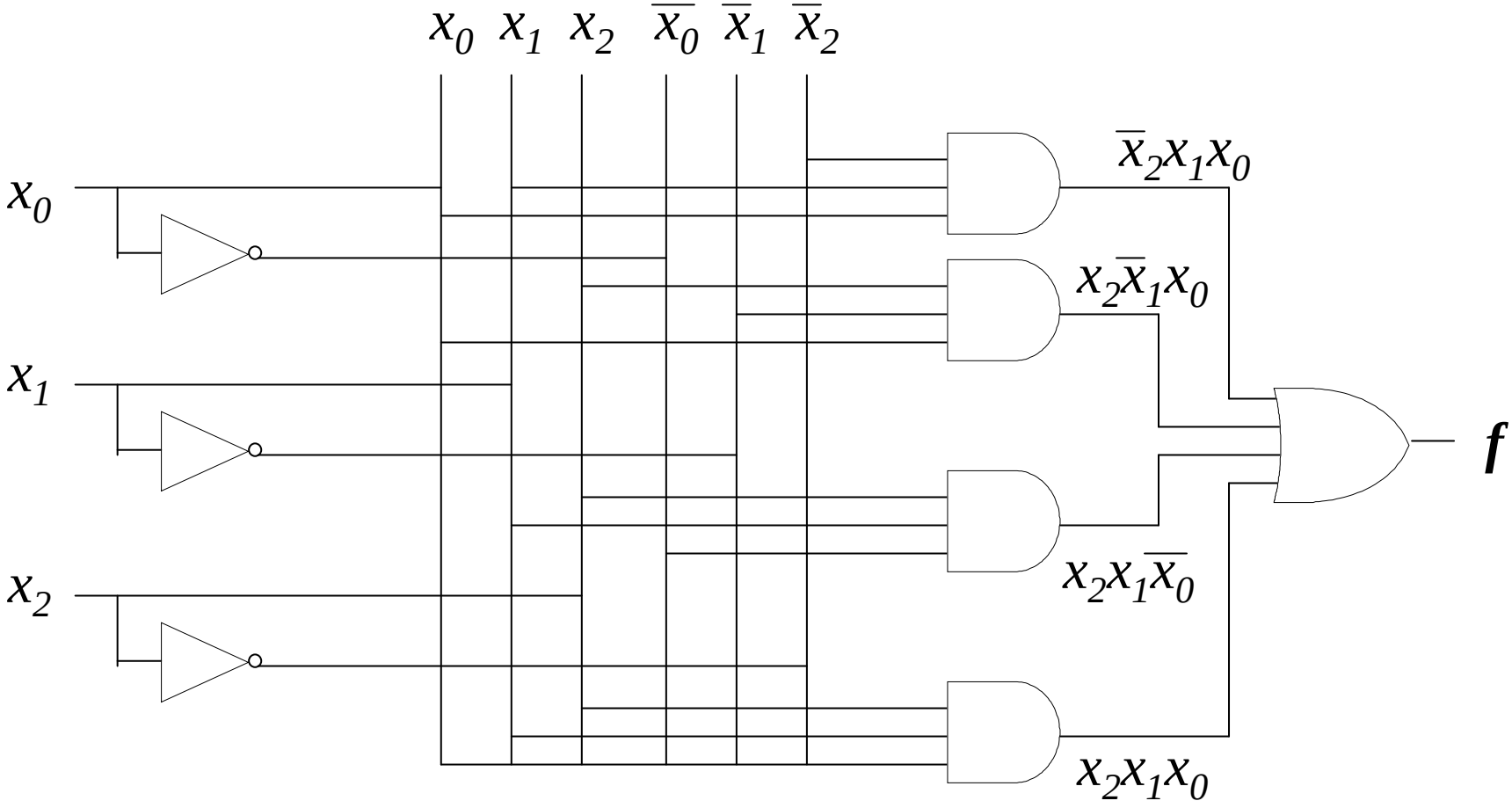
n = 3 ingressi

m = 1 uscite

$$f(x_0, x_1, x_2) = \overline{x}_2 x_1 x_0 + x_2 \overline{x}_1 x_0 + x_2 x_1 \overline{x}_0 + x_2 x_1 x_0$$

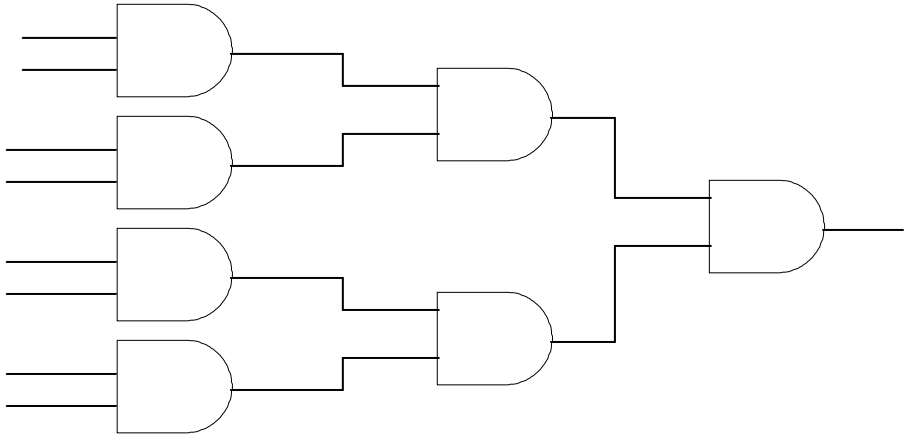
Attenzione: la formula *f* si può semplificare!

Circuito per la funzione dei maggioranza



Porte a k > 2 ingressi

- Come visto nell'es. precedente spesso è preferibile disporre di **porte con un numero arbitrario k di ingressi**
- Avendo a disposizione **porte a 2 ingressi**, le porte a k ingressi vengono realizzate tramite alberi binari di porte a 2 ingressi



Funzione AND con 8 linee di ingresso realizzata con porte AND a 2 ingressi

Condizioni di indifferenza
Don’t cares

- Può accadere che alcune configurazioni di ingresso non si presentino mai ad un circuito
- I valori della funzione in corrispondenza delle configurazioni di ingresso non utilizzate possono essere trascurati, per questo vengono chiamati **valori di indifferenza**
- Questi valori sono in genere indicati con **x** oppure **d**
- Esempio: **codifica BCD** (Binary Coded Decimal):
  - rappresenta le **cifre decimali** mediante gruppi di **4 bit**…
  - … con 4 bit si ottengono **16** configurazioni di ingresso ma ne bastano **10** per rappresentare i numeri da 0 a 9

Condizioni di indifferenza

Esempio: codifica BCD

Cifra decimale rappresentata	**Codifica binaria**			
	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Condizioni di indifferenza

Esempio: codifica BCD

Cifra decimale rappresentata	**Codifica binaria**				***f***
	b_3	b_2	b_1	b_0	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
	1	0	1	0	x
	1	0	1	1	x
	1	1	0	0	x
	1	1	0	1	x
	1	1	1	0	x
	1	1	1	1	x

Uso delle condizioni di indifferenza

- I valori della funzione corrispondenti alle condizioni di indifferenza possono essere considerati 0 o 1, ciò permette di semplificare le espressioni

Cifra decimale rappresentata	Codifica binaria				*f*	
	b_3	b_2	b_1	b_0		
0	0	0	0	0	0	
1	0	0	0	1	0	
2	0	0	1	0	0	
3	0	0	1	1	1	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	
7	0	1	1	1	0	
8	1	0	0	0	0	
9	1	0	0	1	1	
	1	0	1	0	x	
	1	0	1	1	x	← a
	1	1	0	0	x	
	1	1	0	1	x	← b
	1	1	1	0	x	← c
	1	1	1	1	x	← d

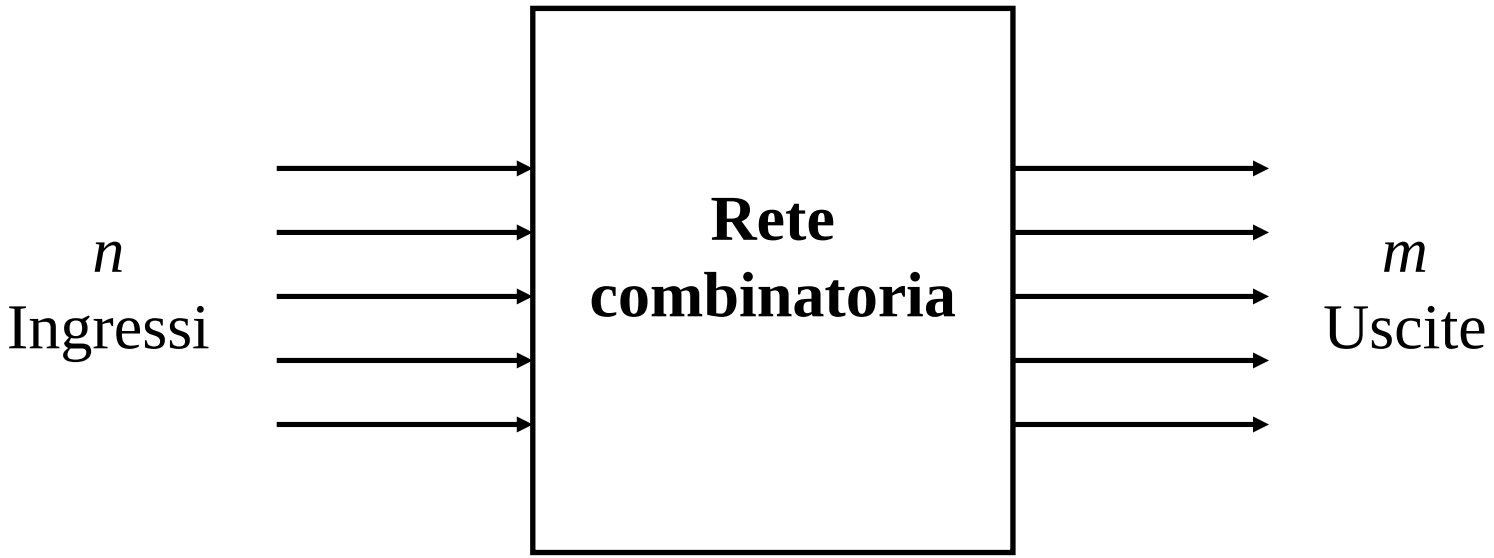
$$f = \overline{b}_3\overline{b}_2 b_1 b_0 + \overline{b}_3 b_2 b_1 \overline{b}_0 + b_3\overline{b}_2\overline{b}_1 b_0 =$$

(e) (f) (g)

$$\overline{b}_2 b_1 b_0 + b_2 b_1 \overline{b}_0 + b_3 b_0$$

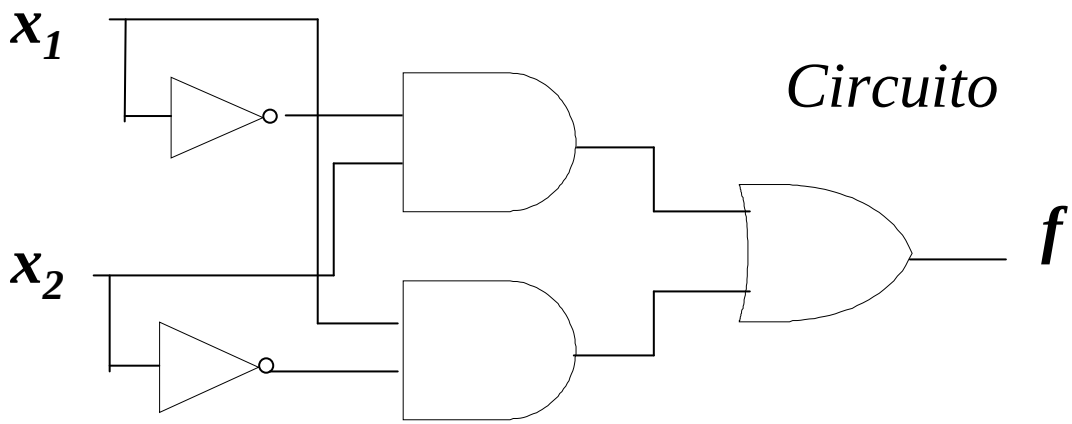
(e+a) (f+c) (g+a+b+d)

Dai circuiti alle funzioni: analisi di reti combinatorie

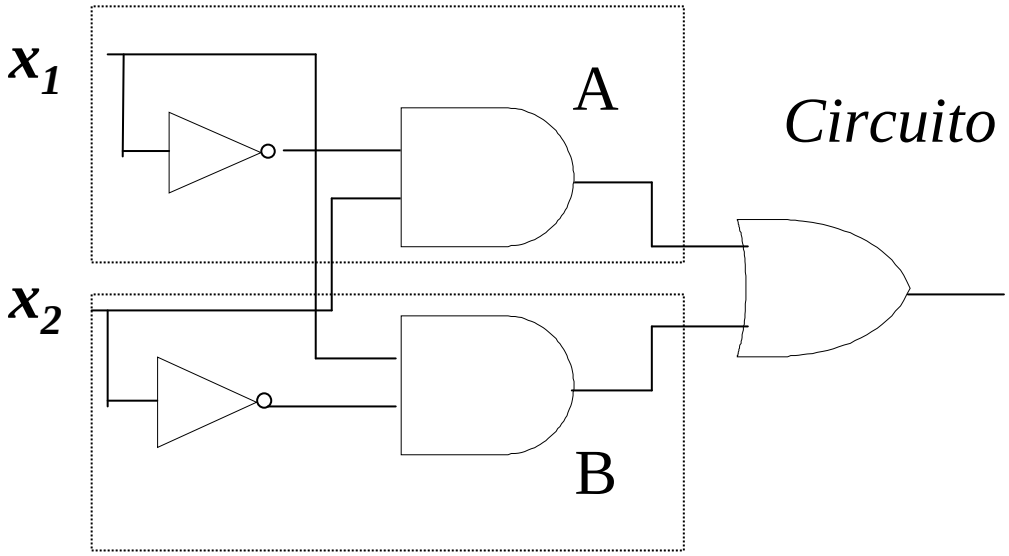


Analisi: ho un certo circuito combinatorio, che cosa calcola?

Esempio



Esempio



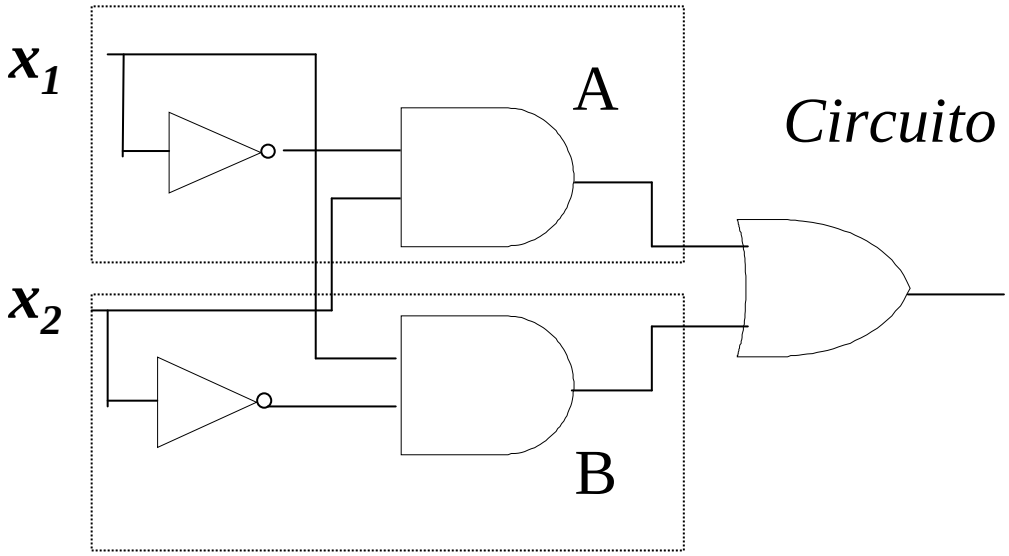
Circuito

f

Espressione booleana

$f = A + B$

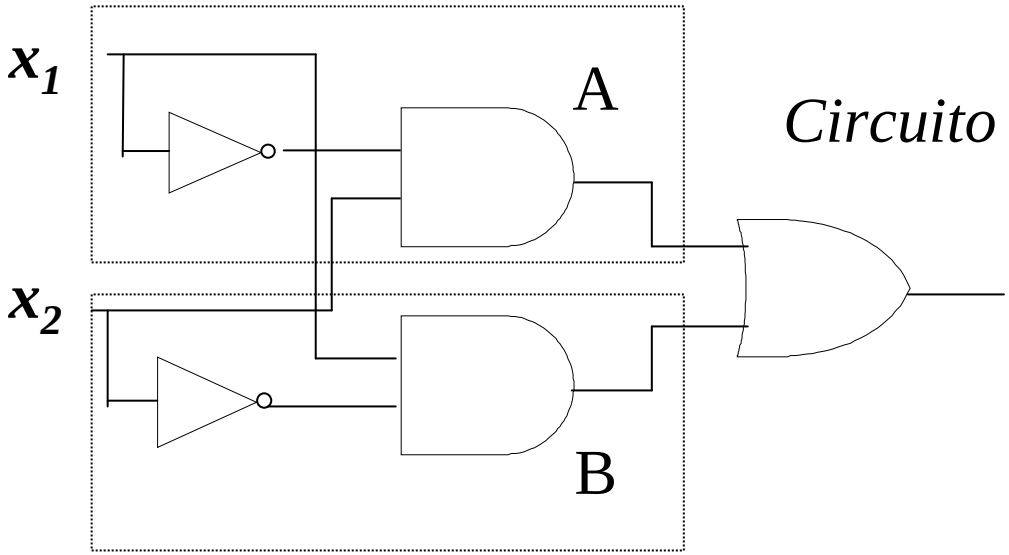
Esempio



Espressione booleana

$f = A + B = \overline{x_1} x_2 + x_1 \overline{x_2}$

Esempio



Espressione booleana

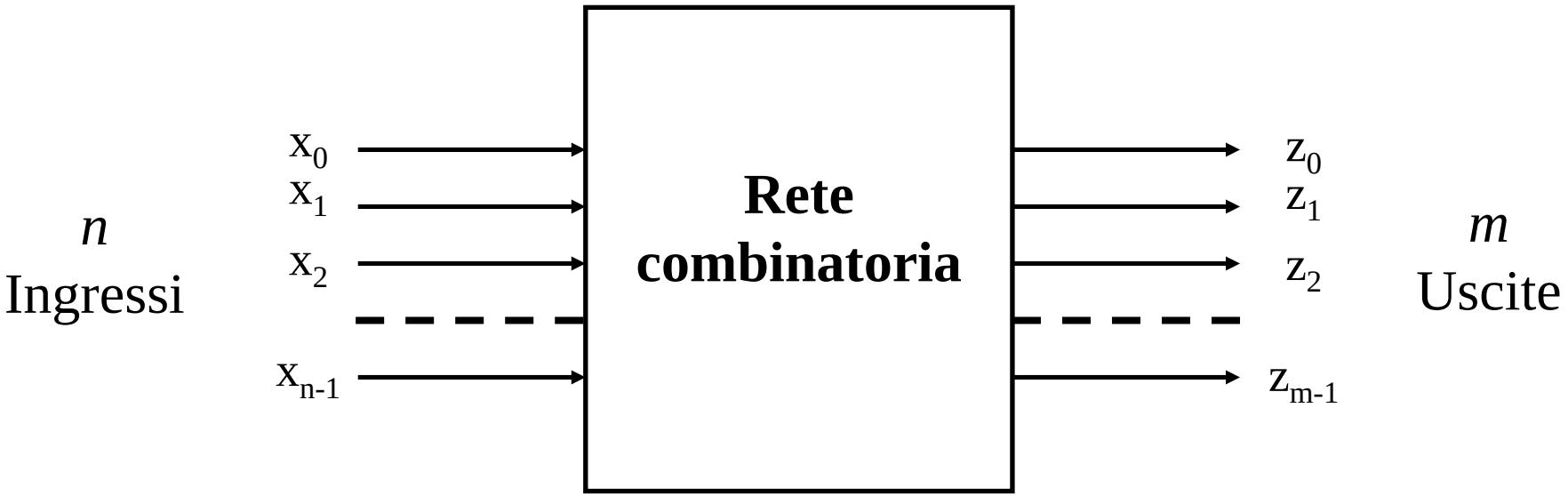
$f = A + B = \overline{x_1} x_2 + x_1 \overline{x_2}$

x_2	x_1	$\overline{x_2}$	$\overline{x_1}$	$x_1\overline{x_2}$	$\overline{x_1}x_2$	$x_1\overline{x_2} + \overline{x_1}x_2$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

Funzione (XOR - ⊕)

Reti combinatorie:

come scatole nere descritte da espressioni booleane



Il comportamento della rete è descritto da m espressioni booleane in n variabili

Alcuni circuiti tipici

- **Codificatori**
- **Decodificatori**
- **Multiplexer**
- **Demultiplexer**
- **Sommatore, sottrattore, comparatore**
- **Logiche strutturate: PLA, ROM**

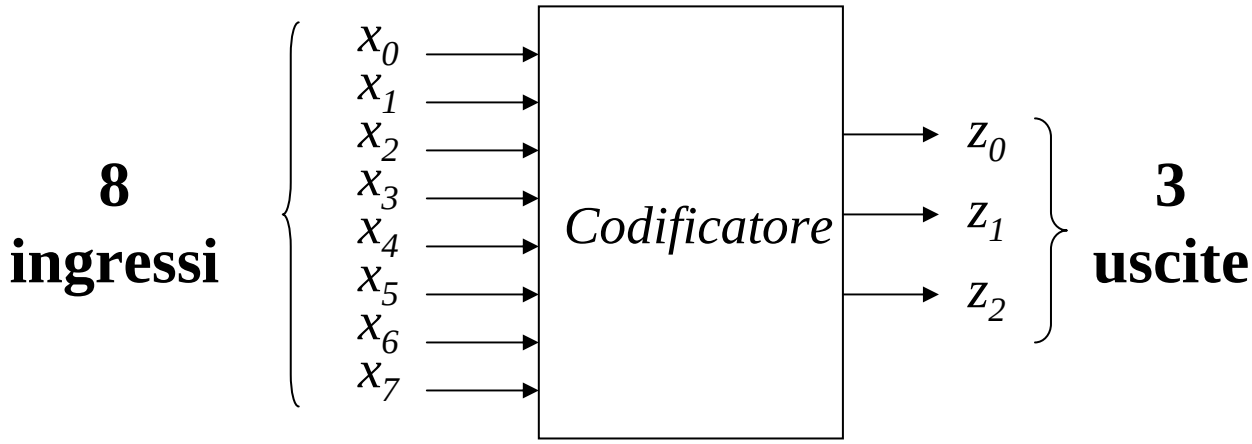
Codificatori

- Codificatore *n-a-m*: rete combinatoria con *n* linee di ingresso ed *m* linee di uscita (con $m = \lceil \log_2(n) \rceil$)
- **Combinazioni ammesse**: quelle che contengono **un solo uno** ed ***n-1* zeri** (*n* combinazioni in tutto)
- L'**uscita** del codificatore è la **codifica binaria dell'indice *i*** dell'unica linea di ingresso attiva

$$i = z_{m-1}2^{m-1} + z_{m-2}2^{m-2} + \ldots + z_1 2^1 + z_0 2^0$$

- **<u>Esempio:</u>**
  - codificatore 64-a-6, linea di ingresso attiva x_{21}
  - Siccome $21_{10} = 010101_2$, si avrà $z_0=1$, $z_1=0$, $z_2=1$, $z_3=0$, $z_4=1$, $z_5=0$

Codificatori
es. 8-a-3



- Supponiamo che in ingresso sia attiva la linea 4
- Poiché $4_{10} = 100_2$
- Le 3 uscite saranno $z_0 = 0$, $z_1 = 0$ e $z_2 = 1$

Codificatori

Tabella di transizione

- **Supponiamo n=4, m=2**

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

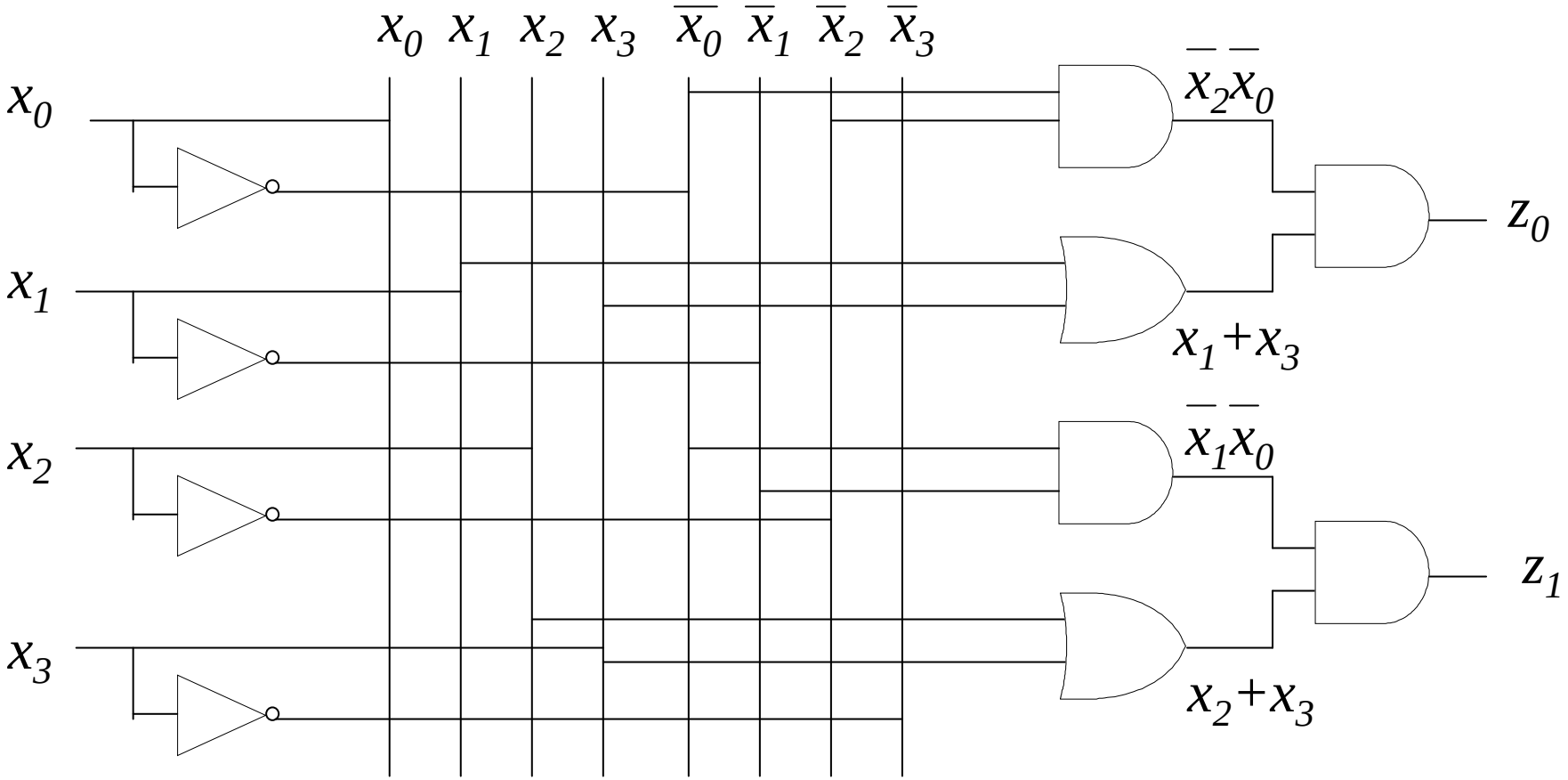
$$z_0 = \bar{x}_3 x_2 x_1 \bar{x}_0 + x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0 = \bar{x}_2 x_1 \bar{x}_0 + x_3 \bar{x}_2 \bar{x}_0 = \mathbf{x_2 \bar{x}_0 (x_1 + x_3)}$$

$$z_1 = \bar{x}_3 x_2 \bar{x}_1 \bar{x}_0 + x_3 \bar{x}_2 \bar{x}_1 \bar{x}_0 = x_2 \bar{x}_1 \bar{x}_0 + x_3 \bar{x}_1 \bar{x}_0 = \mathbf{x_1 \bar{x}_0 (x_2 + x_3)}$$

Usando le condizioni di indifferenza ed attribuendo una priorità maggiore a x_i rispetto a x_{i+1}

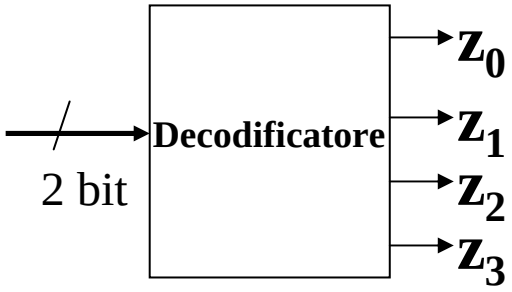
Codificatori

Circuito combinatorio



Decodificatori

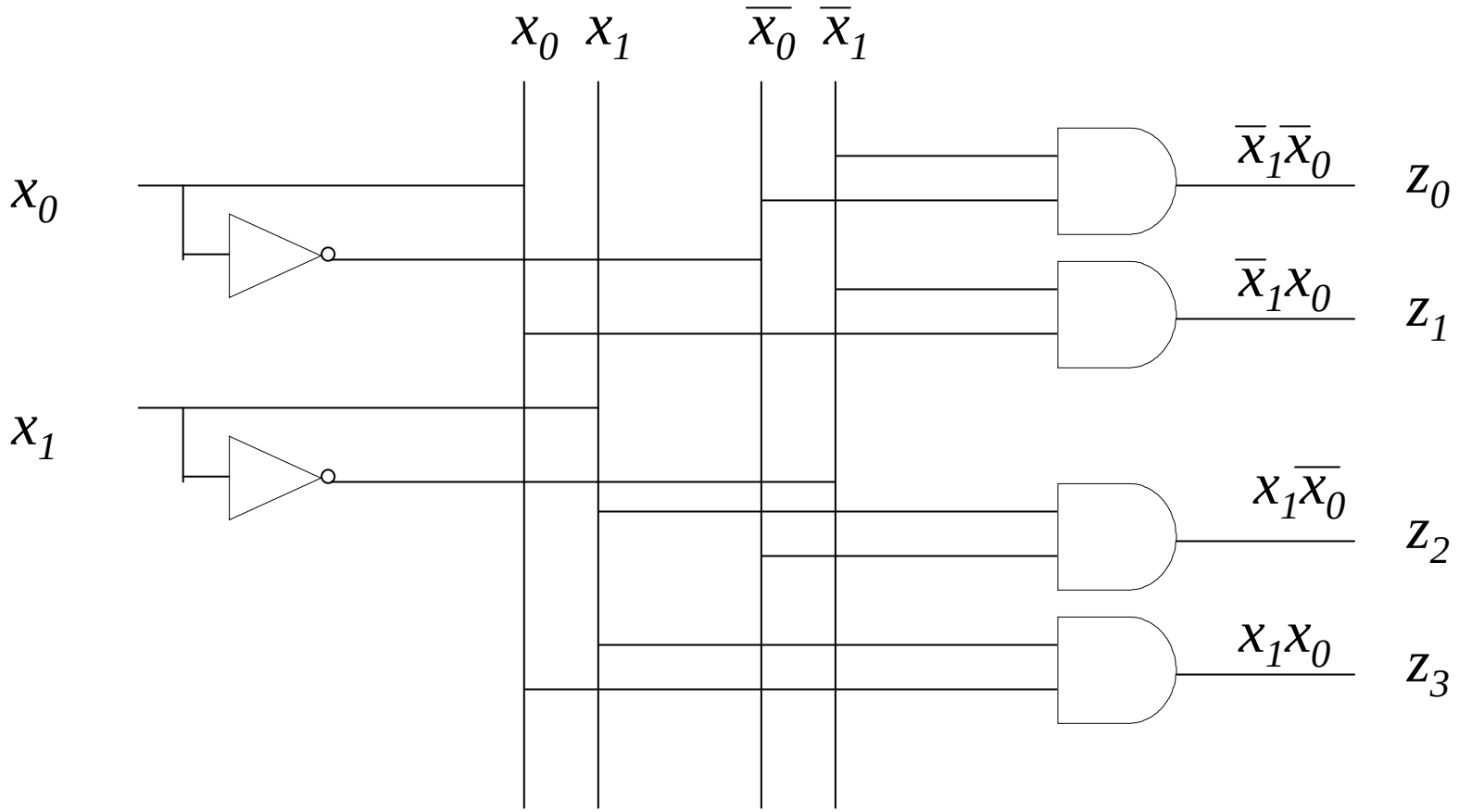
- Ingresso di *n* bit e 2^n uscite
- Per ciascuna combinazione degli ingressi, una sola uscita assume il valore 1 mentre le altre assumono valore 0



Se il valore dell'ingresso è *i* allora l'uscita z_i sarà vera e tutte le altre false

x_1	x_0	z_3	z_2	z_1	z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decodificatore 2-a-4



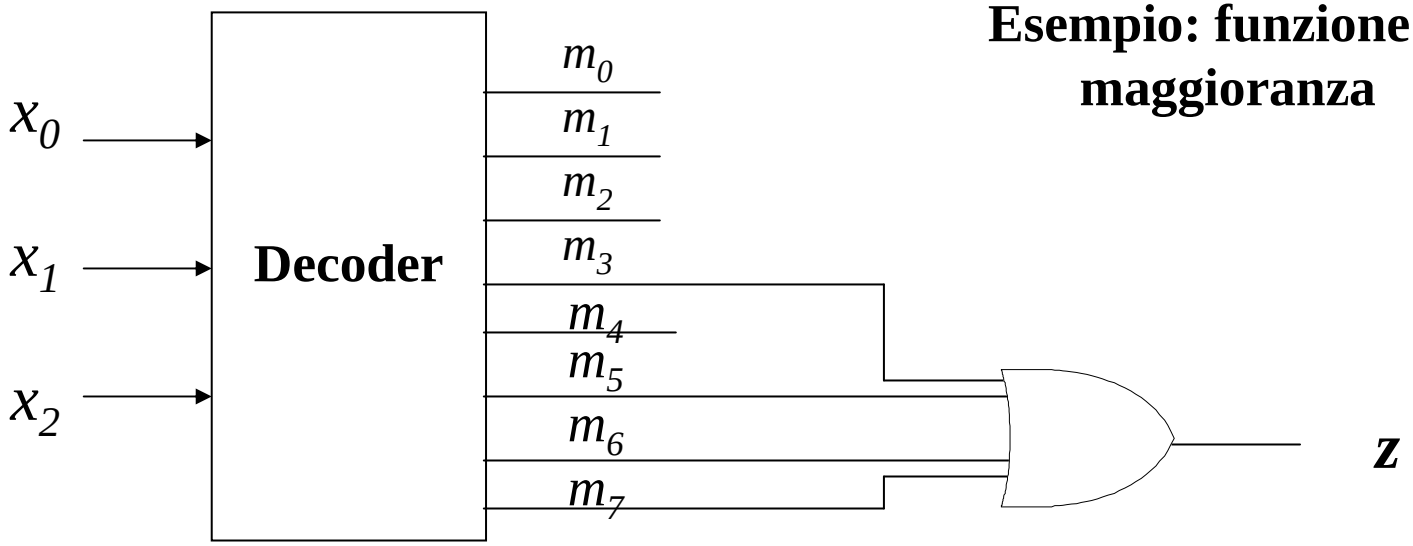
Decodificatori

- In pratica, un decodificatore fornisce alle sue uscite i mintermini degli ingressi

  $z_0 = m_0 \quad z_1 = m_1 \quad z_2 = m_2 \quad z_3 = m_3 \ldots$
- Quindi, una qualsiasi **forma canonica somma di prodotti** può essere realizzata utilizzando un **decodificatore** e una **porta OR**

Decodificatori

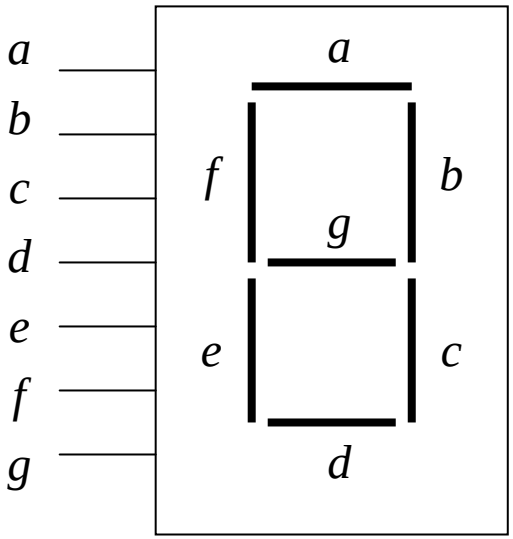
Forma canonica somma di prodotti



Esempio: funzione di maggioranza

$$z = m_3 + m_5 + m_6 + m_7 = \bar{x}_2 x_1 x_0 + x_2 \bar{x}_1 x_0 + x_2 x_1 \bar{x}_0 + x_2 x_1 x_0$$

Esempio di decodificatore Display a 7 segmenti



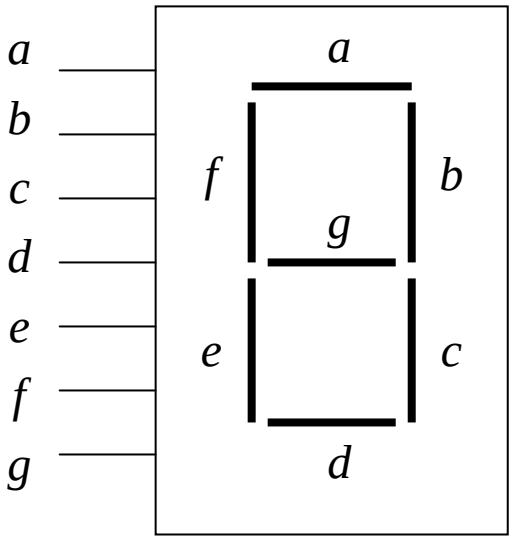
Display a 7 segmenti

- *Ogni segmento è alimentato in modo indipendente dagli altri*
- *Una cifra fra 0 e 9 può essere formata alimentando una parte dei segmenti*
- *La rete combinatoria che comanda il display ha:*

  Tanti ingressi quanti sono i bit del codice (4 nel ns. caso)

  7 uscite (che corrispondono ai segmenti)

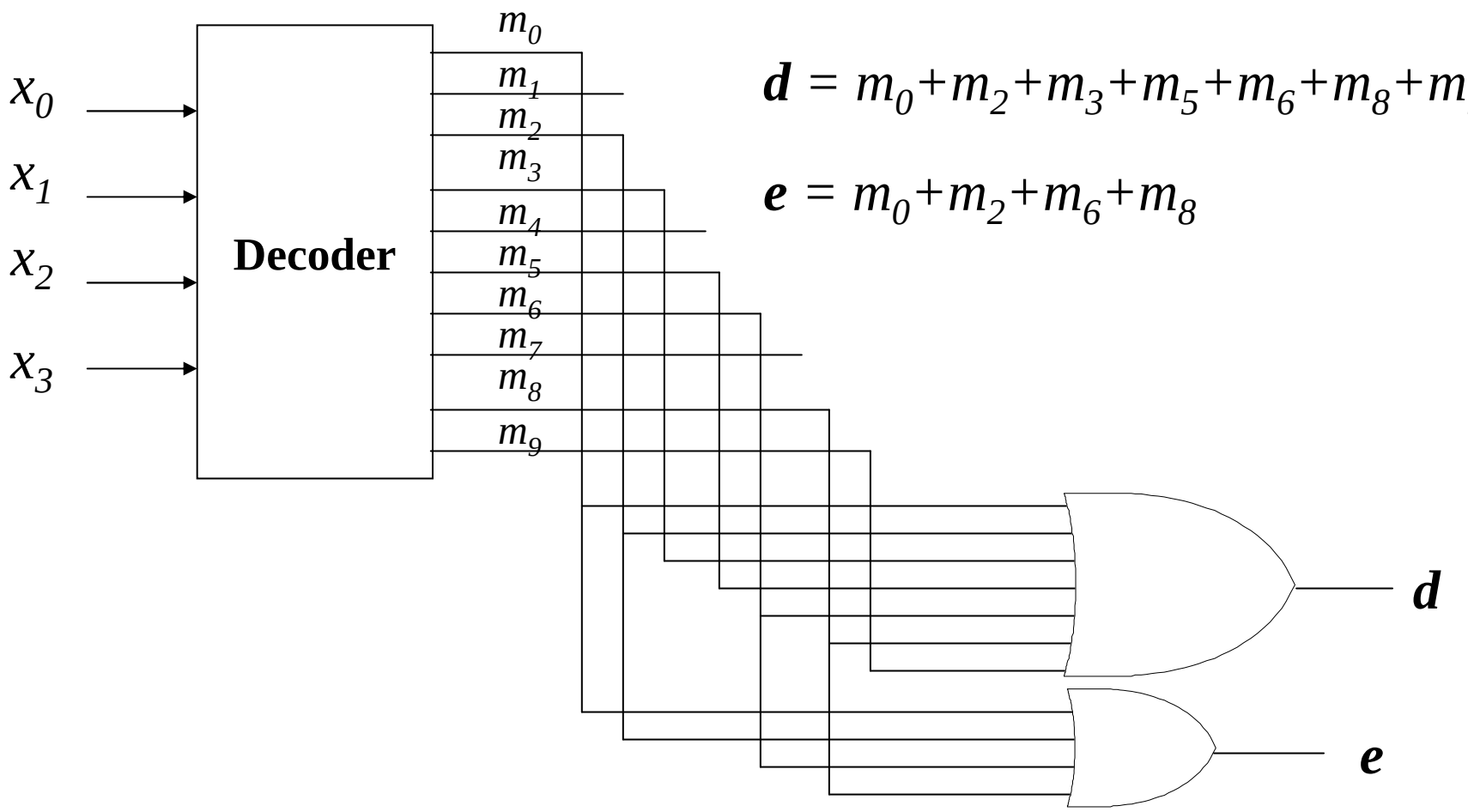
Display a 7 segmenti



Display a 7 segmenti

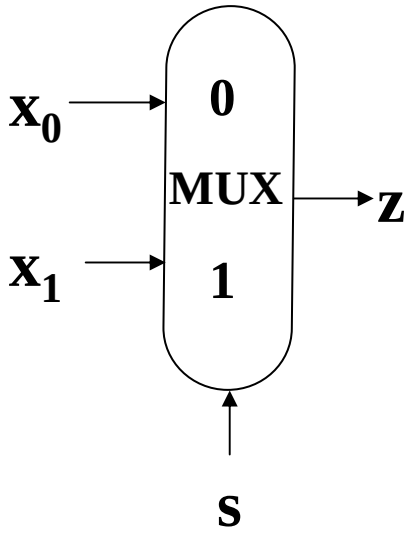
$x_3x_2x_1x_0$	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1

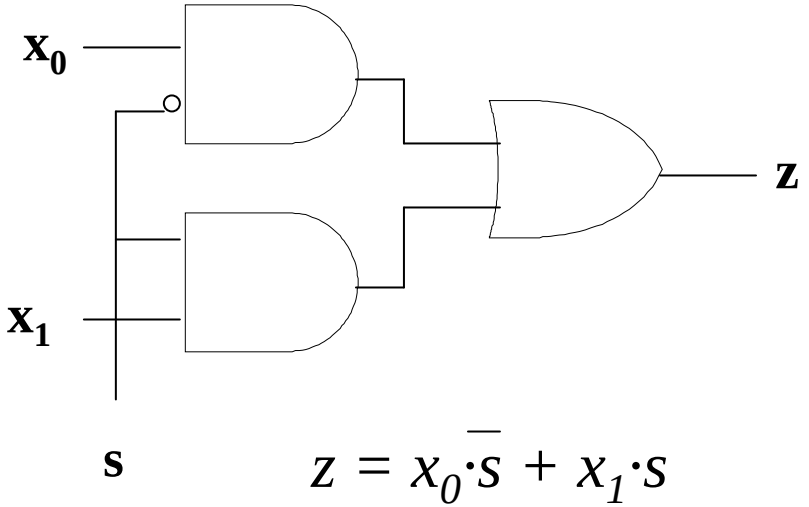
Display a 7 segmenti: esempio



Multiplexer

- Detto anche ***selettore***: la sua uscita è uguale a uno degli ingressi, scelto mediante un segnale di controllo (S)





$$z = x_0 \cdot \overline{s} + x_1 \cdot s$$

Es. Multiplexer a 2 vie

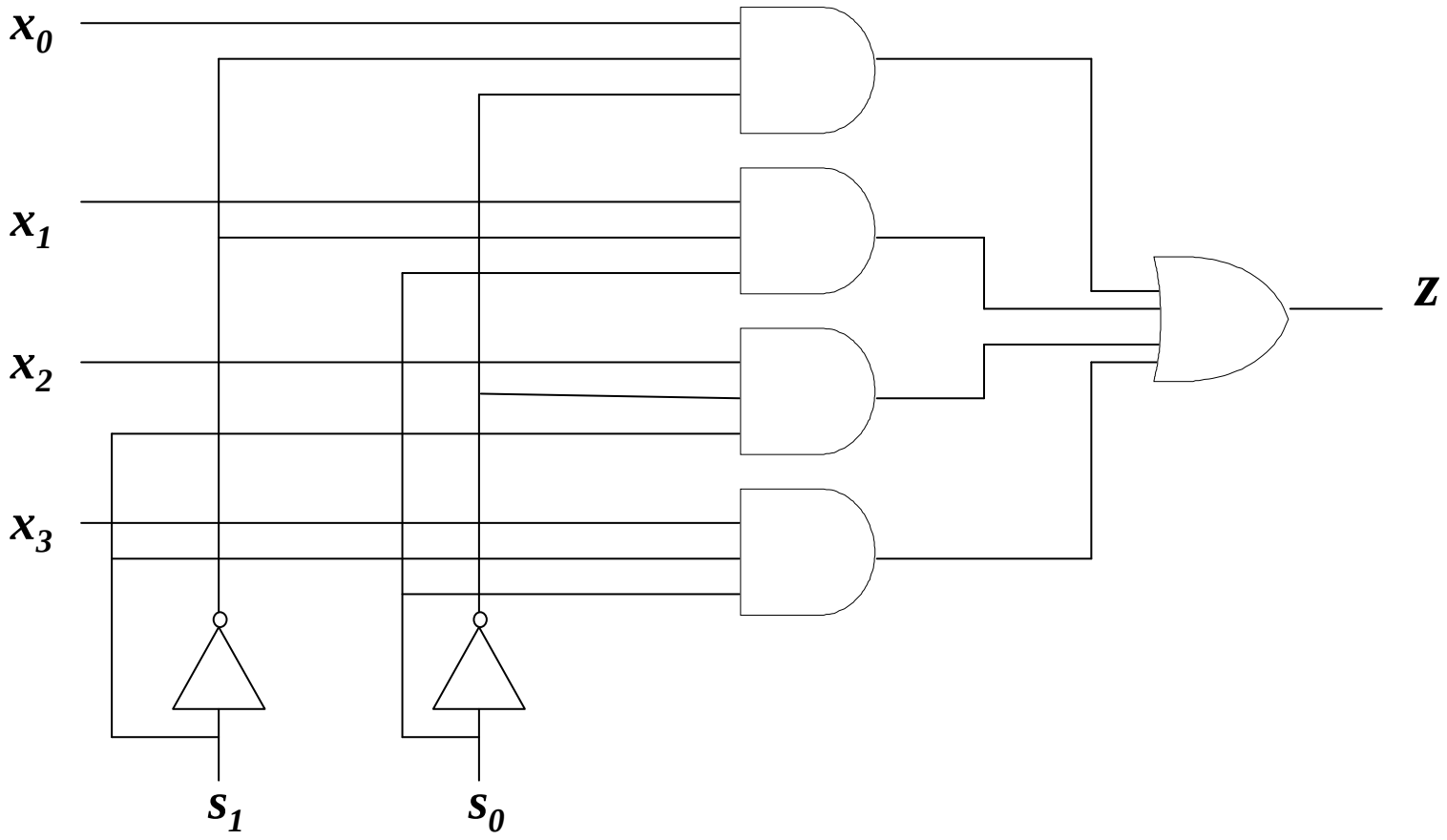
Esempio: multiplexer a 4 vie

- 4 **ingressi di dato**: x_0, x_1, x_2, x_3
- 2 **ingressi di selezione**, s_0 e s_1 $\Rightarrow$ 00 $\rightarrow$ x_0 , 01 $\rightarrow$ x_1, 10 $\rightarrow$ x_2, 11 $\rightarrow$ x_3

s_1	s_0	x_3	x_2	x_1	x_0	z
0	0	x	x	x	0	0
0	0	x	x	x	1	1
0	1	x	x	0	x	0
0	1	x	x	1	x	1
1	0	x	0	x	x	0
1	0	x	1	x	x	1
1	1	0	x	x	x	0
1	1	1	x	x	x	1

Ingressi di selezione ***Ingressi di dato*** ***uscita***

Esempio: multiplexer a 4 vie



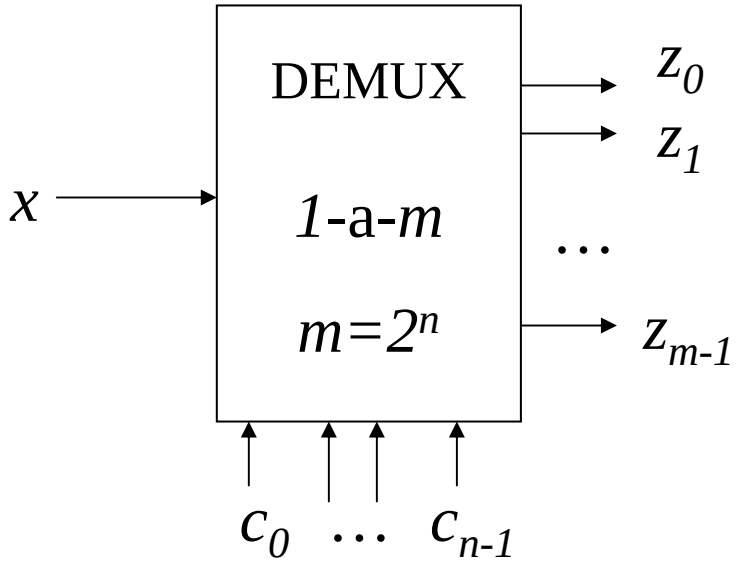
Applicazioni del Multiplexer

- Il multiplexer viene in genere usato per la **selezione di una linea di input** (con *n* linee di dato sono necessarie $\log_2(n)$ linee di selezione)
- Un'altra applicazione è come **convertitore di dati da parallelo a seriale**: avendo in input 8 bit di dati e avendo linee di controllo a 3 bit, queste ultime sono in grado di serializzare in output gli 8 bit instradandoli uno alla volta

Es. tastiera: ogni tasto premuto definisce un numero che viene spedito sulla linea in modo seriale.

Demultiplexer

- Si potrebbe chiamare ***distributore***
- Il suo compito è quello di instradare un singolo input in una delle 2^n linee di output a seconda del valore delle n linee di controllo.



Esempio: demultiplexer 1-a-4

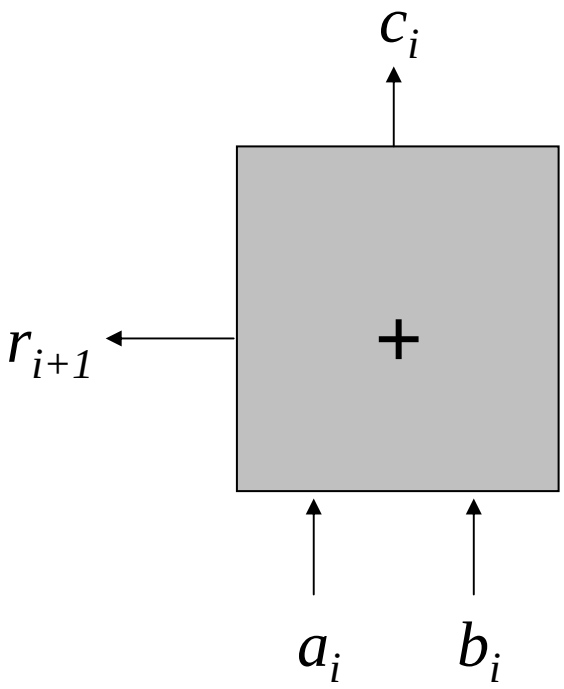
c_1	c_0	x	z_0	z_1	z_2	z_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Ingressi di controllo ***Ingressi di dato*** ***uscita***

$z_0 = \overline{c_1}\,\overline{c_0}x$ $z_1 = \overline{c_1}c_0x$ $z_2 = c_1\overline{c_0}x$ $z_3 = c_1c_0x$

Sommatore

Somma a 1 bit



Circuito half adder

a_i	b_i	c_i	r_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella di verità

Sommatore a *n* bit

- Vogliamo sommare bit a bit due numeri binari di *n* bit: $\boldsymbol{a_{n-1}\ldots a_i\ldots a_0}$ e $\boldsymbol{b_{n-1}\ldots b_i\ldots b_0}$
- Esempio:

$a_3\ a_2\ a_1\ a_0$

1 0 1 1 +

$b_3\ b_2\ b_1\ b_0$

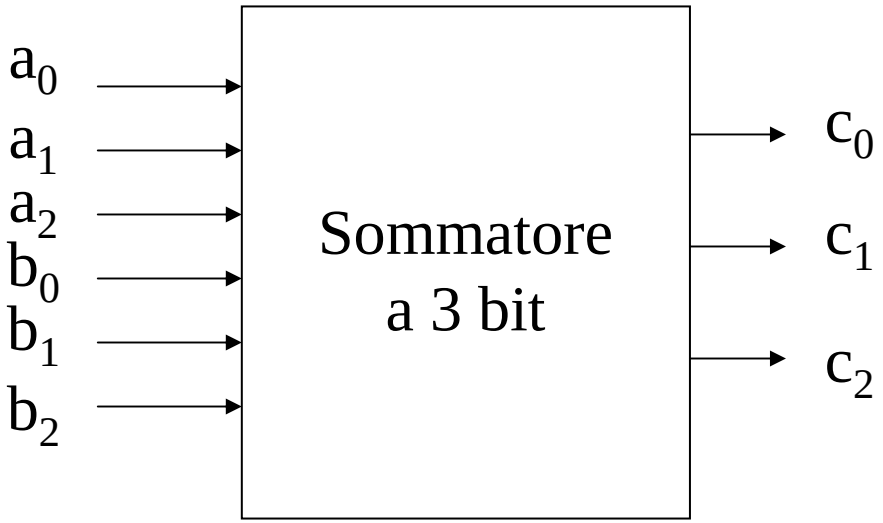
0 1 1 1 =

$c_3\ c_2\ c_1\ c_0$

(1) 0 0 1 0

Soluzione ad hoc

- Si potrebbe costruire un circuito digitale specializzato ad eseguire la somma di n bit
- Ad esempio, con $n = 3$

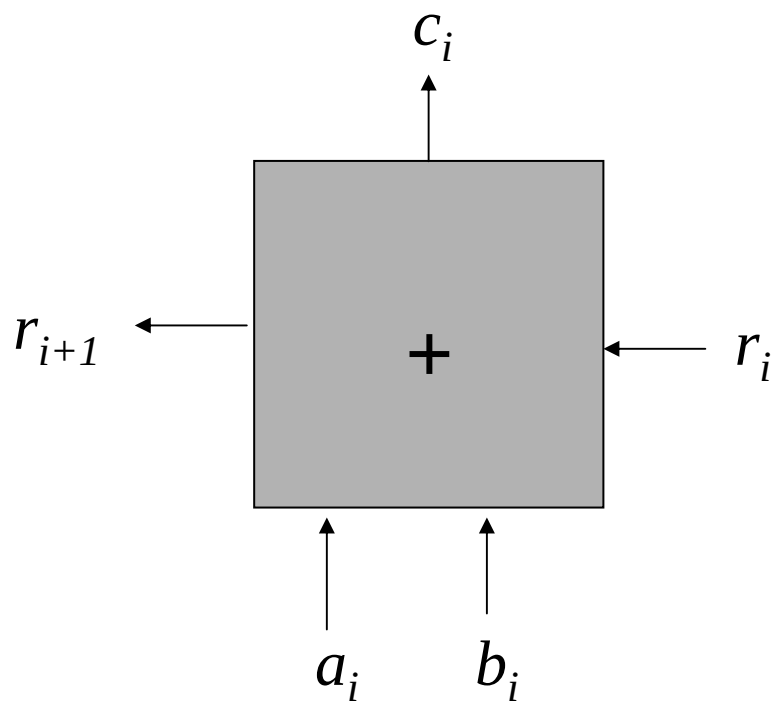


- Problema: se si cambia n occorre costruire un altro circuito

Tecnica di decomposizione di funzioni

- In pratica, si usa la tecnica di **decomposizione di funzioni** in funzioni più semplici
- Si introducono per questo nuove variabili booleane chiamate **“variabili di appoggio”**
- Le variabili di appoggio permettono di **connettere tra loro reti combinatorie** più semplici ottenute in seguito a una decomposizione
- Nel nostro caso le variabili di riporto r_i sono le variabili di appoggio utilizzate
- Non esistono metodi algoritmici che permettano in generale di individuare funzioni semplici e variabili di appoggio

Sommatore di tipo Full Adder



Circuito sommatore ad *n* bit

In pratica: per ogni *i* da *0* a *n-1*

- si sommano a_i e b_i e il bit di riporto r_i dalla posizione precedente
- si generano il bit c_i di somma e il bit r_{i+1} di riporto per le cifre successive

Sommatore ad *n* bit con *n* full adder

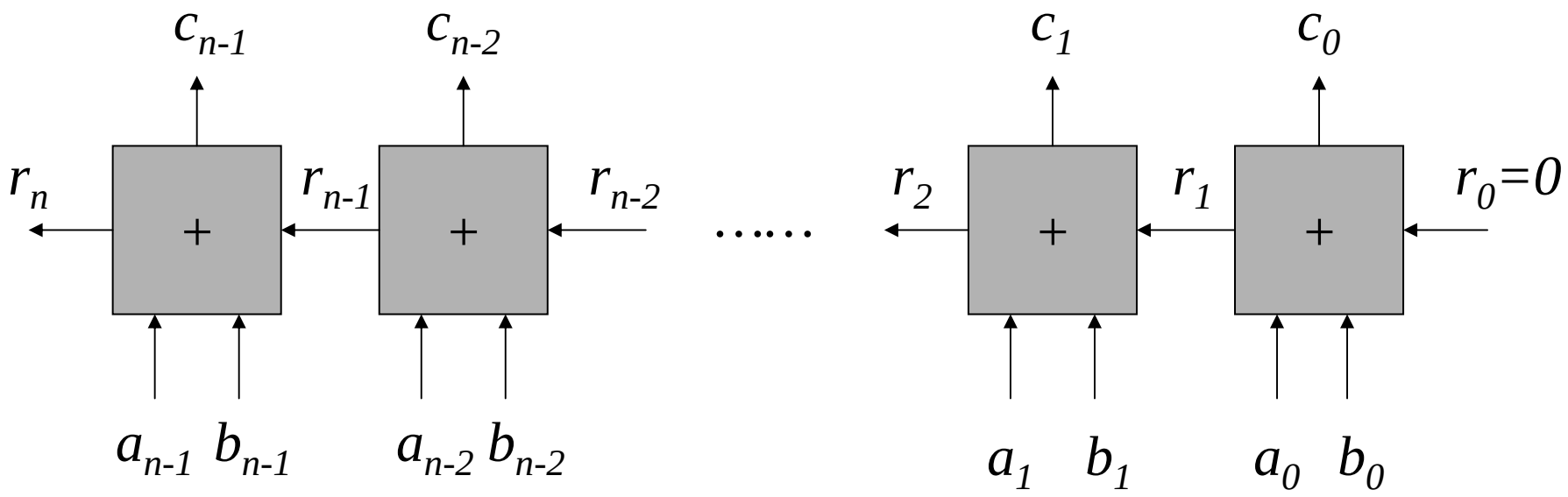


Tabella di verità per il full adder

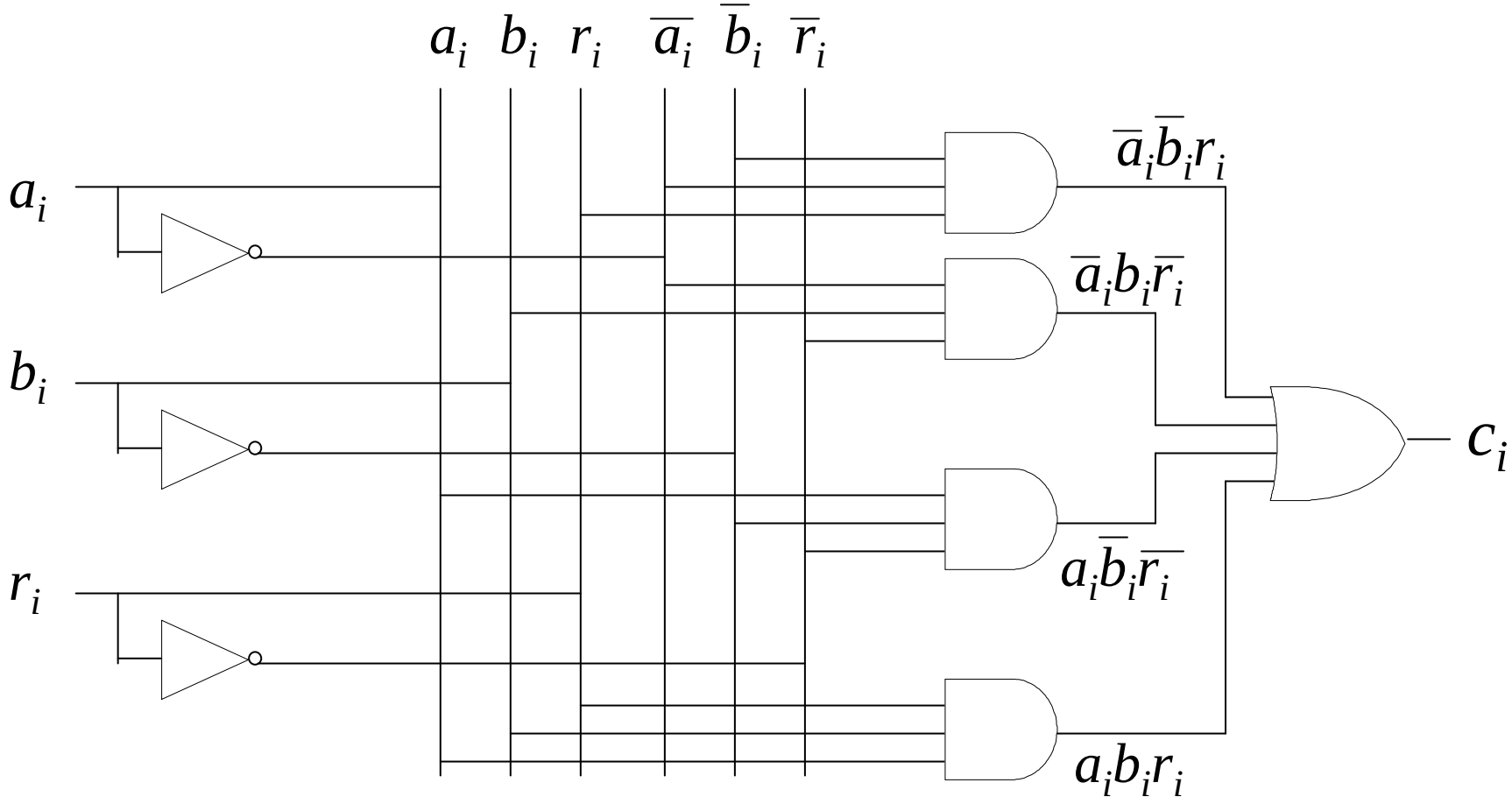
a_i	b_i	r_i	c_i	r_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\boldsymbol{c_i} = \overline{a}_i\overline{b}_i r_i + \overline{a}_i b_i \overline{r}_i + a_i\overline{b}_i\overline{r}_i + a_i b_i r_i$$

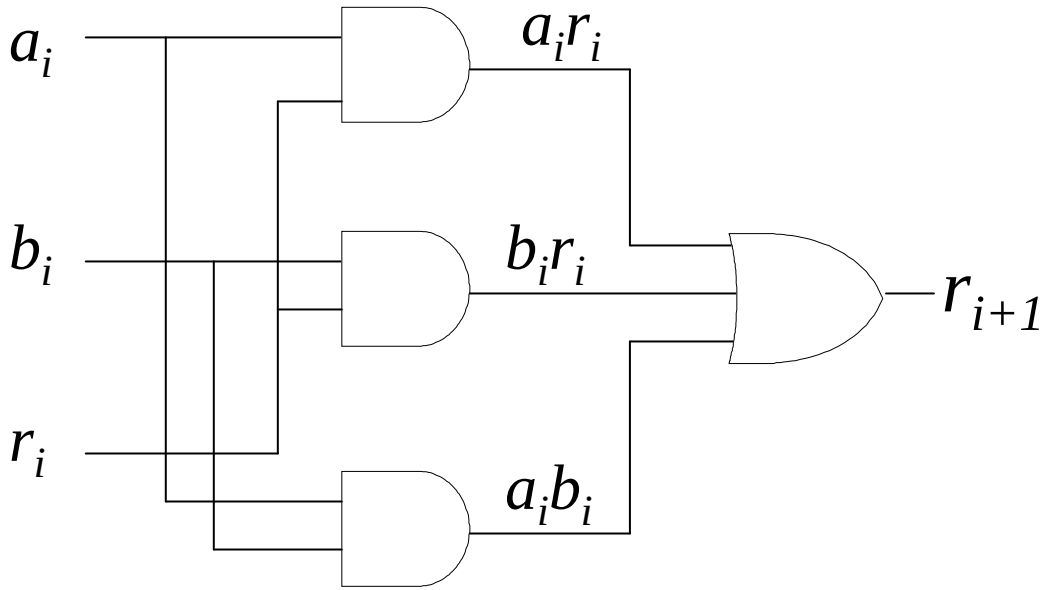
$$\boldsymbol{r_{i+1}} = \overline{a}_i b_i r_i + a_i\overline{b}_i r_i + a_i b_i\overline{r}_i + a_i b_i r_i =$$
$$\overline{a}_i b_i r_i + a_i\overline{b}_i r_i + a_i b_i\overline{r}_i + a_i b_i r_i + \boldsymbol{a_i b_i r_i}$$
$$+ \boldsymbol{a_i b_i r_i} =$$

$$= b_i r_i(\overline{a}_i + a_i) + a_i r_i(\overline{b}_i + b_i) + a_i b_i \cdot$$
$$(\overline{r}_i + r_i) = \boldsymbol{b_i r_i + a_i r_i + a_i b_i}$$

Circuito per la generazione di c_i



Circuito per la generazione di r_{i+1}



$$\boldsymbol{r_{i+1} = b_i r_i + a_i r_i + a_i b_i}$$

Sottrattore ad *n* bit

- Variabile di appoggio p_i (“prestito”)

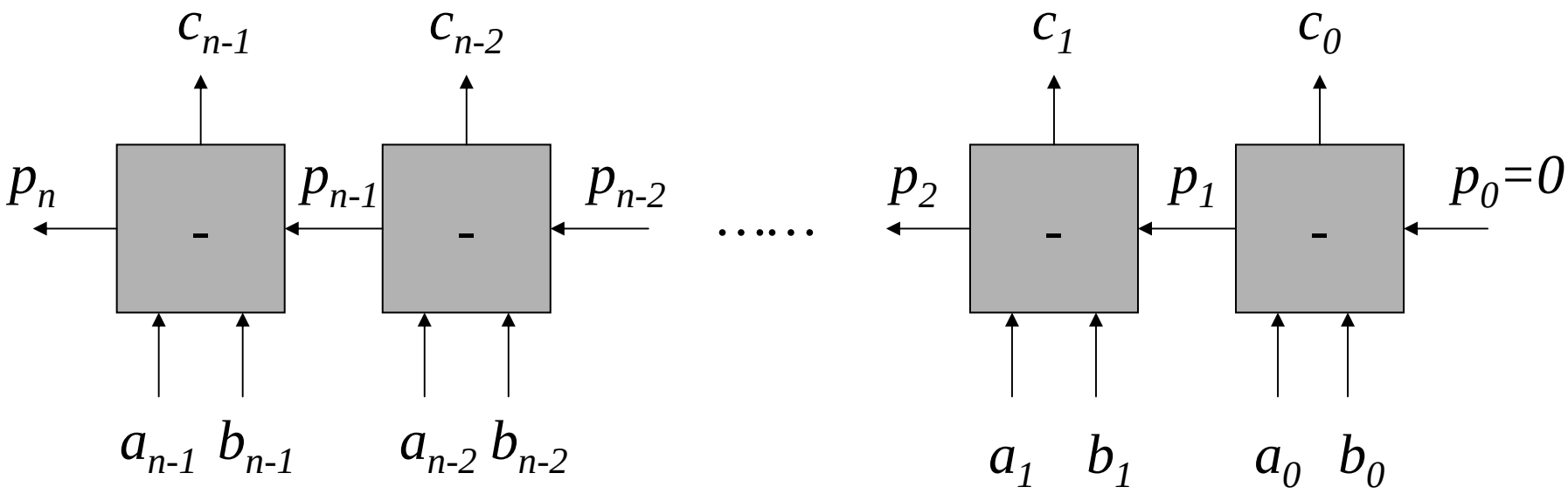


Tabella di verità per il sottrattore

a_i	b_i	p_i	c_i	p_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Esempio

$a_3\, a_2\, a_1\, a_0$

1 1 0 0 -

$b_3\, b_2\, b_1\, b_0$

0 0 1 1 =

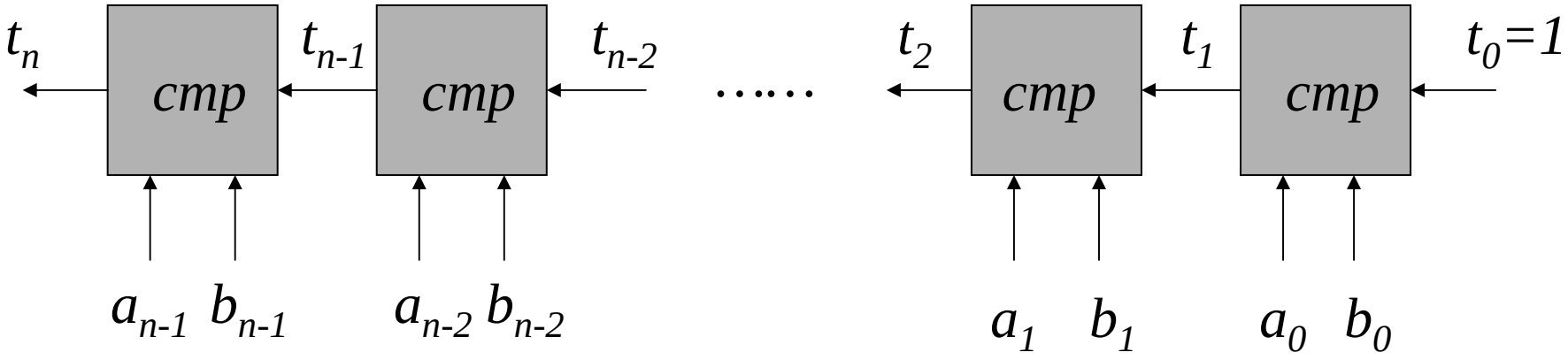
$c_3\, c_2\, c_1\, c_0$

1 0 0 1

Comparatore ad *n* bit

a ≥ b ?

- Variabile di appoggio t_i (test) che vale 1 se i primi *i* bit di *a* rappresentano un numero maggiore o uguale ai primi *i* bit di *b*
- Si dà la tabella di verità per la variabile di uscita t_{i+1}



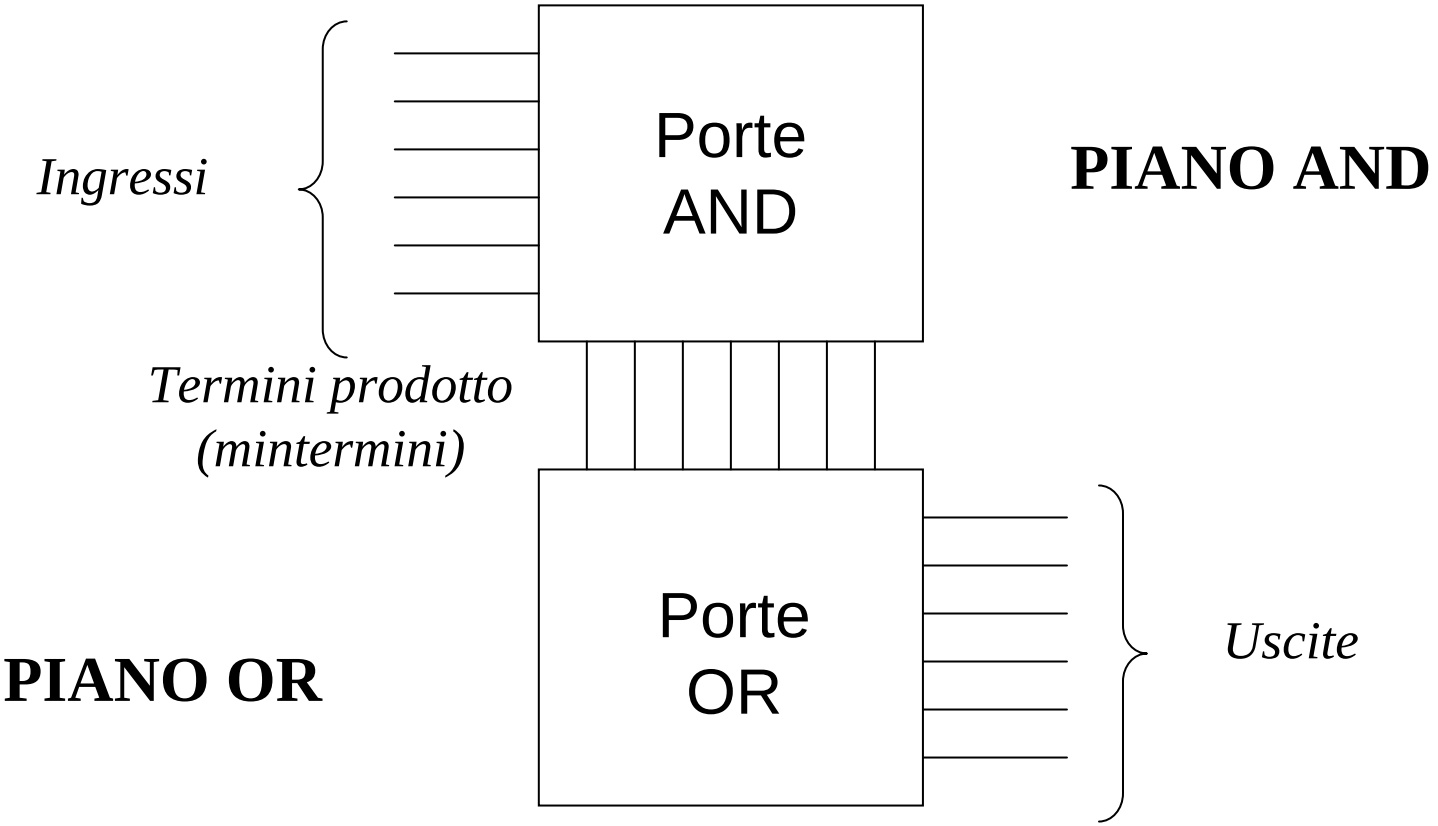
Circuito risultante

t_n = 1 se e solo se a ≥ b

Logiche strutturate

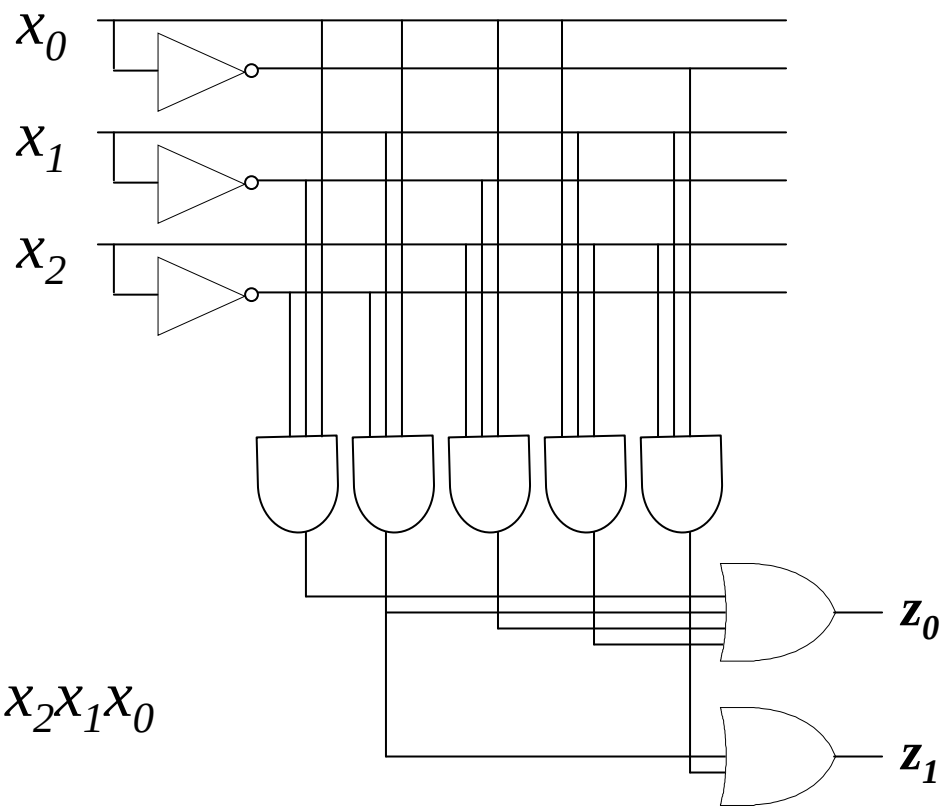
- Un **insieme di funzioni logiche** è rappresentabile da una **tabella di verità con più colonne di uscita**
- La tecnologia di implementazione detta ***Programmable Logic Array (PLA)*** permette di realizzare direttamente la tabella di verità di un insieme di funzioni logiche

PLA
Programmable Logic Array



Esempio di PLA

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

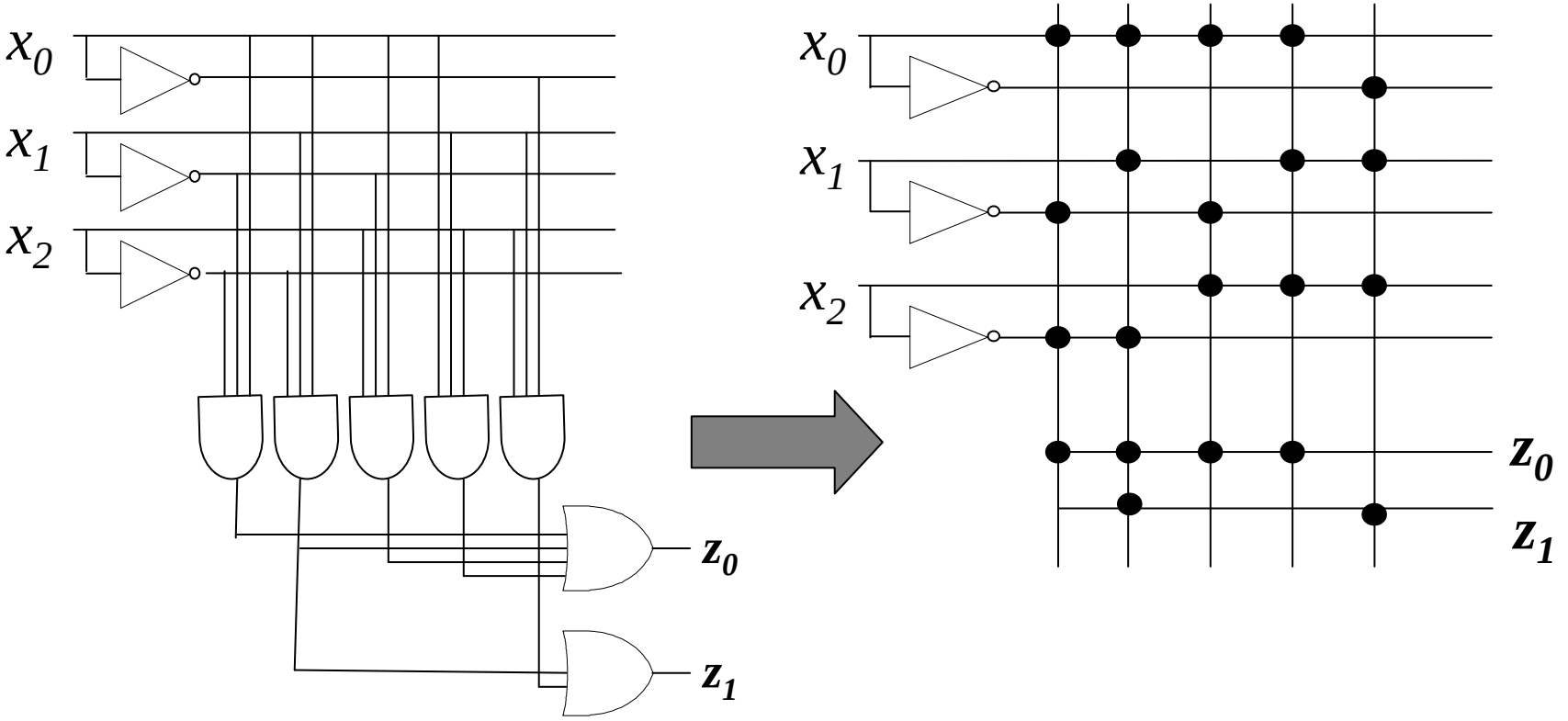


$$z_0 = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1x_0 + x_2\overline{x}_1x_0 + x_2x_1x_0$$

$$z_1 = \overline{x}_2x_1x_0 + x_2x_1\overline{x}_0$$

- 5 AND per i mintermini necessari
- 2 OR perché 2 sono le uscite

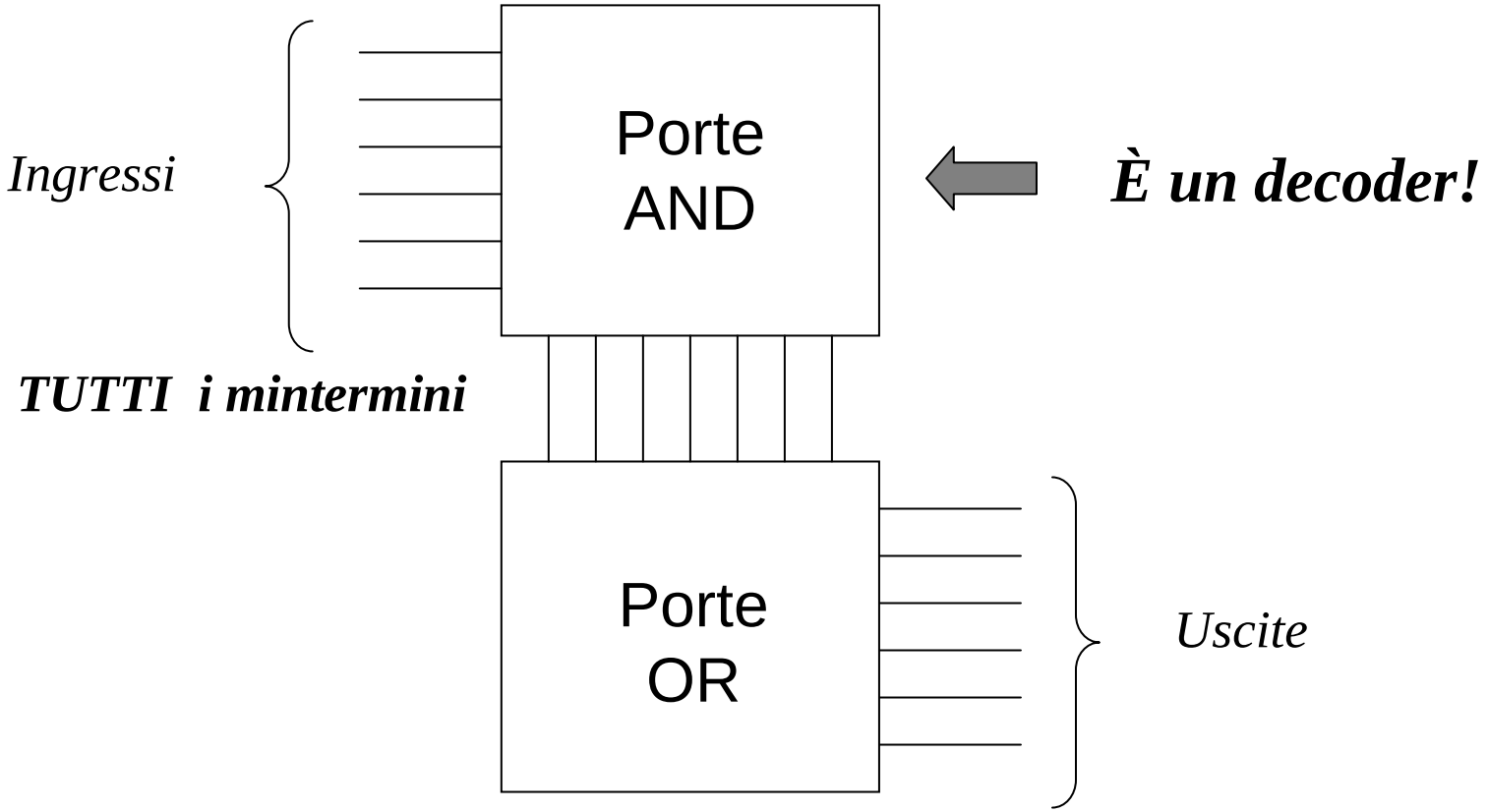
Un altro tipo di rappresentazione per le PLA



ROM (Read Only Memory)

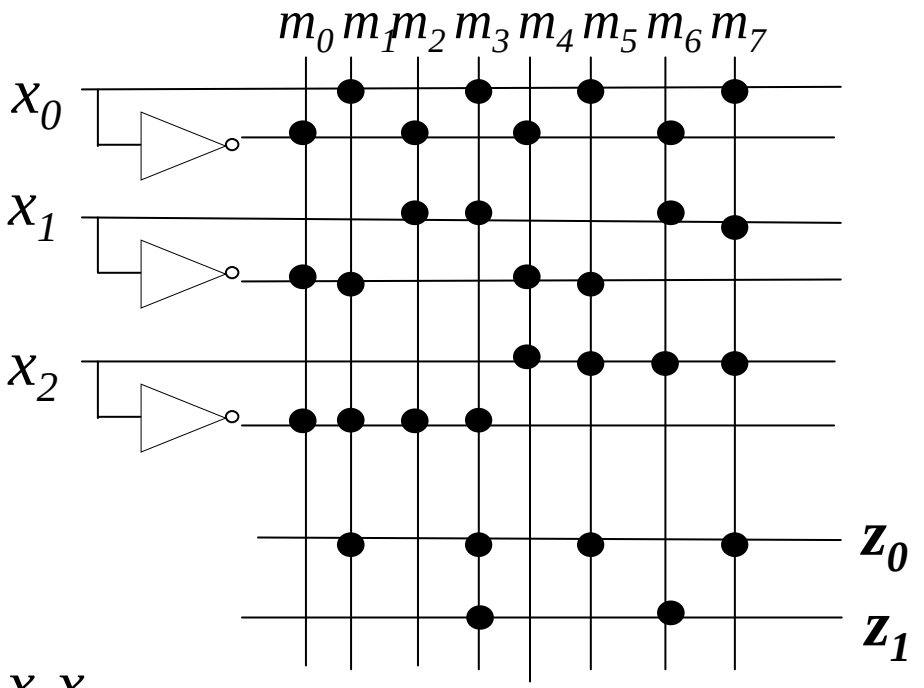
- Una ROM è un altro tipo di logica strutturata
- Ha un insieme di locazioni il cui contenuto è fisso
- I valori degli **ingressi** possono essere visti come **indirizzi**, mentre i valori delle **uscite**, in corrispondenza di certi valori degli ingressi, corrispondono al **contenuto di una locazione**
- Una ROM codifica *m* funzioni ad *n* ingressi: *n* linee di indirizzo con elementi ampi *m* bit ciascuno
- Rispetto alla PLA, è un dispositivo **completamente decodificato**

ROM (Read Only Memory)



ROM

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1



$$z_0 = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1x_0 + x_2\overline{x}_1x_0 + x_2x_1x_0$$

$$z_1 = \overline{x}_2x_1x_0 + x_2x_1\overline{x}_0$$

Esempio:
una PLA per un transcodificatore

Codifica BCD o “8421”

Cifra decimale	Codifica binaria			
	p=8	***4***	***2***	***1***
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Codifica “2421”

Cifra decimale	Codifica binaria			
	p=2	***4***	***2***	***1***
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Esempio:
una PLA per un transcodificatore

- Un transcodificatore da codice “8421” a codice “2421” è una rete combinatoria con 4 ingressi e 4 uscite
- Quando in ingresso viene applicata una configurazione di valori che codifica una cifra decimale in codice “8421”, deve dare alle uscite la codifica della stessa cifra in codice “2421”
- Costruiamo la tabella di verità, le espressioni logiche corrispondenti alle uscite e la PLA che realizza il transcodificatore

Tabella di verità per il transcodificatore

x_8	x_4	x_2	x_1	a	b	c	d
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

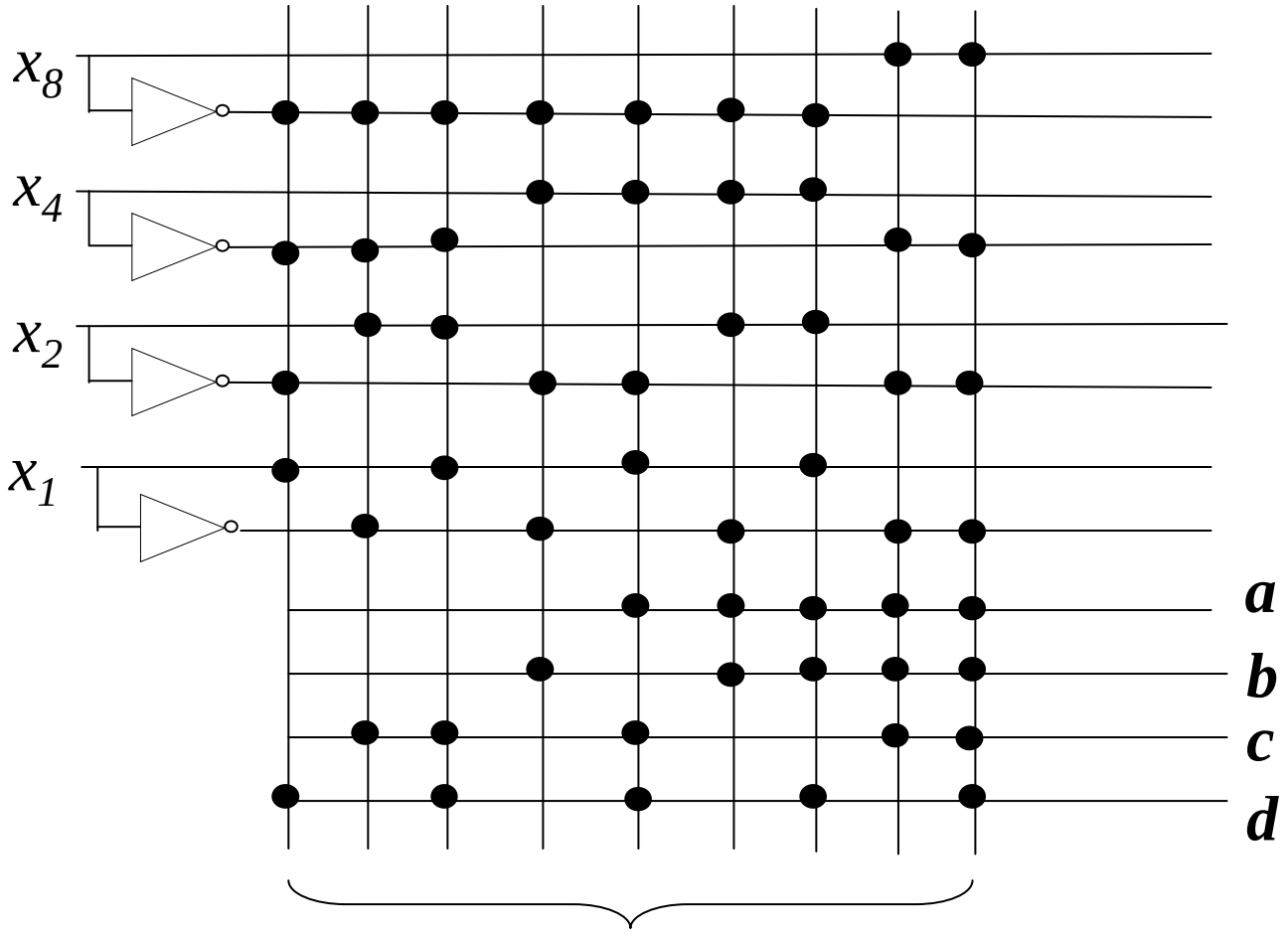
$$a = \overline{x}_8 x_4 \overline{x}_2 x_1 + \overline{x}_8 x_4 x_2 \overline{x}_1 + \overline{x}_8 x_4 x_2 x_1 + x_8 \overline{x}_4 \overline{x}_2 \overline{x}_1 + x_8 \overline{x}_4 \overline{x}_2 x_1$$

$$b = \overline{x}_8 x_4 \overline{x}_2 \overline{x}_1 + \overline{x}_8 x_4 x_2 \overline{x}_1 + \overline{x}_8 x_4 x_2 x_1 + x_8 \overline{x}_4 \overline{x}_2 \overline{x}_1 + x_8 \overline{x}_4 \overline{x}_2 x_1$$

$$c = \overline{x}_8 \overline{x}_4 x_2 \overline{x}_1 + \overline{x}_8 \overline{x}_4 x_2 x_1 + \overline{x}_8 x_4 \overline{x}_2 x_1 + x_8 \overline{x}_4 \overline{x}_2 \overline{x}_1 + x_8 \overline{x}_4 \overline{x}_2 x_1$$

$$d = \overline{x}_8 \overline{x}_4 \overline{x}_2 x_1 + \overline{x}_8 \overline{x}_4 x_2 x_1 + \overline{x}_8 x_4 \overline{x}_2 x_1 + \overline{x}_8 x_4 x_2 x_1 + x_8 \overline{x}_4 \overline{x}_2 x_1$$

PLA per il transcodificatore



Minimizzazione

- Il **problema** della minimizzazione consiste nel derivare l'**espressione minima** corrispondente a una data funzione
- **Espressione minima** rispetto a cosa? Dipende anche dalla tecnologia…
- **Perché minimizzare?** In genere, per realizzare il circuito corrispondente a una data funzione con il numero minimo di elementi di calcolo
- Esistono **tecniche** particolari **di minimizzazione** (es. Mappe di Karnaugh)
- Tool automatici, strumenti CAD

Mappe di Karnaugh

Per minimizzare somme di prodotti tipicamente si applicano riduzioni del tipo:

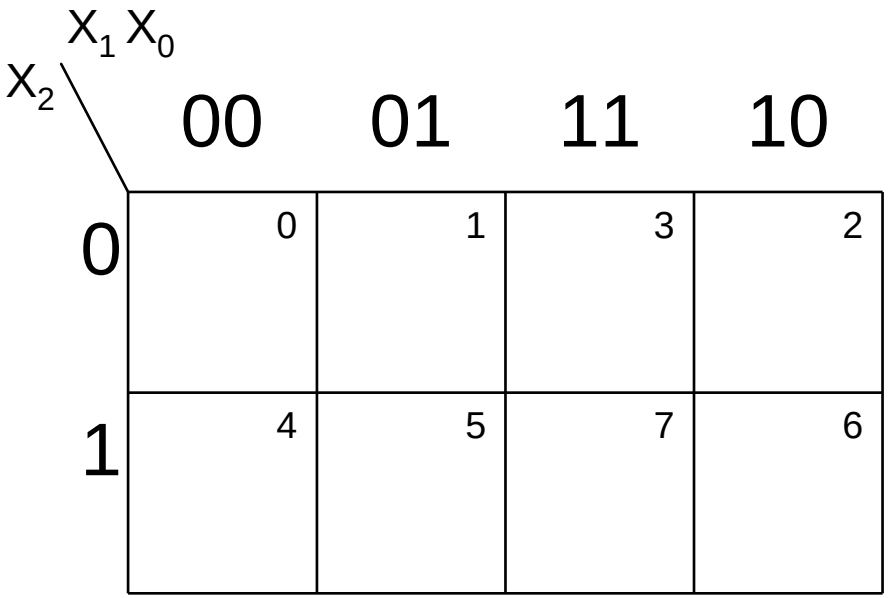
$xE+\bar{x}E=(x+\bar{x})E=E$

Dove E è un prodotto delle rimanenti variabili, una volta estratto x.

Applicare questa ed altre proprietà senza un metodo rischia di non condurre ad una forma minima, dato che le riduzioni spesso dipendono dall'ordine con cui sono applicate le proprietà.

La tecnica delle mappe di Karnaugh ci viene in aiuto tramite una rappresentazione geometrica delle configurazioni binarie.

Mappe di Karnaugh a 3 variabili



I valori delle variabili sono assegnati in modo che le celle della mappa siano adiacenti, nel senso che spostandoci da una cella all'altra in orizzontale o verticale varia una sola delle variabili considerate (coordinate).

Ad esempio: la cella 2 è adiacente alla 3, alla 6, ma anche alla 0.

Mappe di Karnaugh a 4 variabili

$X_3 X_2$ \ $X_1 X_0$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Mappe di Karnaugh

Sottocubi e implicanti

Si trasferiscono i valori di una tabella di verità in una mappa di Karnaugh.

2^k celle adiacenti contenenti gli 1, ovvero quelle in cui abbiamo trascritto i mintermini della funzione da minimizzare, vengono definite **sottocubo di ordine k**.

Ad ogni sottocubo di ordine k corrisponde un **implicante**, ovvero il prodotto delle n-k variabili che mantengono lo stesso valore in tutte le 2^k coordinate, prese in forma normale se nelle coordinate hanno valore 1, in forma complementata se hanno valore 0.

Mappe di Karnaugh

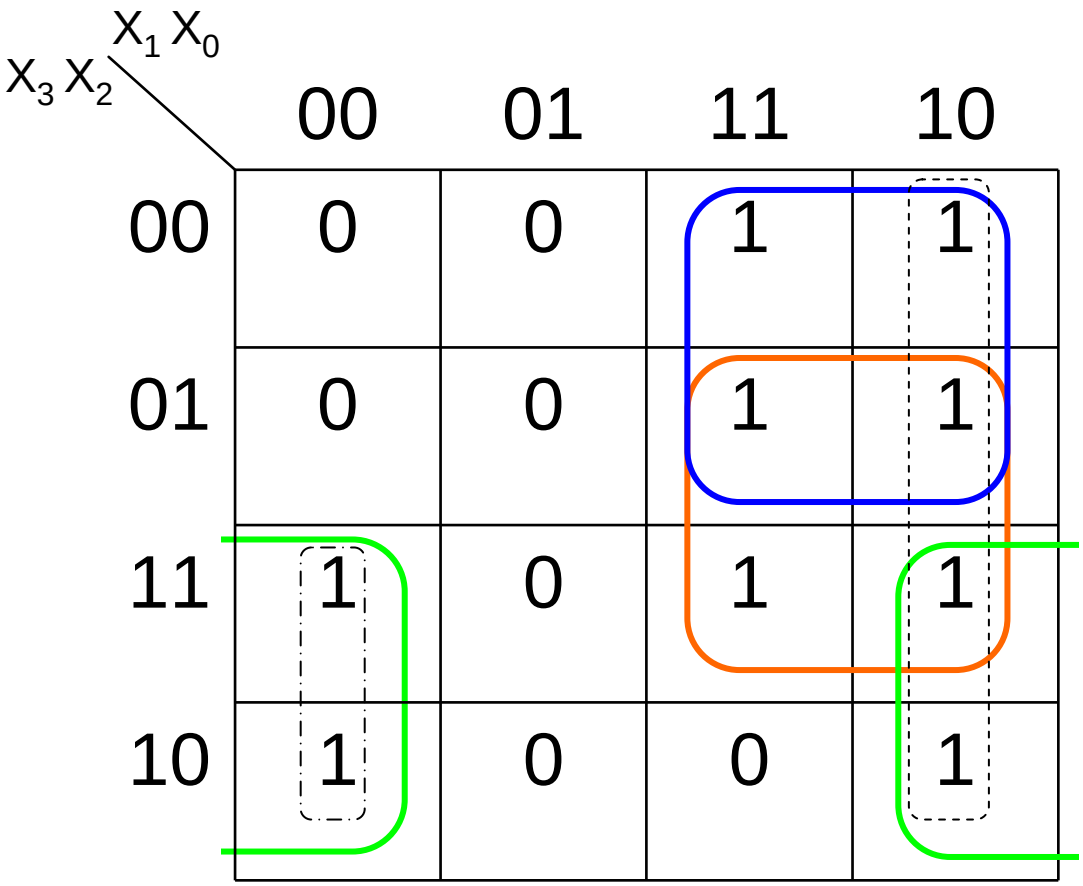
Minimizzazione

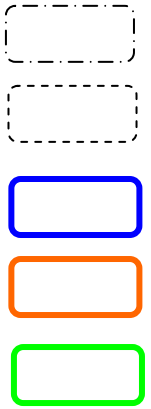
Si individua il minimo numero di sottocubi di massime dimensioni che coprano tutti gli 1 della funzione. Tali sottocubi non possono quindi essere contenuti in altri sottocubi; il corrispondente termine prodotto è chiamato **implicante primo**.

La forma minima della funzione del tipo somma di prodotti si ottiene sommando gli implicanti primi **essenziali**, ovvero gli implicanti primi che sono gli unici a coprire almeno un mintermine della funzione.

Mappe di Karnaugh

Esempi





Implicante non primo

Implicante non essenziale

Implicante primo $\overline{x}_3\,x_1$

Implicante primo $x_2\,x_1$

Implicante primo $x_3\,\overline{x}_0$

$$F = \overline{x}_3\,x_1 + x_2\,x_1 + x_3\,\overline{x}_0$$

Mappe di Karnaugh

Condizioni di indifferenza

Data una funzione non completamente specificata, si sfruttano le condizioni di indifferenza, in un'ottica di minimizzazione, per **ridurre il numero dei sottocubi** o per **creare sottocubi di dimensioni maggiori** con gli 1 già presenti nella mappa.

Nel primo caso le condizioni di indifferenza saranno poste a 0 nel secondo a 1.

Tabella di verità per il transcodificatore

x_8	x_4	x_2	x_1	a	b	c	d
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

ESPRESSIONI MINIMIZZATE

$$a = x_8 + x_4x_2 + x_4\overline{x_2}x_1$$

$$b = x_8 + x_4x_2 + x_4\overline{x_1}$$

$$c = x_8 + x_4\overline{x_2}x_1 + \overline{x_4}x_2$$

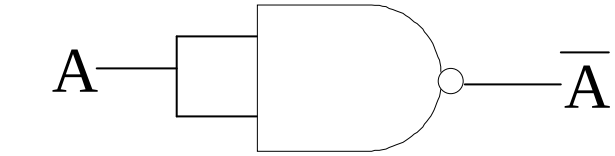
$$d = x_1$$

Esercizio: disegnare il circuito

Porte NAND e NOR

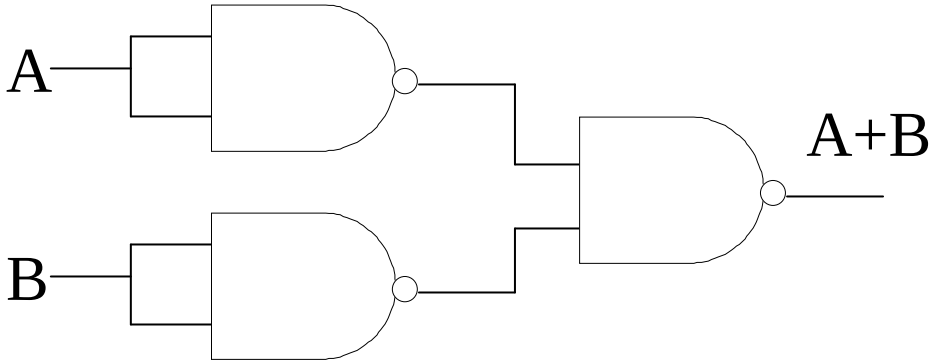
- Si può dimostrare che **tutte** le funzioni logiche sono realizzabili a partire da **un solo tipo di porta logica**, a patto che questa comprenda un'**inversione**
- **NAND** (porta AND con uscita invertita) e **NOR** (porta OR con uscita invertita) sono le due porte invertenti più diffuse
- NAND e NOR sono dette ***universali***

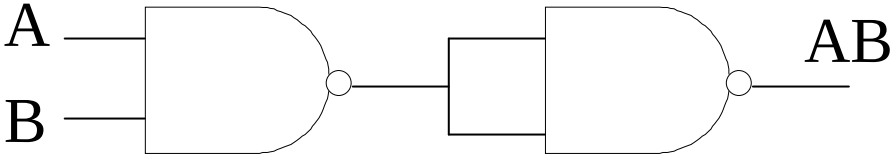
Porte NOT, AND, OR usando solo porte NAND



NOT

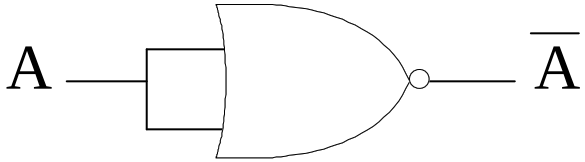
OR





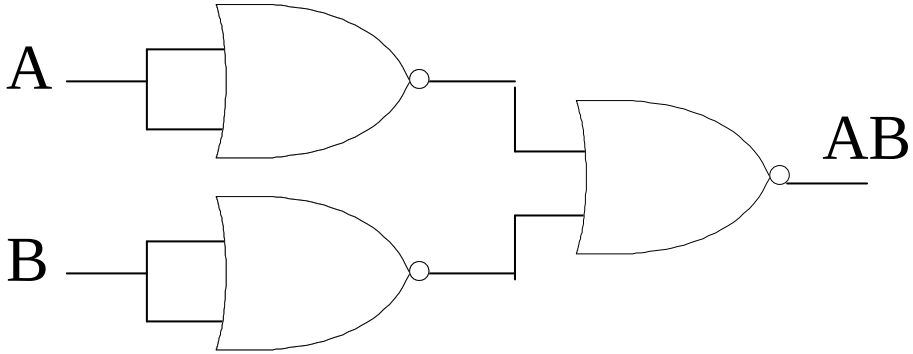
AND

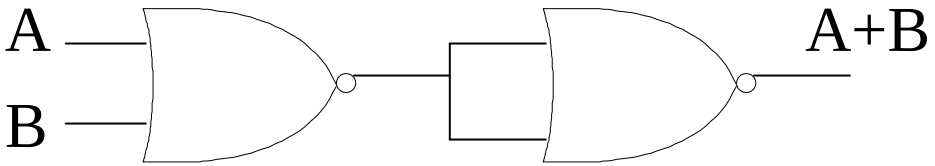
Porte NOT, AND, OR usando solo porte NOR



NOT

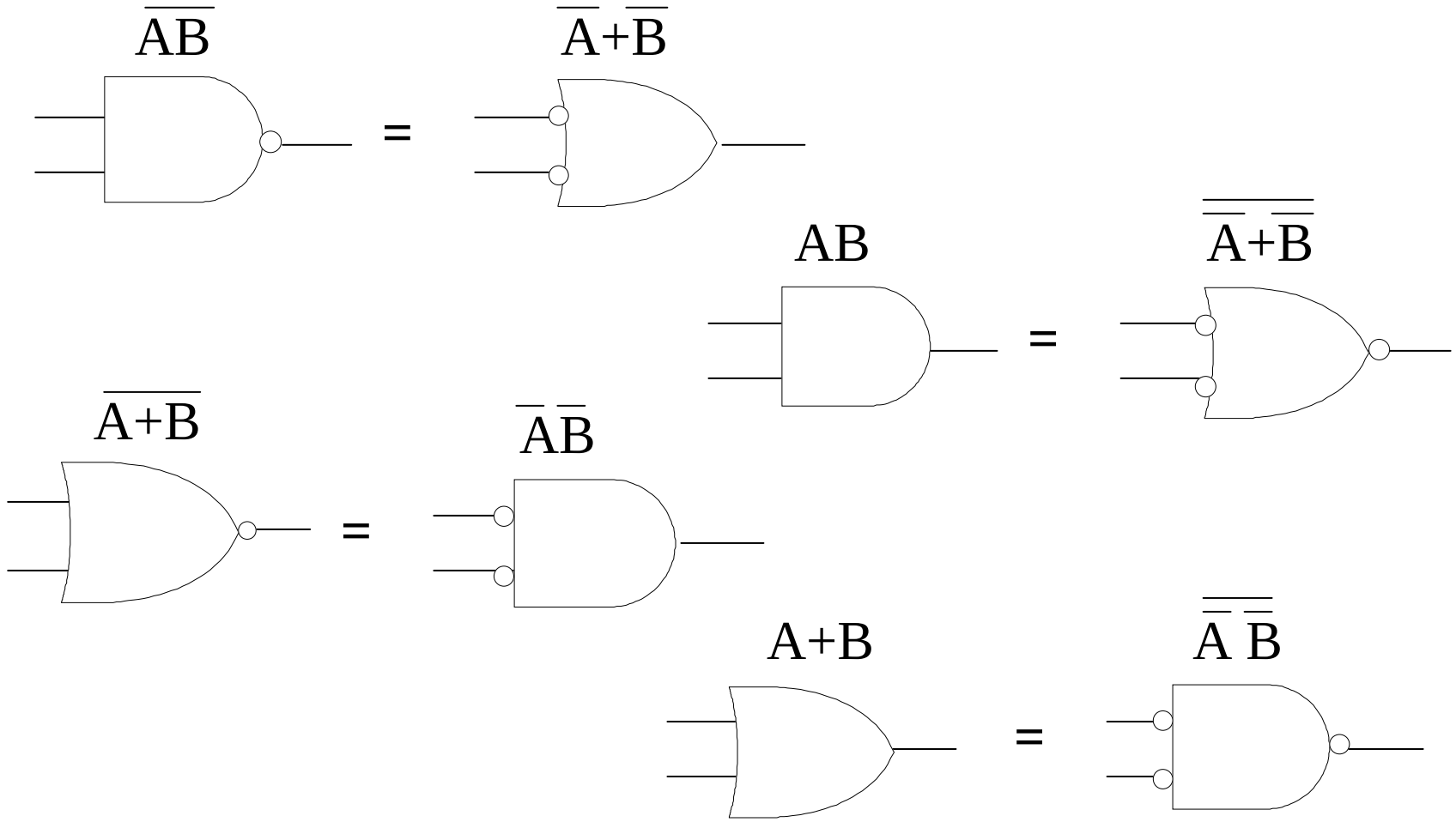
AND





OR

Notazione alternativa



Es: trasformare l'espressione seguente in modo da utilizzare solo porte NAND

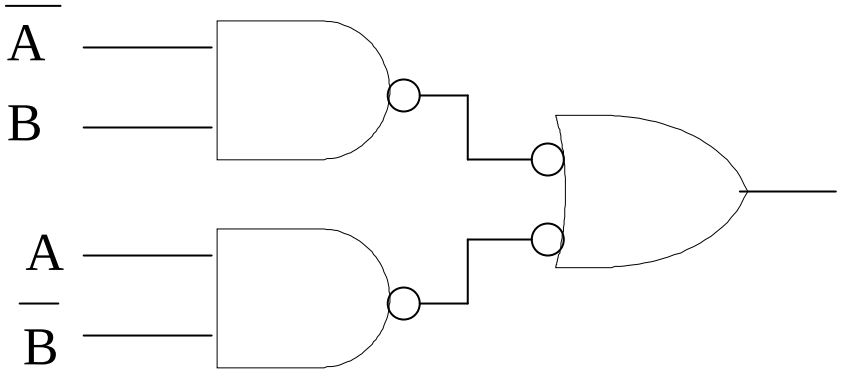
$$\bar{x}yz + x\bar{y}z + \bar{w}$$

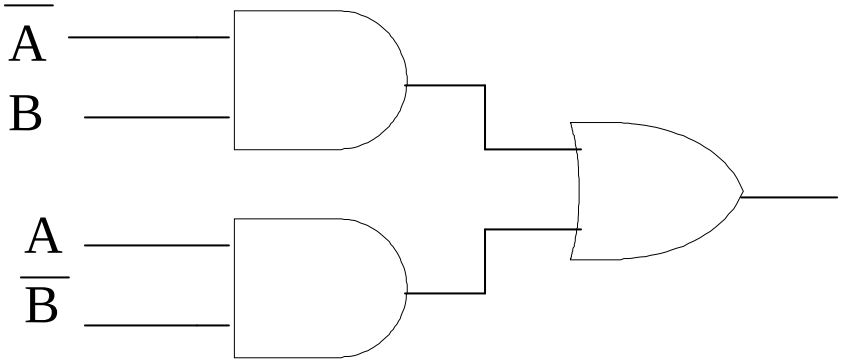
Con due negazioni e applicando il teorema di De Morgan si ha:

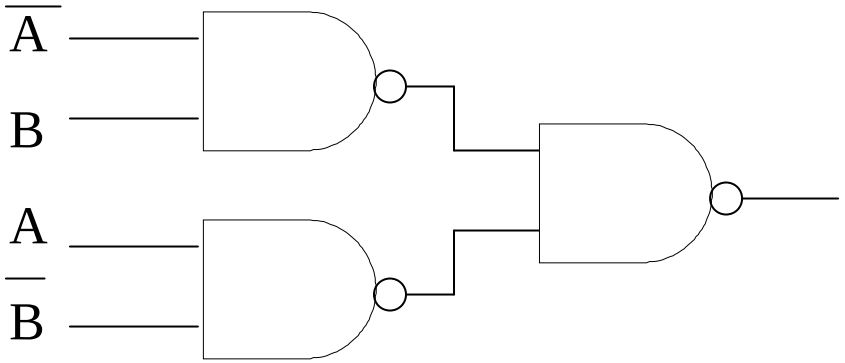
$$\overline{\overline{\bar{x}yz + x\bar{y}z + \bar{w}}} = \overline{\overline{\bar{x}yz} \cdot \overline{x\bar{y}z} \cdot w}$$

La funzione XOR e 3 circuiti per calcolarla

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0







Riferimenti

Computer Organization and Design

The Hardware/Software Interface 3rd Edition

David A. Patterson, John L. Hennessy

Appendice B

Versione italiana:

Struttura e Progetto dei Calcolatori

L'Interfaccia Hardware-Software

2ª edizione Zanichelli

http://en.wikipedia.org/ o http://it.wikipedia.org/