

A.A. 2021-2022

Elementi di Elettronica (INF)

Prof. Paolo Crippa

Circuiti Logici Combinatori – P3

Un circuito combinatorio è composto da porte logiche le cui uscite sono istante per istante funzioni logiche dei valori assunti dagli ingressi



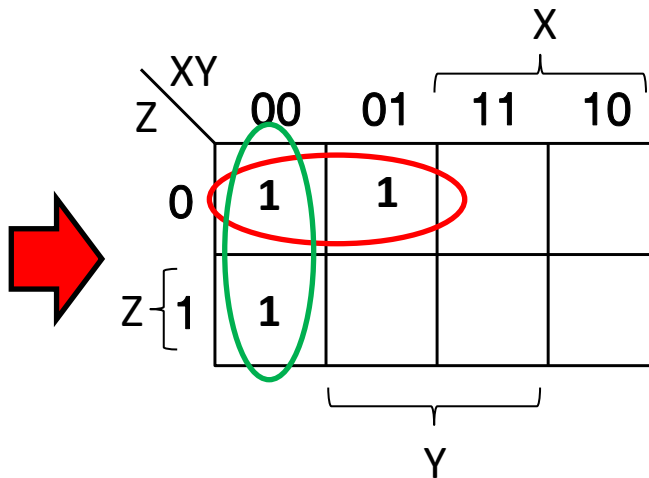
La progettazione di un circuito combinatorio consta dei seguenti (cinque) passi fondamentali:

- **Definizione delle specifiche**
- **Sintesi**
- **Ottimizzazione**
- **Implementazione**
- **Verifica**

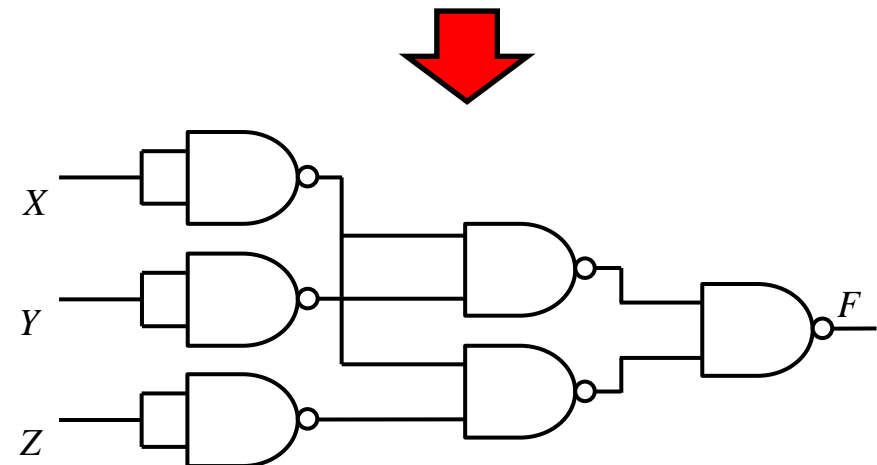
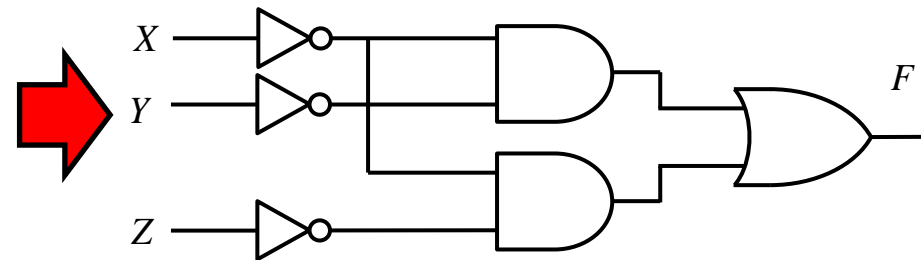
Esempio

Si progetti il circuito combinatorio a 3 ingressi e 1 uscita. L'uscita vale 1 quando il valore binario degli ingressi accostati è minore di 3 e 0 negli altri casi. Si usino soltanto porte NAND.

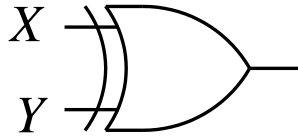
X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



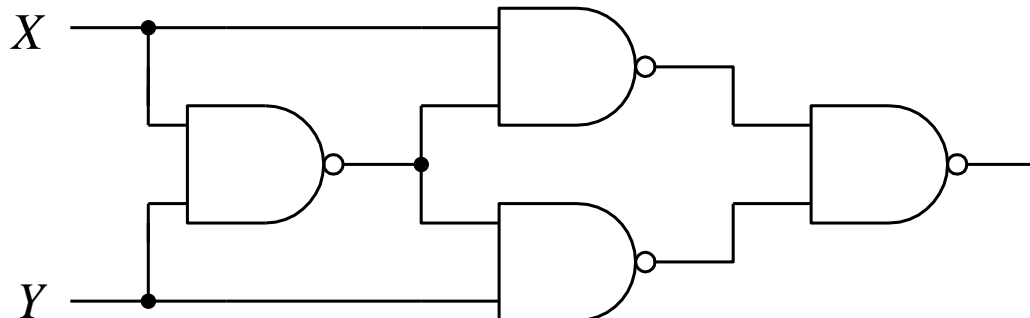
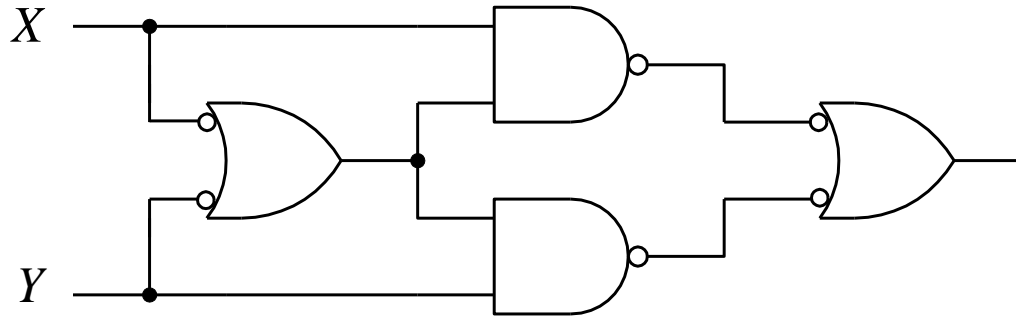
$$F = \bar{X} \cdot \bar{Y} + \bar{X} \cdot \bar{Z}$$



XOR



$$X \oplus Y = X \cdot \bar{Y} + \bar{X} \cdot Y = X \cdot (\bar{X} + \bar{Y}) + Y \cdot (\bar{X} + \bar{Y})$$



Funzione Dispari

A_0	A_1	A_2	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$P = A_0 \oplus A_1 \oplus A_2$$

$$= (A_0 \oplus A_1) \oplus A_2$$

$$= (A_0 \cdot \overline{A_1} + \overline{A_0} \cdot A_1) \cdot \overline{A_2} + (A_0 \cdot A_1 + \overline{A_0} \cdot \overline{A_1}) \cdot A_2$$

$$= A_0 \cdot \overline{A_1} \cdot \overline{A_2} + \overline{A_0} \cdot A_1 \cdot \overline{A_2} + \overline{A_0} \cdot \overline{A_1} \cdot A_2 + A_0 \cdot A_1 \cdot A_2$$

-> definizione di funzione dispari per
l'operatore XOR a 3 o più variabili

$$P = A_0 \oplus A_1 \oplus A_2$$

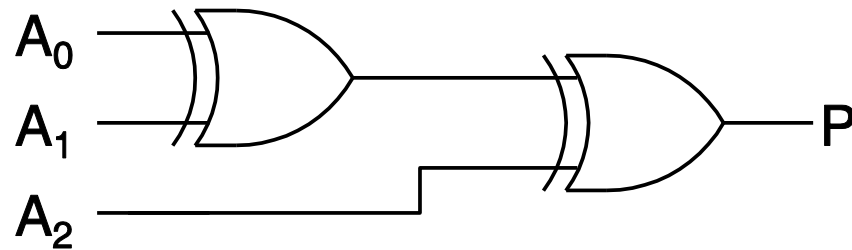
		$A_0 A_1$		A_0	
		00	01	11	10
A_2	0		1		1
A_2	1	1		1	

A_1

Generazione e Controllo di Parità

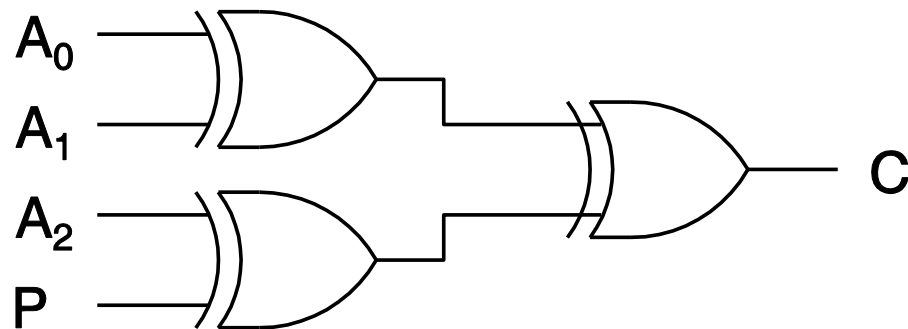
Messaggio a 3 bit			Bit di parità (pari)
A_0	A_1	A_2	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Generatore di Parità



$$P = A_0 \oplus A_1 \oplus A_2$$

Controllore di Parità

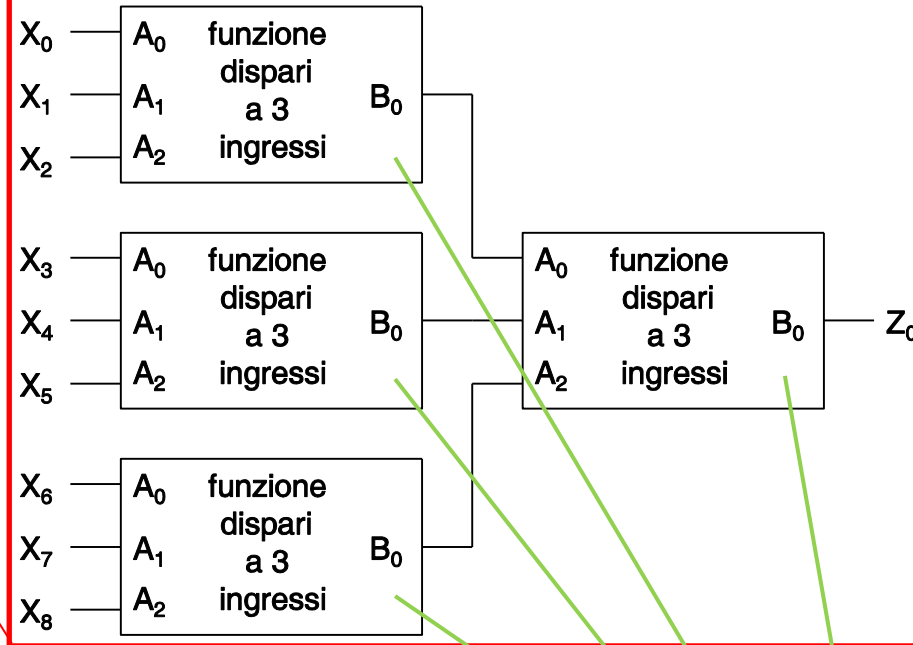


Funzione Dispari

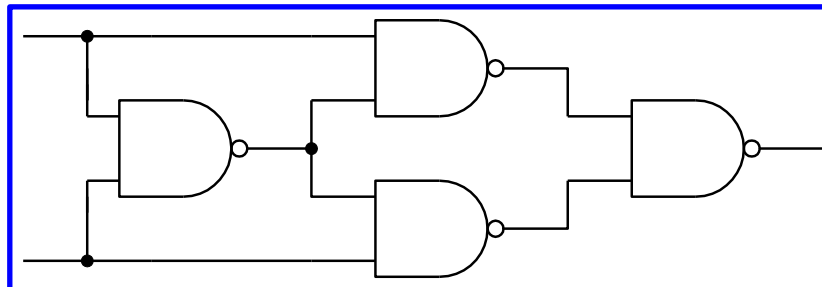
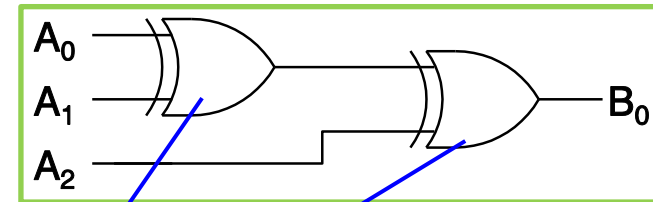
$$C = A_0 \oplus A_1 \oplus A_2 \oplus P$$

Parità PARI
C = 1 -> ERRORE

Progettazione Gerarchica

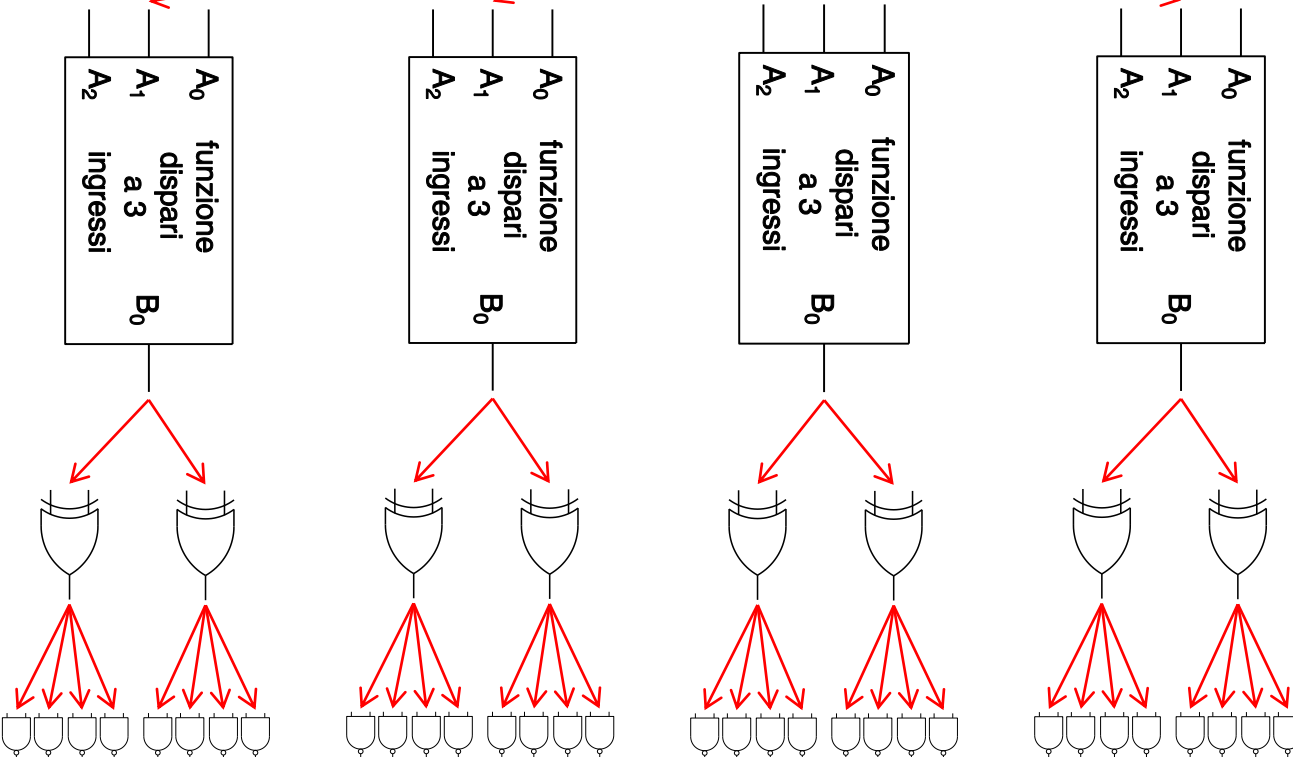
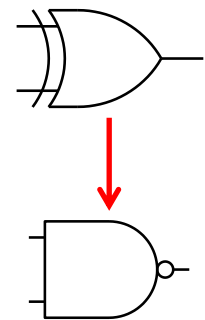
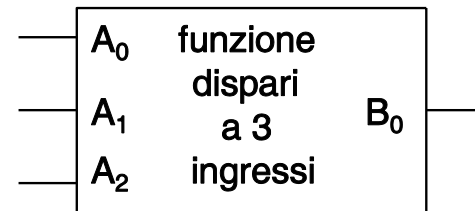
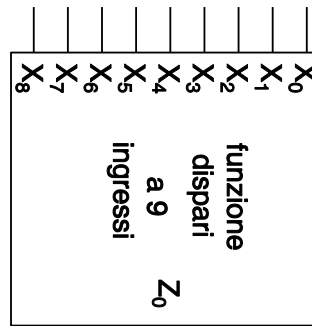


Esempio:
Funzione dispari a 9 ingressi



Progettazione Gerarchica

Esempio:
**Funzione dispari
a 9 ingressi**



Convertitori di Codice

Si progetti un convertitore di codice da “codice BCD” a “Codice Eccesso-3”

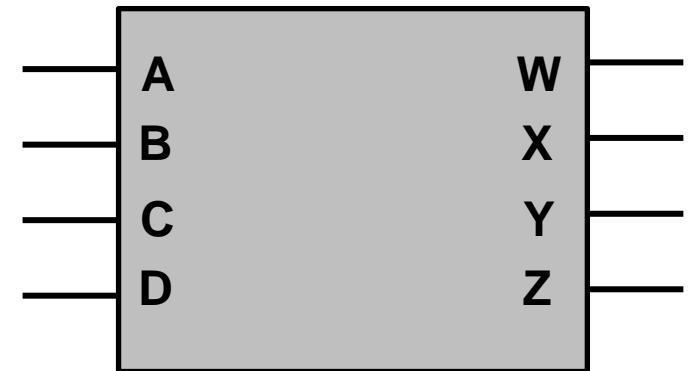
1) Tabella di verità

Funzione logica a 4 ingressi (BCD) e 4 uscite (Eccesso-3)

Cifra Decimale	Ingresso BCD				Uscita Eccesso-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

1) Tabella di verità completa

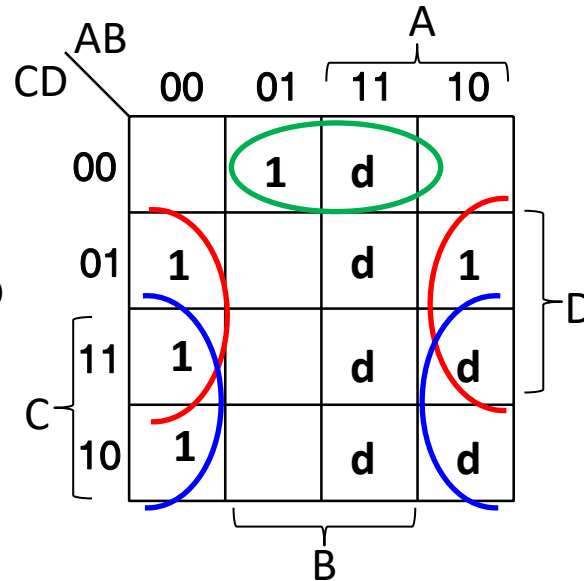
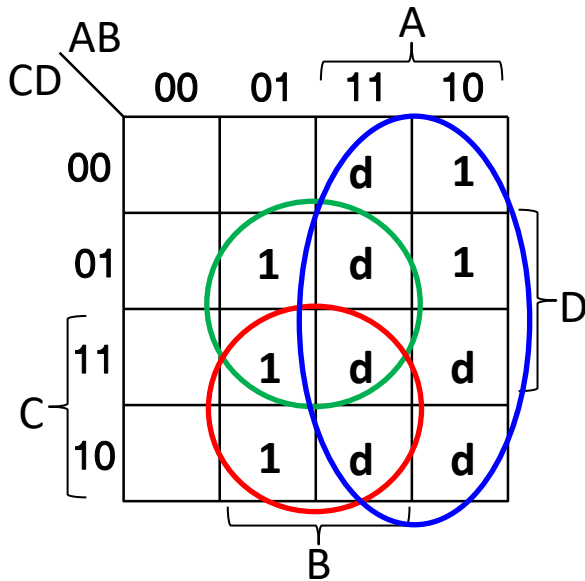
Ingresso BCD				Uscita Eccesso-3			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	d	d	d	d
1	0	1	1	d	d	d	d
1	1	0	0	d	d	d	d
1	1	0	1	d	d	d	d
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d



Convertitori di Codice

2)

Mappe di Karnaugh

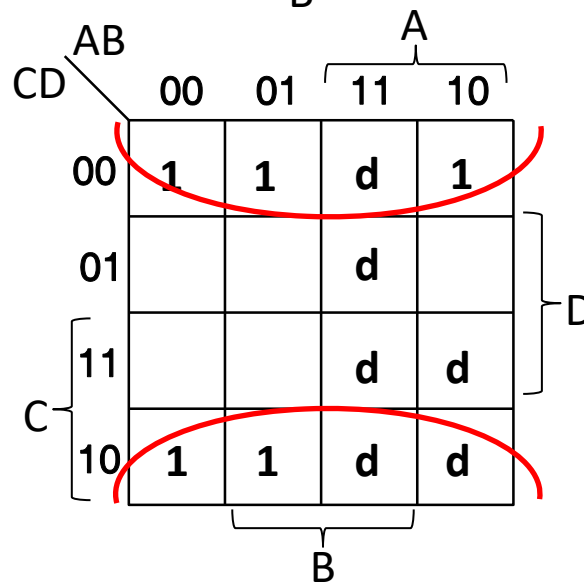
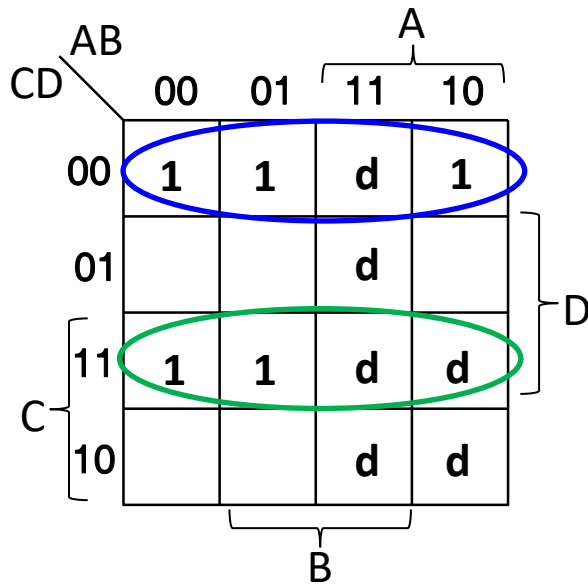


$$W = A + B \cdot C + B \cdot D$$

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

$$X = \bar{B} \cdot C + \bar{B} \cdot D + B \cdot \bar{C} \cdot \bar{D}$$

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



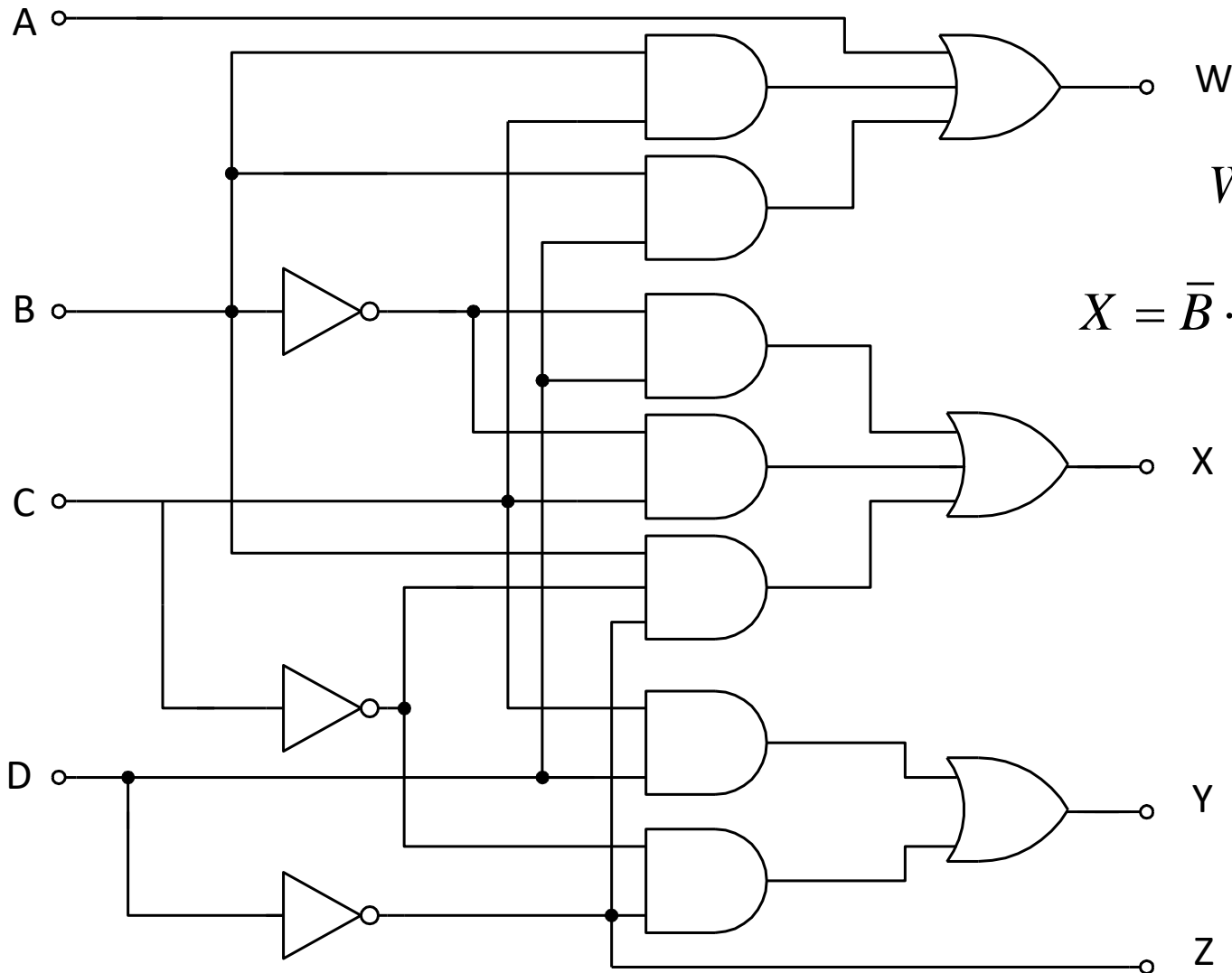
$$Y = C \cdot D + \bar{C} \cdot \bar{D}$$

$$= \overline{C \oplus D}$$

$$Z = \bar{D}$$

3) Diagramma Logico - I

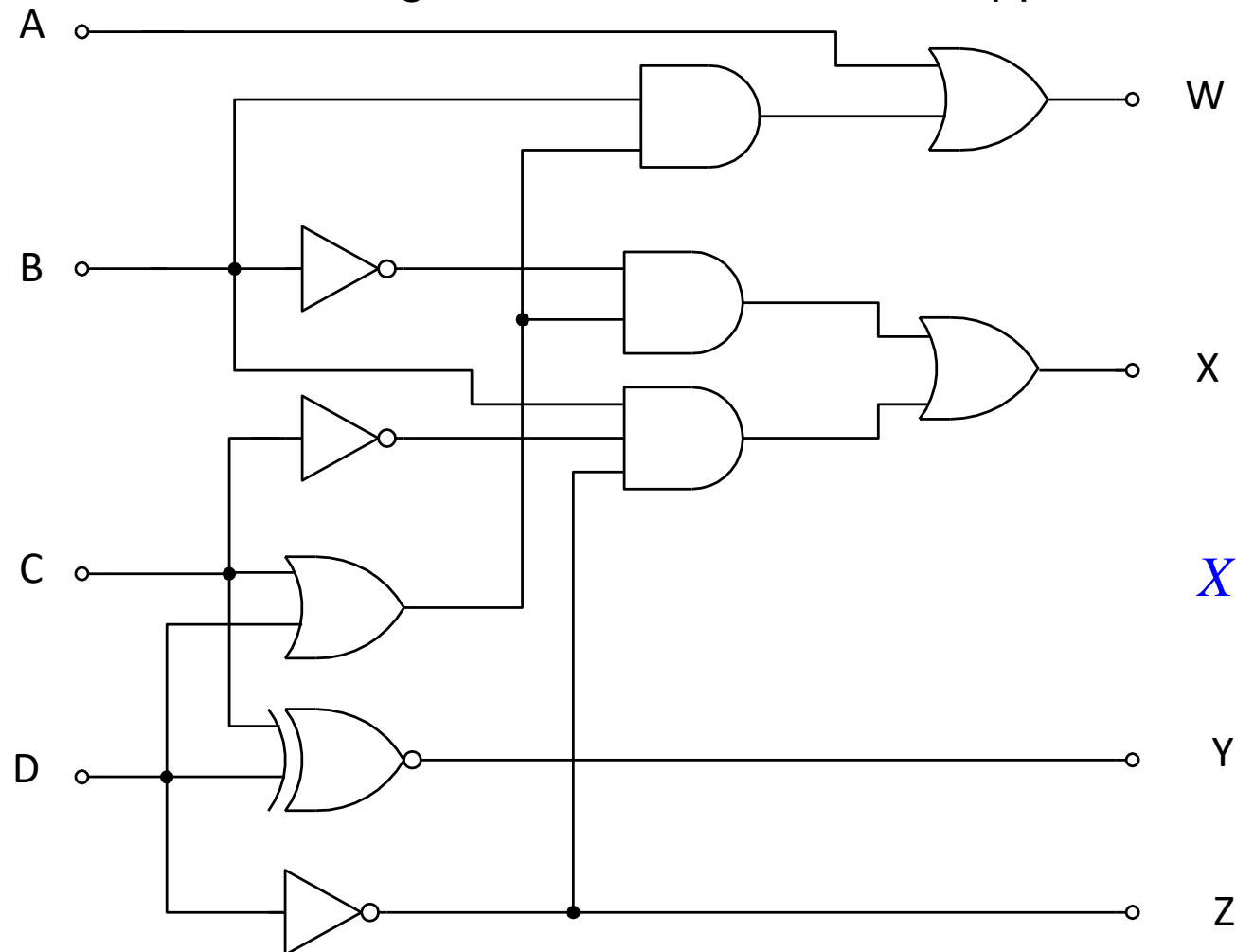
- A due livelli: direttamente dalle mappe di Karnaugh



Convertitori di Codice

3) Diagramma Logico - II

- Multilivello: manipolando i risultati precedenti cioè le espressioni algebriche derivate dalle mappe di Karnaugh



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

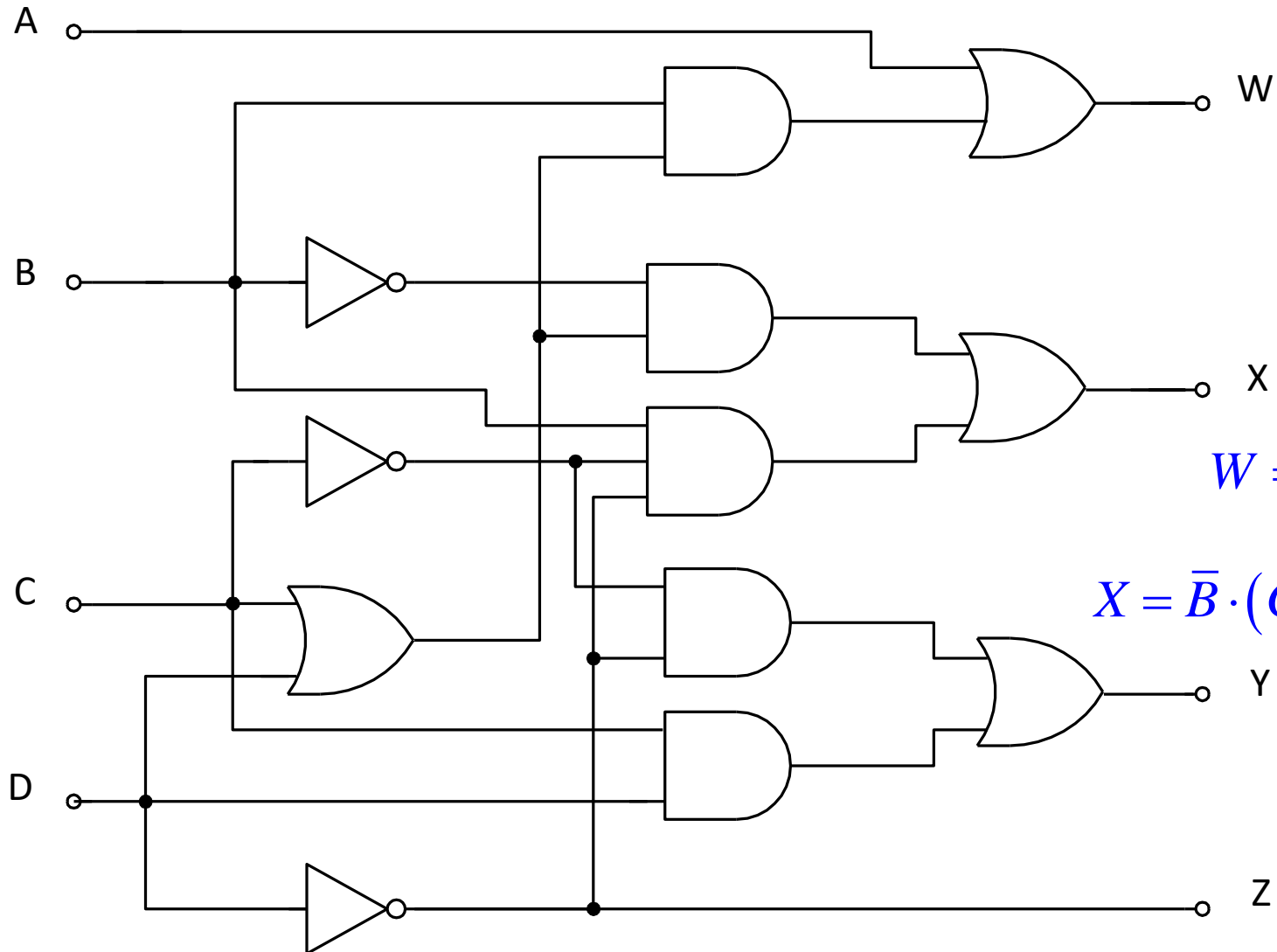
$$X = \bar{B} \cdot (C + D) + B \cdot \bar{C} \cdot \bar{D}$$

$$Y = \overline{C \oplus D}$$

$$Z = \bar{D}$$

3) Diagramma Logico - III

Multilivello (5 AND, 4 OR, 3 NOT)



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

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

$$Y = C \cdot D + \bar{C} \cdot \bar{D}$$

$$Z = \bar{D}$$

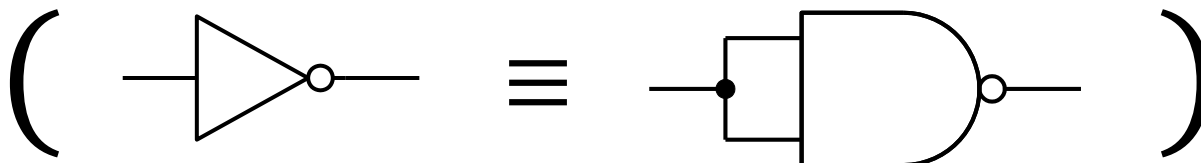
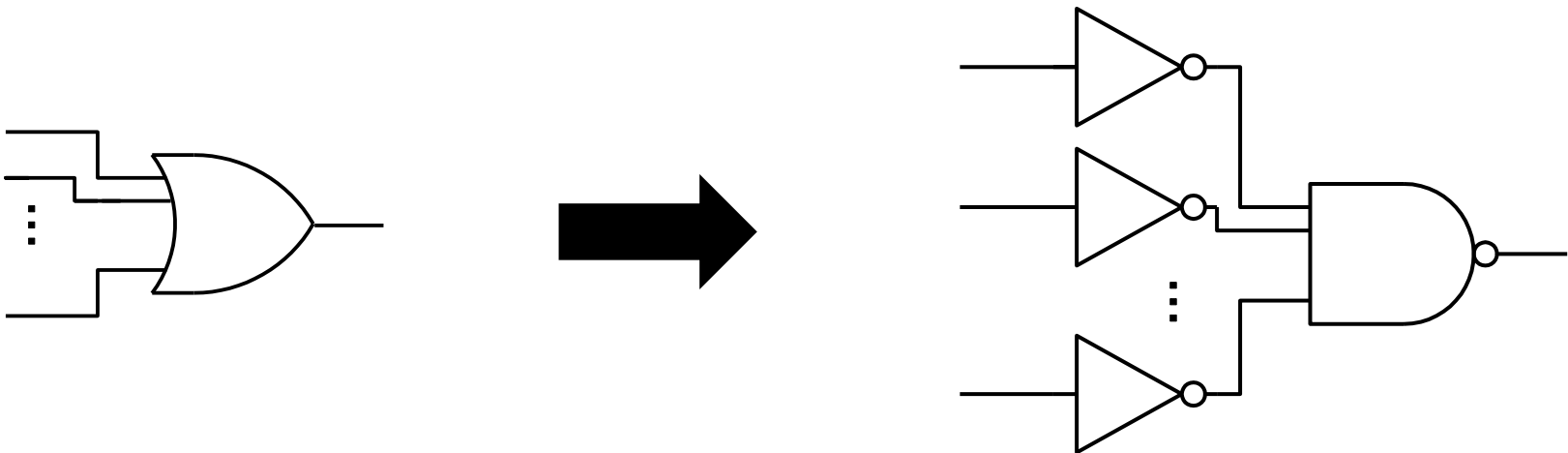
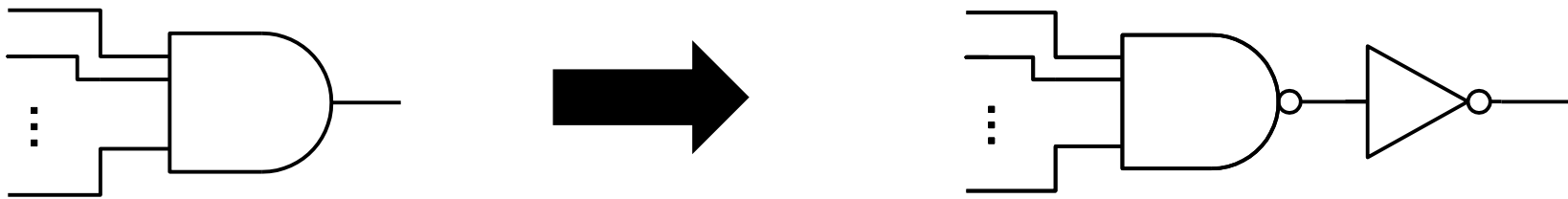
4) Implementazione con sole porte NAND (NOR)

Dato un circuito ottimizzato costituito da porte AND, OR e NOT, è possibile ottenere un circuito con soli NAND (NOR) senza limitazioni di fan-in (e di fan-out) basandosi sulla seguente procedura:

1. Sostituire ogni porta AND e ogni porta OR con il proprio circuito equivalente costituito da una porta NAND (NOR) e da uno o più invertitori.
2. Eliminare le coppie di invertitori ottenute a seguito delle sostituzioni precedenti.
3. Spostare gli inverter - che si trovano **tra** un ingresso del circuito o l'uscita di una porta NAND (NOR) che pilota un ramo **e** un ingresso a una porta NAND (NOR) pilotata - verso la porta NAND (NOR) pilotata eliminando le coppie di inverter che si vanno via via formando.
4. Sostituire un inverter che pilota n rami in parallelo con n invertitori sui rami in parallelo pilotati.
5. Ripetere gli ultimi due passi precedenti sino a quando non si ha, al più, un singolo invertitore **tra** un ingresso del circuito o l'uscita di una porta NAND (NOR) che pilota il ramo **e** la porta NAND (NOR) pilotata.

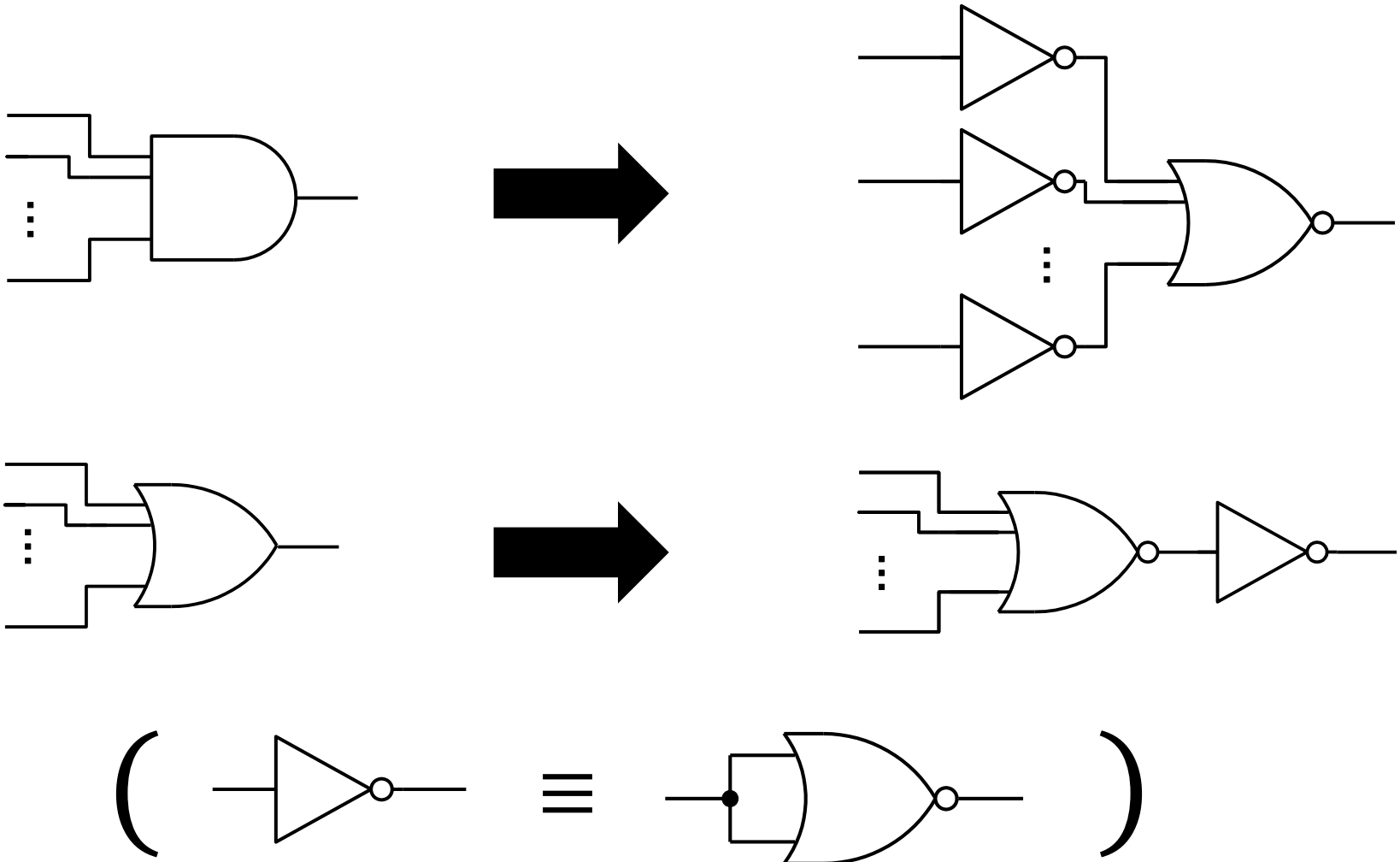
4) Implementazione (con porte NOR o con porte NAND)

Rappresentazione a porte NAND



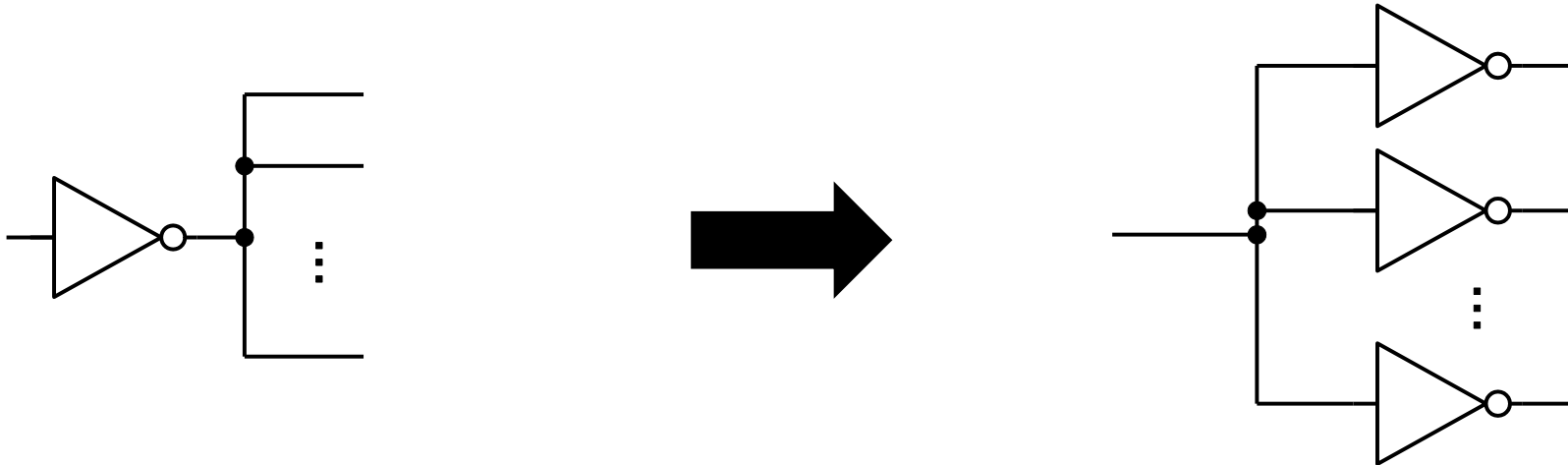
4) Implementazione (con porte NOR o con porte NAND)

Rappresentazione a porte NOR

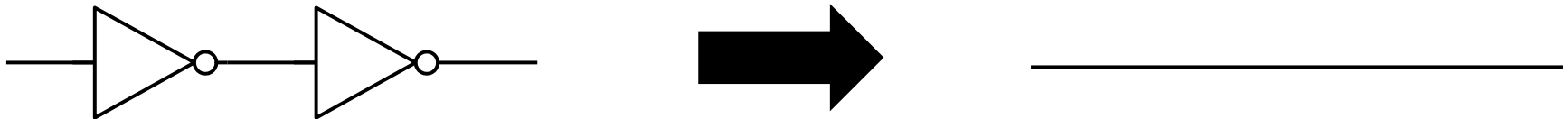


4) Implementazione (con porte NOR o con porte NAND)

Passaggio di un inverter (NOT) attraverso un "punto"



Cancellazione di coppie di inverter

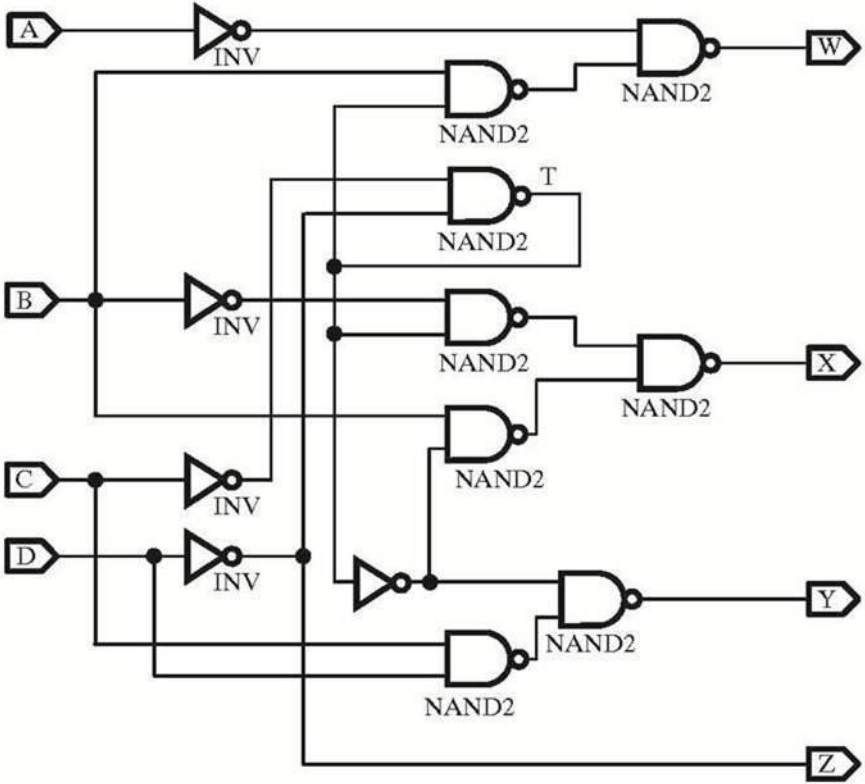


Convertitori di Codice

5)

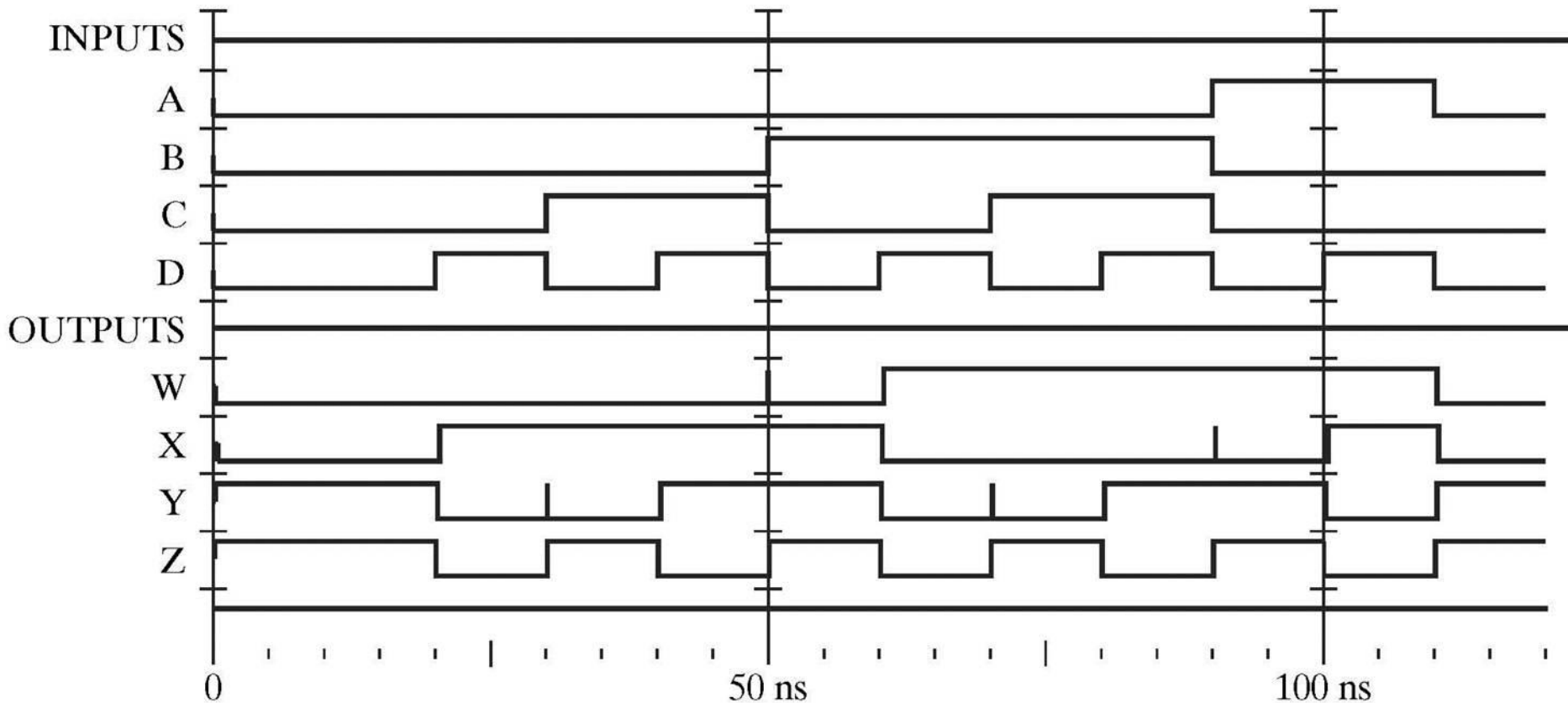
Verifica funzionale (manuale)

Ingresso BCD				Uscita Eccesso 3			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

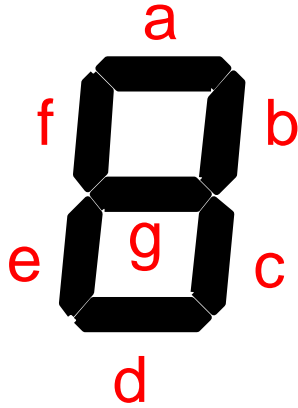


Ingresso BCD				Uscita Eccesso 3			
A	B	C	D	W	X	Y	Z
0	0	0	0				1
0	0	0	1				
0	0	1	0		1		1
0	0	1	1		1	1	
0	1	0	0				1
0	1	0	1	1			
0	1	1	0	1			1
0	1	1	1	1		1	
1	0	0	0	1			1
1	0	0	1	1			

5) Verifica funzionale (mediante simulatore)



Si progetti un convertitore di codice da “codice BCD” a “Codice a 7 segmenti”



Denominazione dei 7 segmenti



Scelta dei segmenti per la rappresentazione
delle diverse 10 cifre decimali

1) Tabella di verità

Funzione logica a 4 ingressi (BCD) e 7 uscite (7 segmenti)

Ingresso BCD				Convertitore sette-segmenti						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
tutti gli altri ingressi				0	0	0	0	0	0	0

2) Mappe di Karnaugh (7 mappe a 4 variabili)

$$a = \bar{A}C + \bar{A}BD + \bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}$$

$$b = \bar{A}\bar{B} + \bar{A}\bar{C}\bar{D} + \bar{A}CD + A\bar{B}\bar{C}$$

$$c = \bar{A}B + \bar{A}D + \bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}$$

$$d = \bar{A}C\bar{D} + \bar{A}\bar{B}C + \bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C} + \bar{A}B\bar{C}D$$

$$e = \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D}$$

$$f = \bar{A}B\bar{C} + \bar{A}\bar{C}\bar{D} + \bar{A}B\bar{D} + A\bar{B}\bar{C}$$

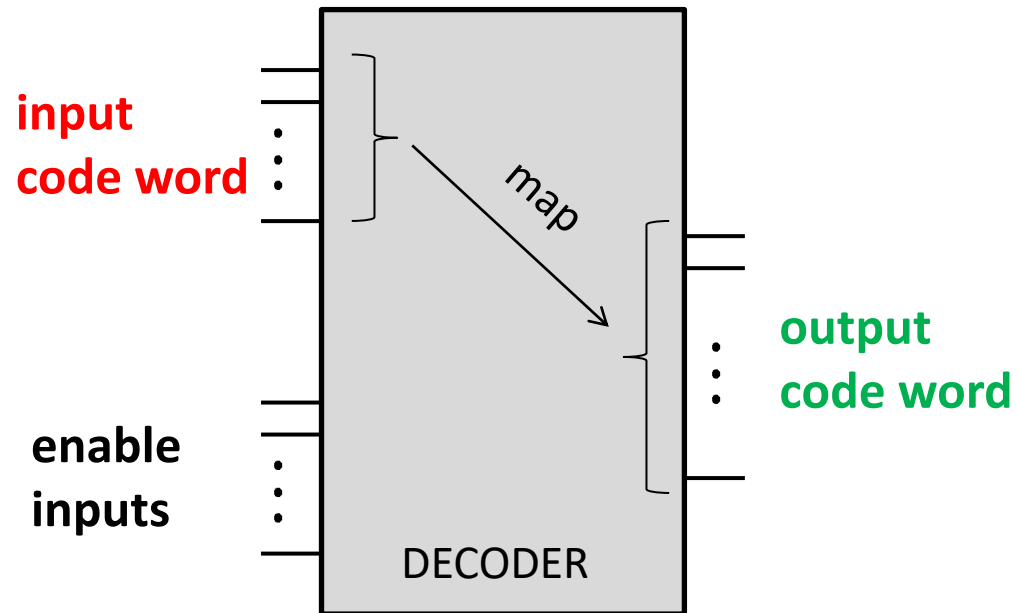
$$g = \bar{A}C\bar{D} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

3) Diagramma Logico (14 porte AND e 7 porte OR)

Decodificatori (Decoders)

Un **decodificatore** (*decoder*) è un circuito combinatorio che converte le informazioni binarie codificate a n bit applicate agli ingressi nelle corrispondenti 2^n che si trovano in uscita. Se alcune delle possibili informazioni codificate con n bit non sono utilizzate, il decoder può utilizzare un numero di uscite minore di 2^n

Schema di Principio



input code : codice binario a n bit

output code : 1-out-of- m (un solo bit = 1 su un totale di m bit)

mintermini



Decodificatori n-m ($m \leq 2^n$)

Esempio: decodificatore 3-8 ($n = 3$, $m = 8$)

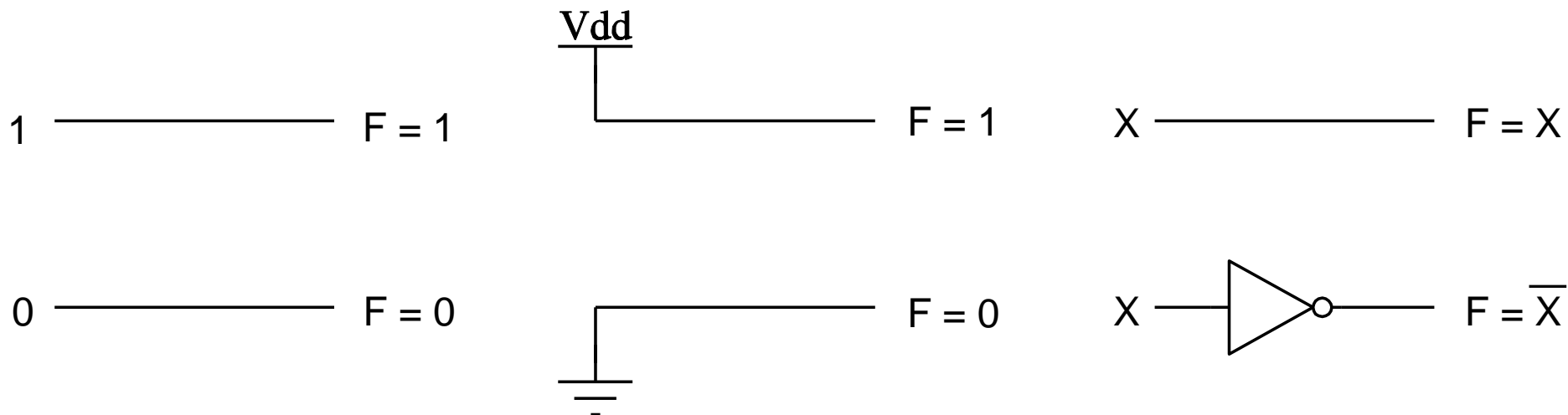
Tabella di Verità

Ingressi			Uscite							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

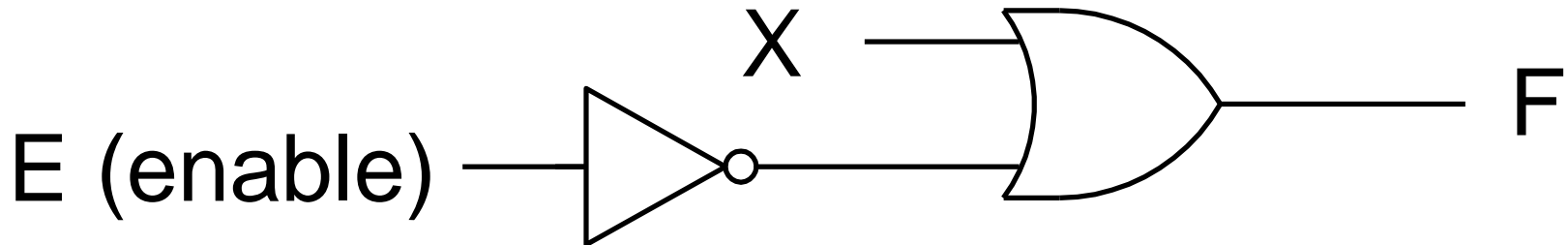
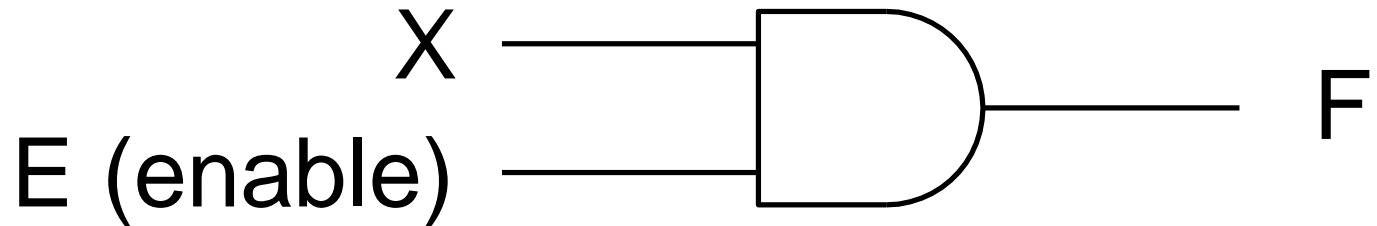
Funzioni Logiche Elementari

Assegnazione di Costante, Trasferimento e Negazione

X	$F = 0$	$F = X$	$F = \overline{X}$	$F = 1$
0	0	0	1	1
1	0	1	0	1



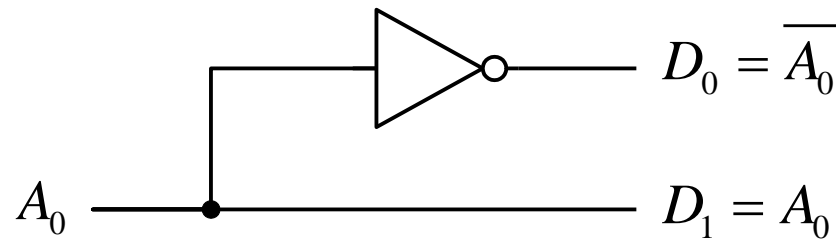
Abilitazione (Enabling)



Decodificatori n-m ($m \leq 2^n$)

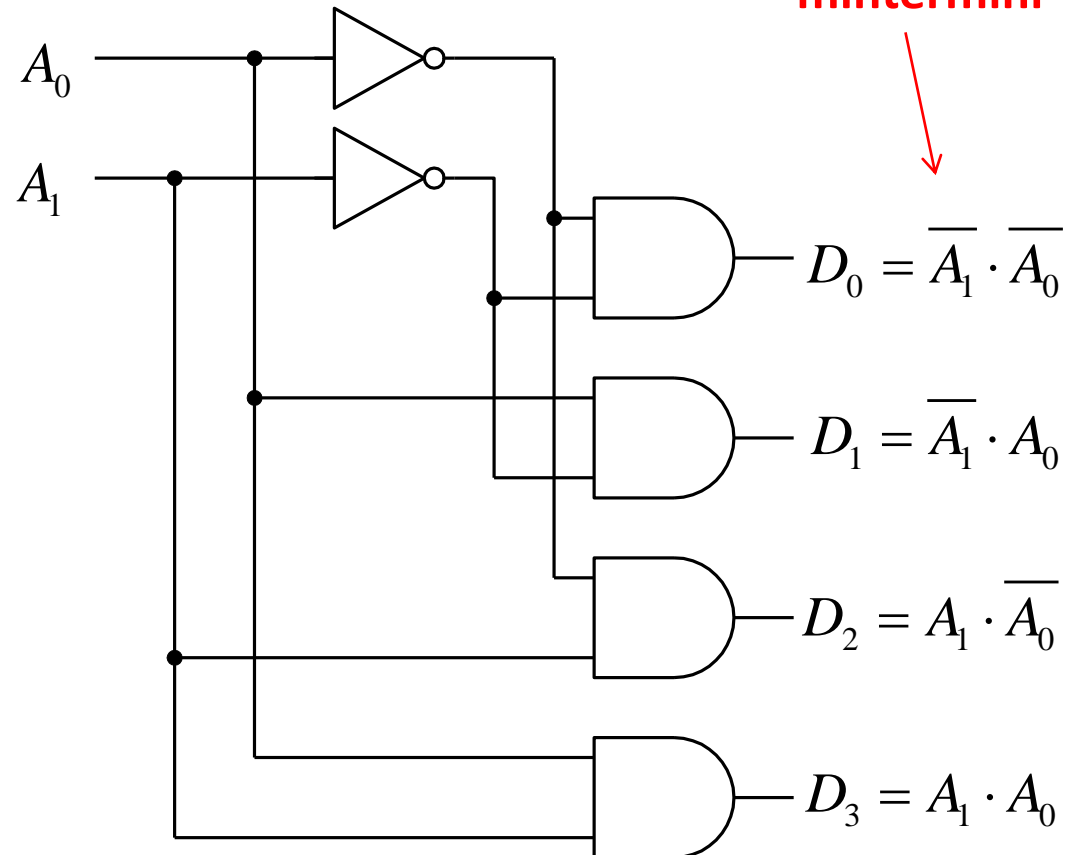
Decodificatore 1-2

A_0	D_0	D_1
0	1	0
1	0	1



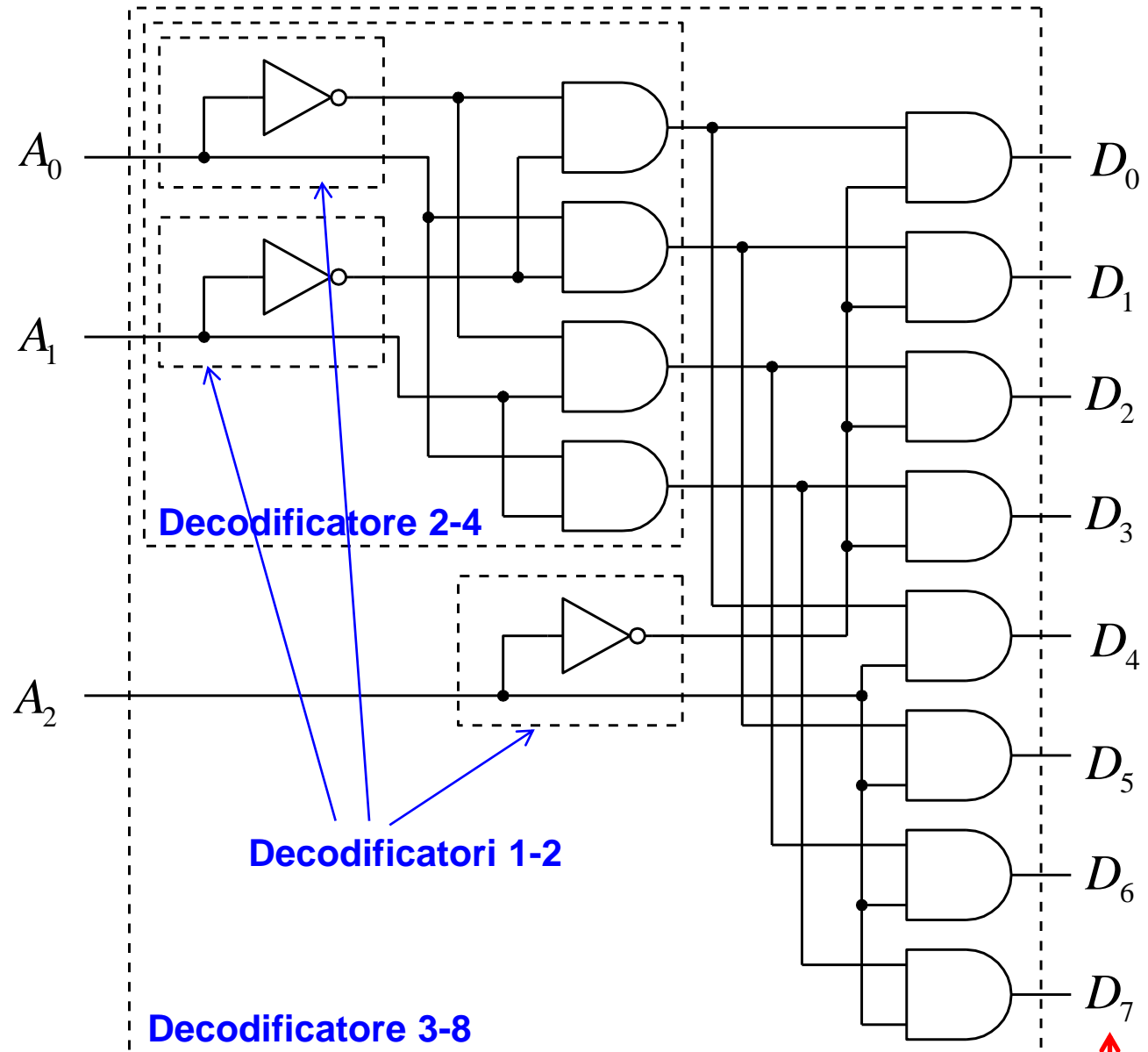
Decodificatore 2-4

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decodificatori n-m ($m \leq 2^n$)

Decodificatore 3-8

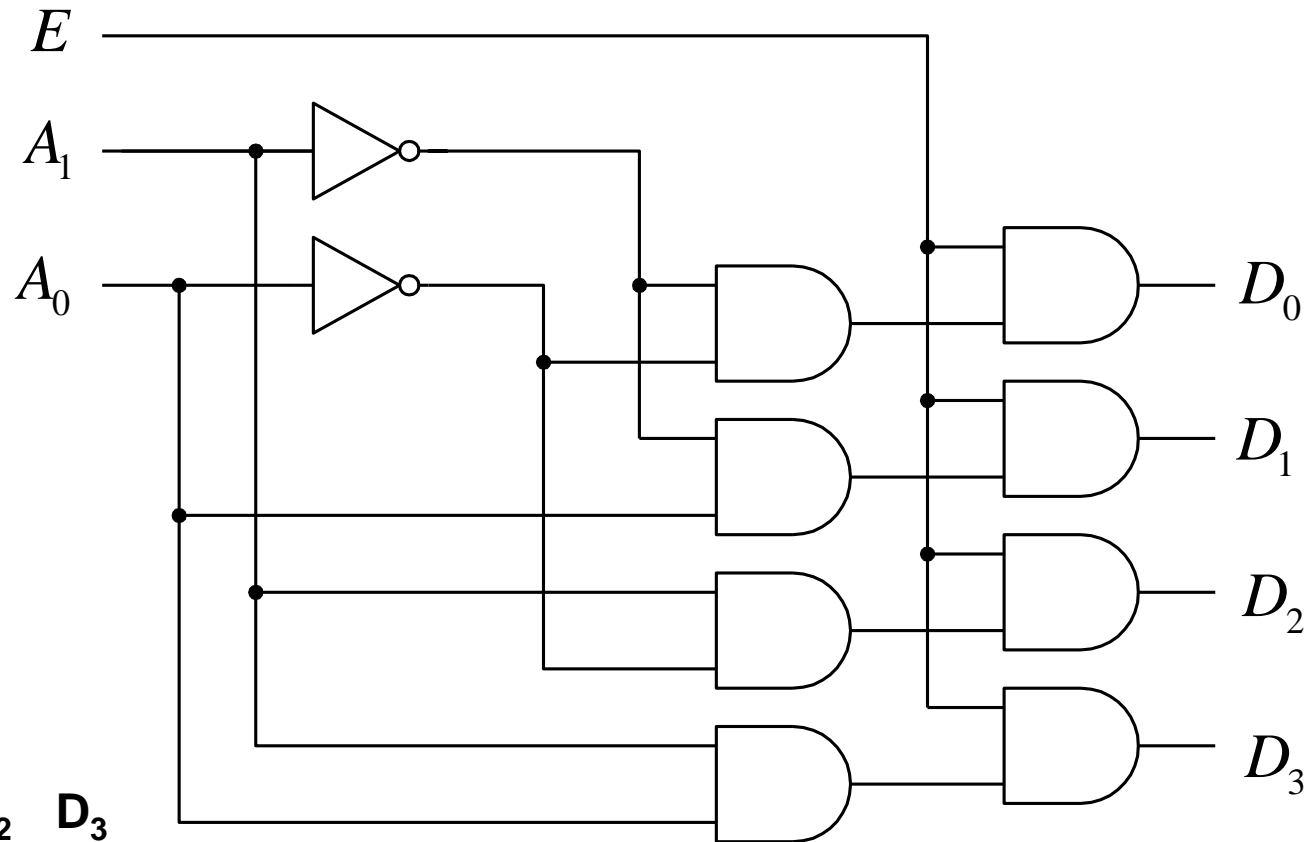


OSS.: solo porte
AND a 2 ingressi

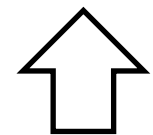
mintermini

Decodificatori n-m con Abilitazione

**OSS.: solo porte
AND a 2 ingressi**



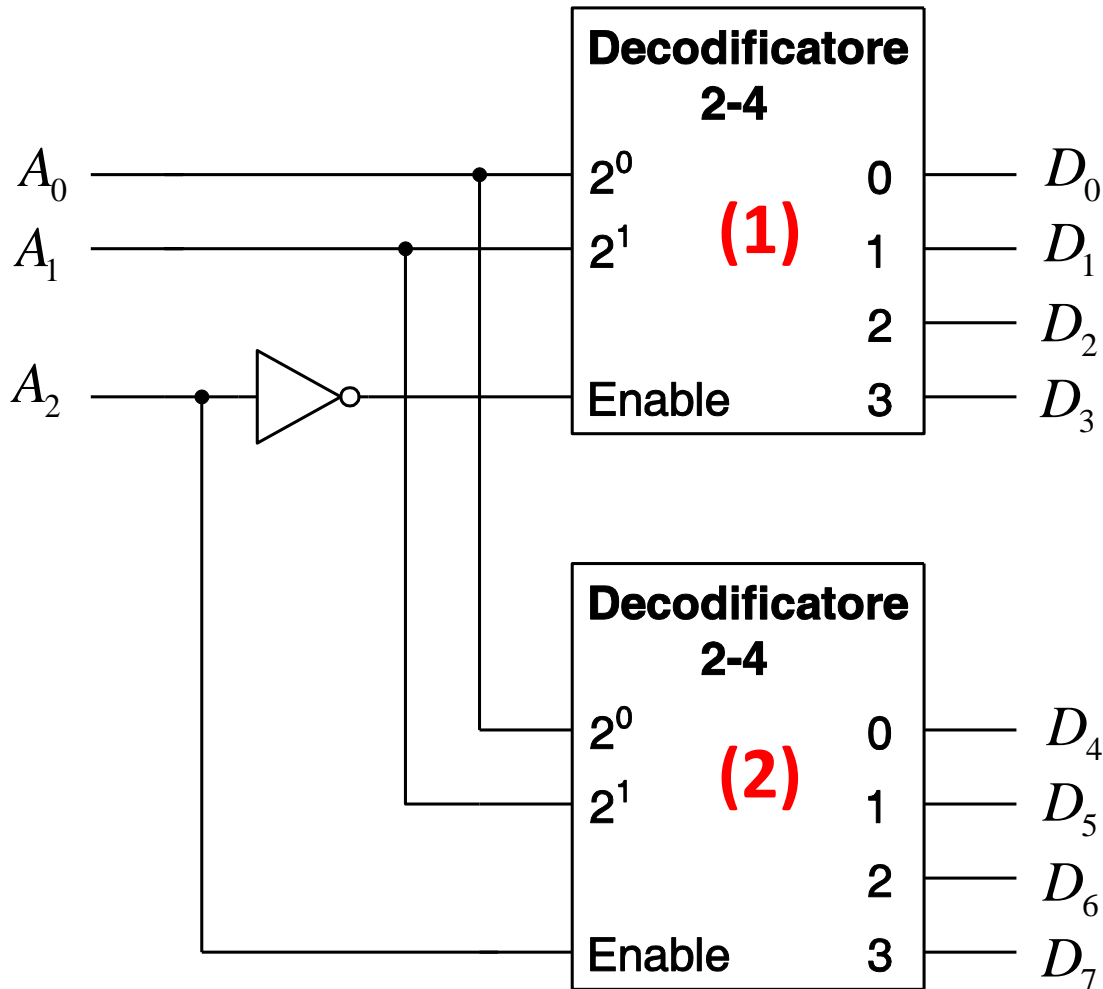
E	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



Demultiplexer 1-4 !!

Espansione dei Decodificatori

Decodificatore 3-8 realizzato con due Decodificatori 2-4



$A_2 = 0 \rightarrow (1) \text{ ABILITATO } \rightarrow D_0, D_1, D_2, D_3 \quad (D_4 = D_5 = D_6 = D_7 = 0)$

$A_2 = 1 \rightarrow (2) \text{ ABILITATO } \rightarrow D_4, D_5, D_6, D_7 \quad (D_0 = D_1 = D_2 = D_3 = 0)$

Espansione dei Decodificatori

Decodificatore 3-8 realizzato con due Decodificatori 2-4

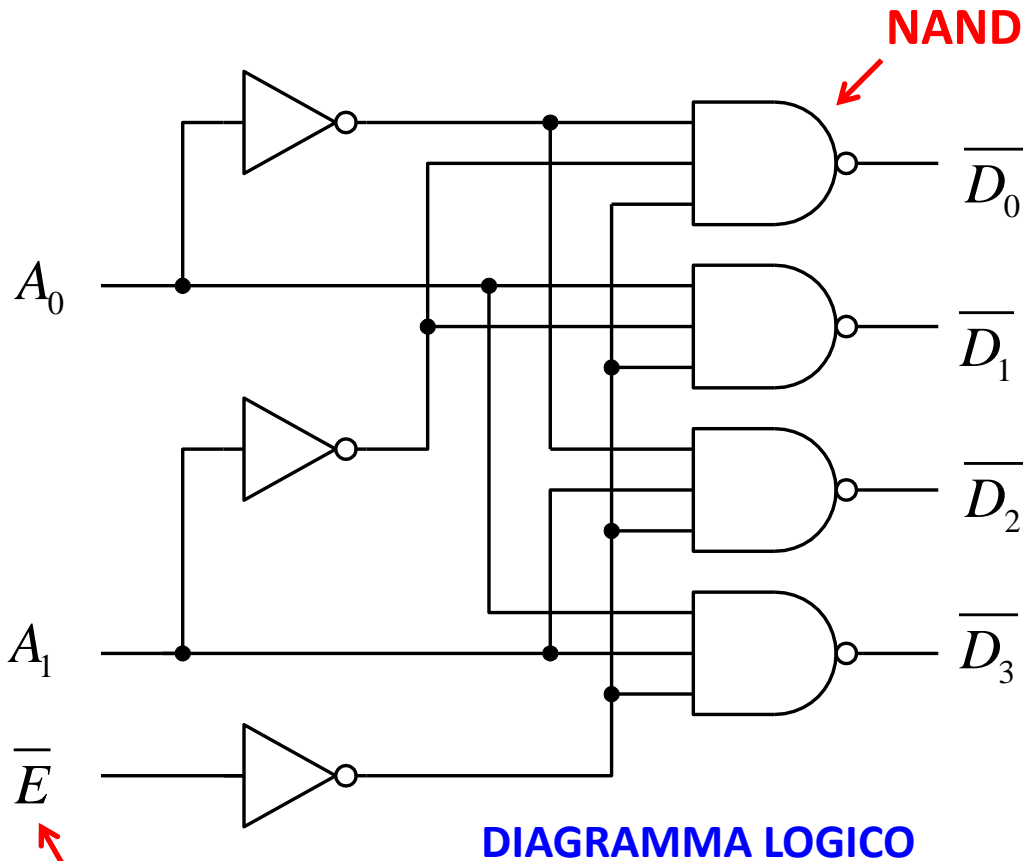
Ingressi			Uscite							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$A_2 = 0 \rightarrow$ (1) ABILITATO $\rightarrow D_0, D_1, D_2, D_3$ ($D_4 = D_5 = D_6 = D_7 = 0$)

$A_2 = 1 \rightarrow$ (2) ABILITATO $\rightarrow D_4, D_5, D_6, D_7$ ($D_0 = D_1 = D_2 = D_3 = 0$)

Decodificatori a NAND

Decodificatore 2-4: circuito logico e tabella di verità



\overline{E}	A_1	A_0	$\overline{D_0}$	$\overline{D_1}$	$\overline{D_2}$	$\overline{D_3}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

TABELLA DI VERITA'

$$\overline{D_0} = \overline{E \cdot \overline{A_1} \cdot \overline{A_0}}$$

$$\overline{D_1} = \overline{E \cdot \overline{A_1} \cdot A_0}$$

$$\overline{D_2} = \overline{E \cdot A_1 \cdot \overline{A_0}}$$

$$\overline{D_3} = \overline{E \cdot A_1 \cdot A_0}$$

EQUAZIONI LOGICHE

OSS. Enable e uscite sono complementate

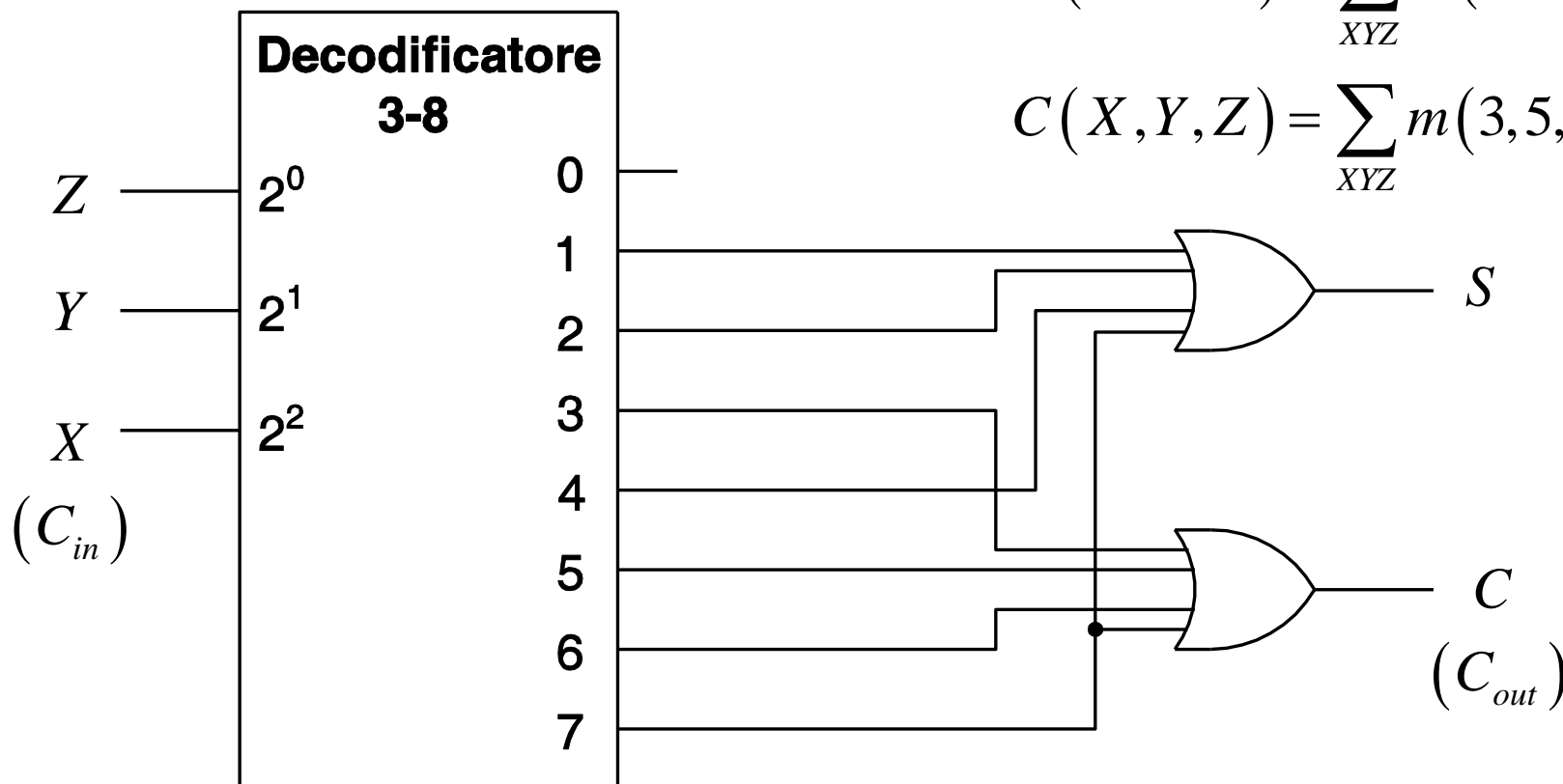
Implementazione di Circuiti Combinatori Mediante Decodificatori (e Porte OR)

Un decoder fornisce 2^n mintermini corrispondenti alle n variabili di ingresso



Un qualunque circuito combinatorio a n ingressi e m uscite può essere implementato con un decoder $n-2^n$ e m porte OR

Esempio: sommatore binario



$$S(X, Y, Z) = \sum_{XYZ} m(1, 2, 4, 5)$$

$$C(X, Y, Z) = \sum_{XYZ} m(3, 6, 7)$$

Implementazione di Circuiti Combinatori Mediante Decodificatori (e Porte OR)

Sommatore binario: tabella di verità

X (Cin)	Y	Z	C (Cout)	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2^n (o meno) ingressi
n uscite

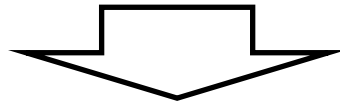
[illegible]

Codificatori

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$



Il codificatore da ottale a binario può essere implementato con 3 porte OR a 4 ingressi

Ambiguità:

1) due ingressi attivi contemporaneamente

-> occorre stabilire una priorità

2) ingressi tutti nulli a cui corrispondono uscite tutte nulle
(coincide con la condizione $D_0 = 1$)

-> occorre aggiungere un'uscita che indichi che almeno un ingresso = 1

Codificatori con Priorità

Un codificatore con priorità è un circuito combinatorio che comprende una funzione di priorità: se 2 o più ingressi sono contemporaneamente uguali a 1 ha precedenza l'ingresso con priorità maggiore

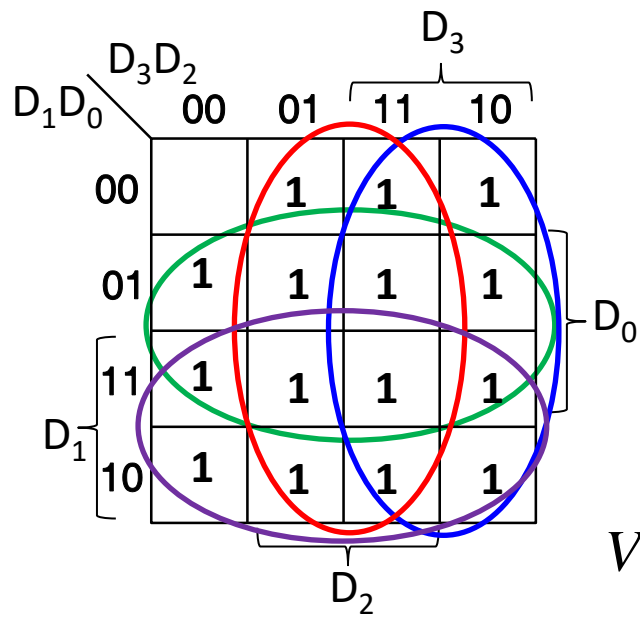
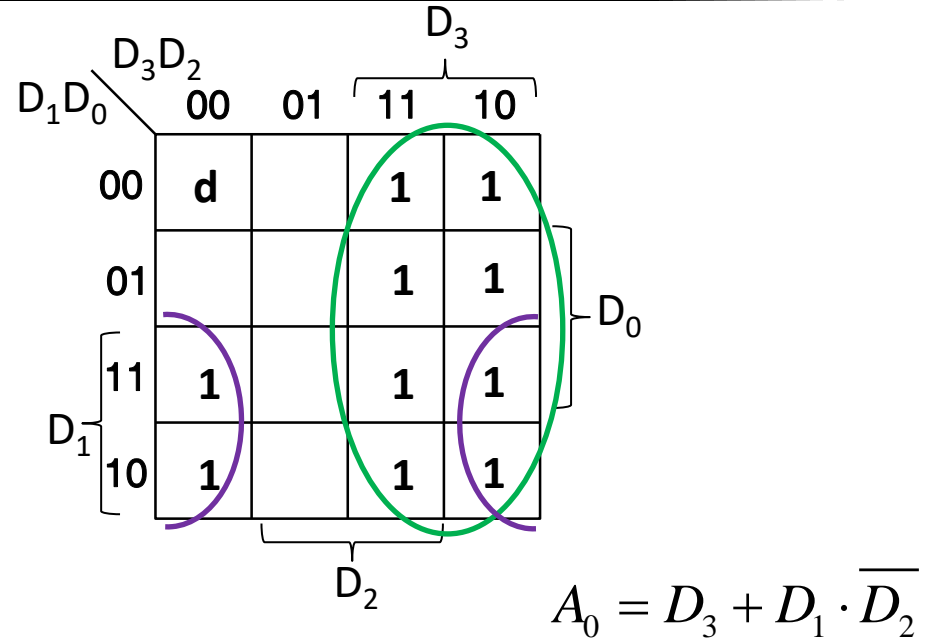
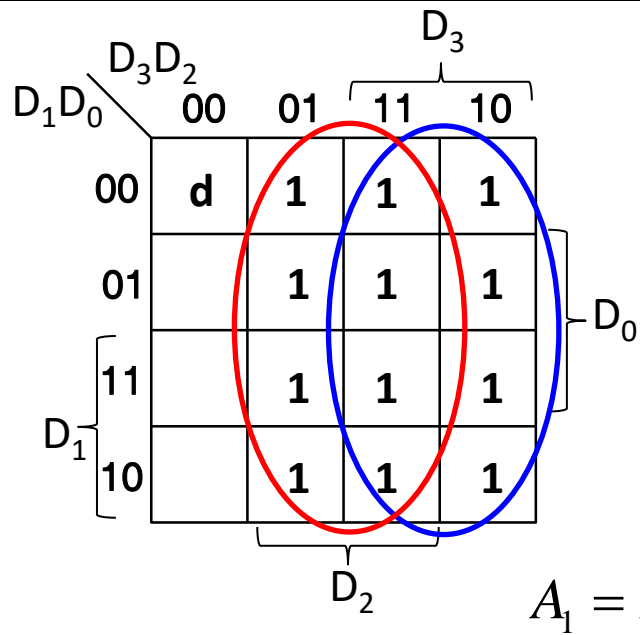
Tabella di verità **densa**

Ingressi				Uscite		
D ₃	D ₂	D ₁	D ₀	A ₁	A ₀	V
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

} 2 righe
 } 4 righe
 } 8 righe

don't care

Codificatori con Priorità



Mappe di Karnaugh

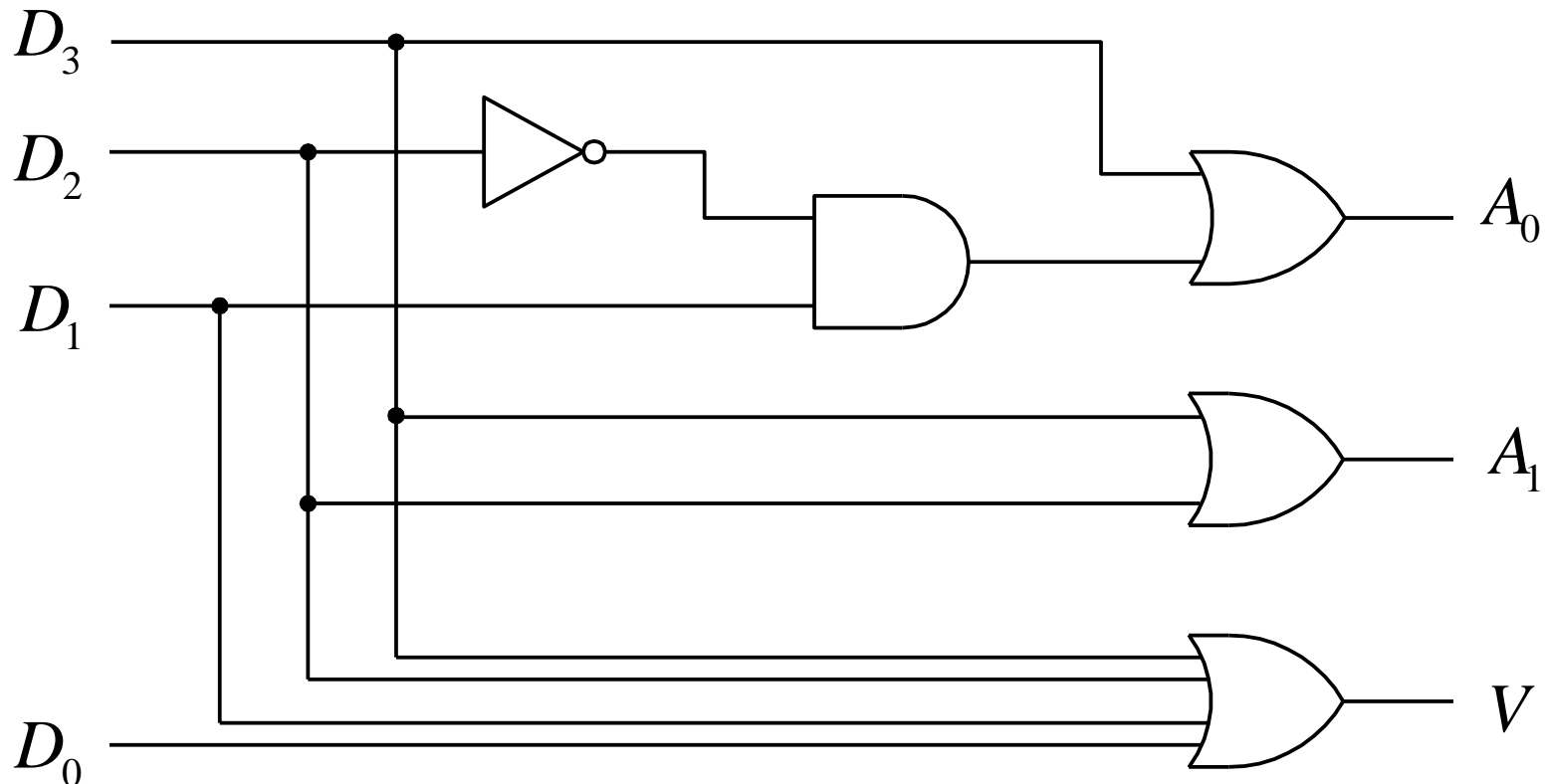
Codificatori con Priorità

$$A_0 = D_3 + D_1 \cdot \bar{D}_2$$

$$A_1 = D_2 + D_3$$

$$V = D_0 + D_1 + D_2 + D_3$$

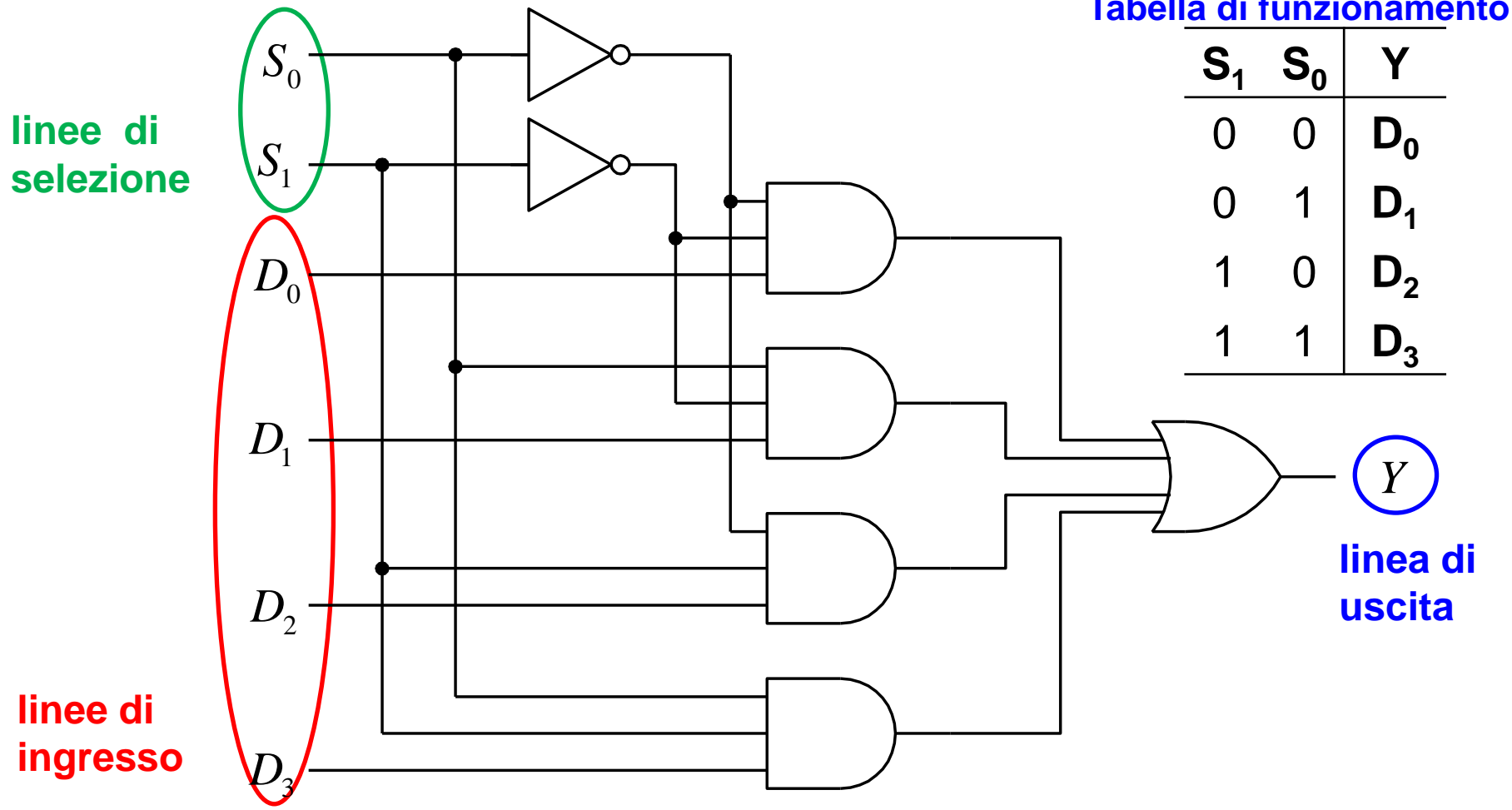
Diagramma logico per un codificatore con priorità a 4 ingressi



Multiplexer

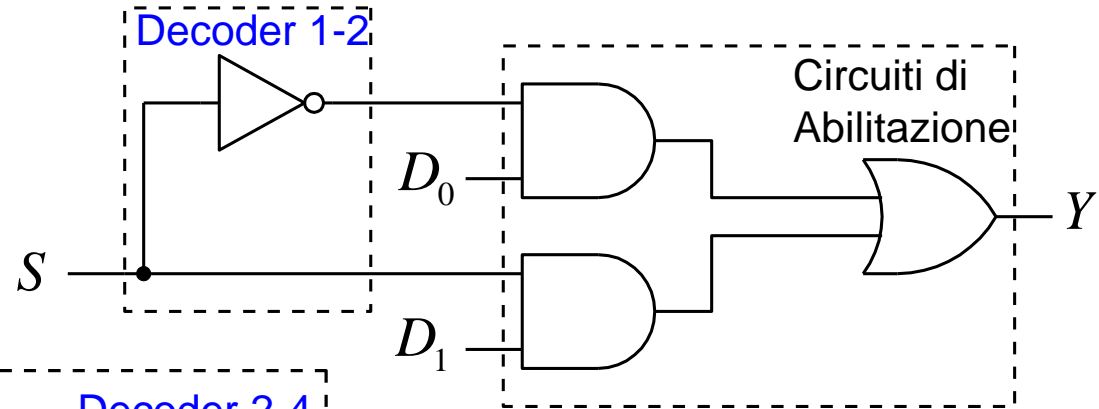
Un **multiplexer** (**selettore**) è un circuito combinatorio che seleziona segnali binari provenienti da una o più linee di ingresso e li dirige a una singola linea di uscita

$$Y = \bar{S}_1 \cdot \bar{S}_0 \cdot D_0 + \bar{S}_1 \cdot S_0 \cdot D_1 + S_1 \cdot \bar{S}_0 \cdot D_2 + S_1 \cdot S_0 \cdot D_3$$

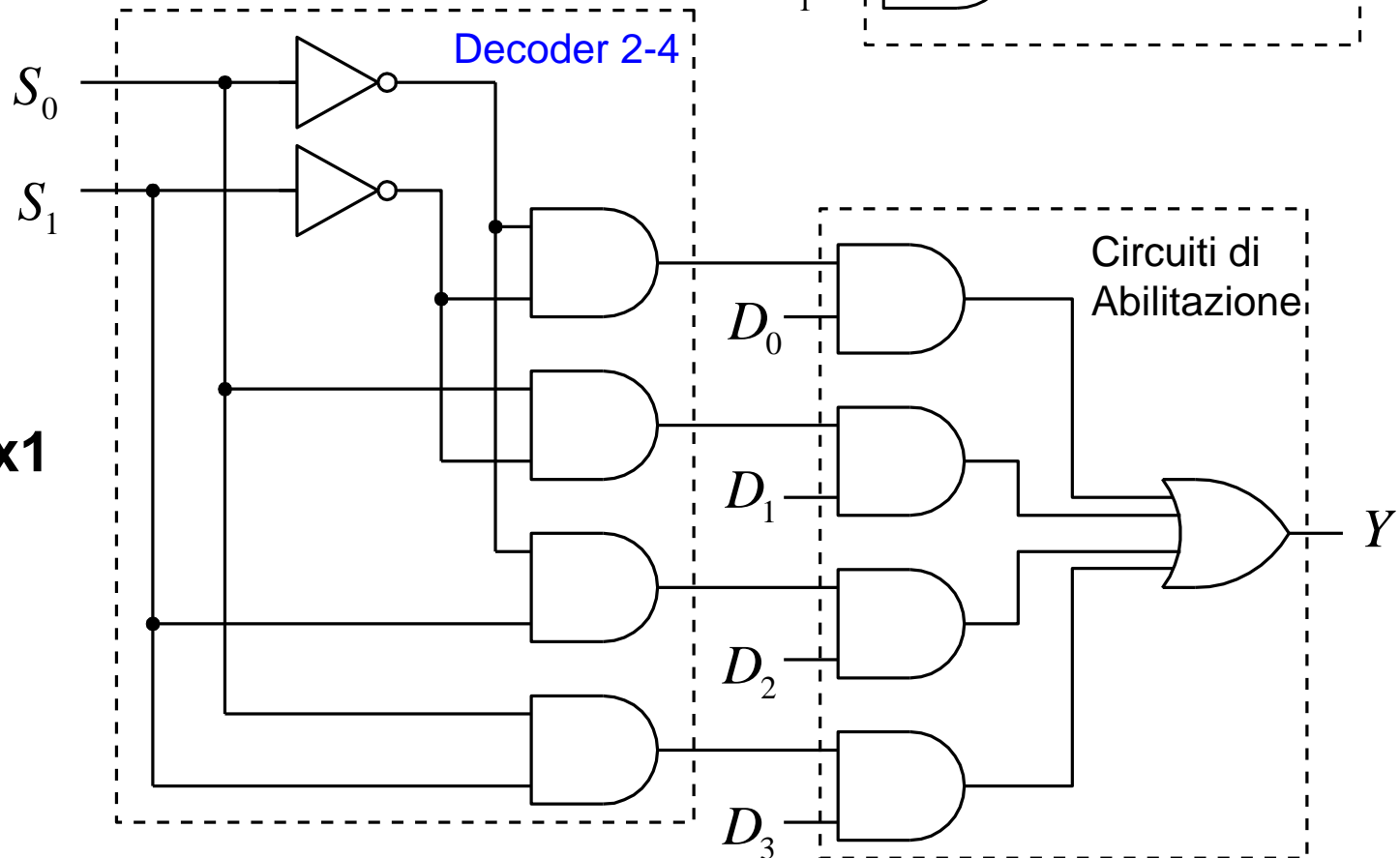


Multiplexer

Multiplexer 2x1

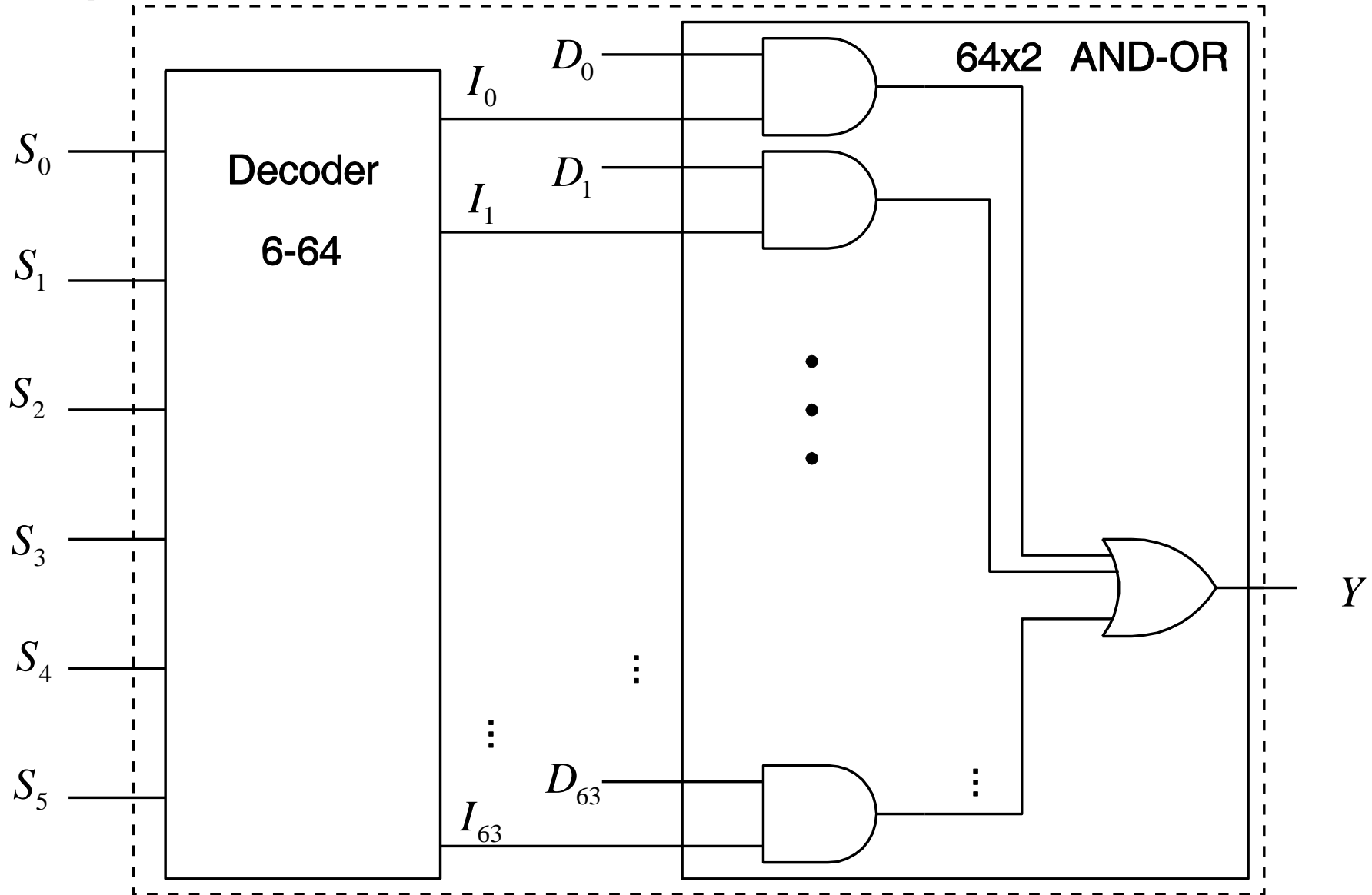


Multiplexer 4x1



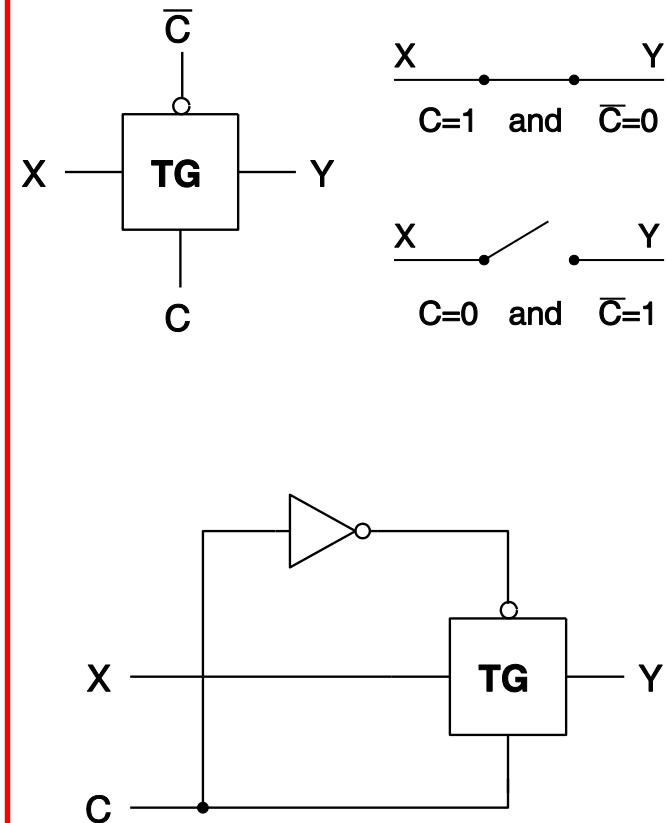
Multiplexer

Multiplexer 64x1

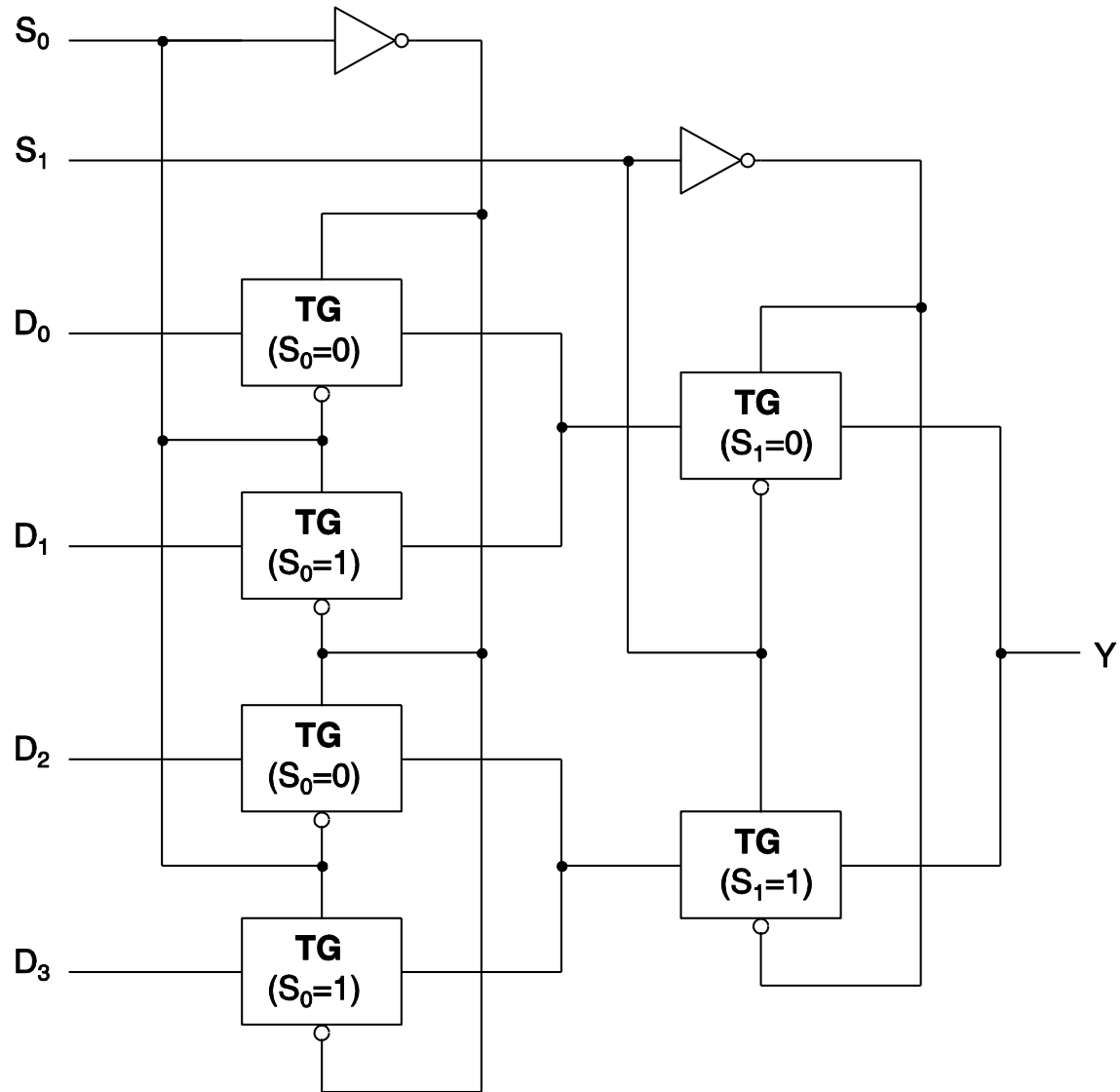


Multiplexer con Transmission Gate

Transmission Gates (TG)
o porte di trasmissione



Multiplexer 4x1 con TG



Multiplexer Combinati in Parallelo

Multiplexer 2x1 a quattro canali

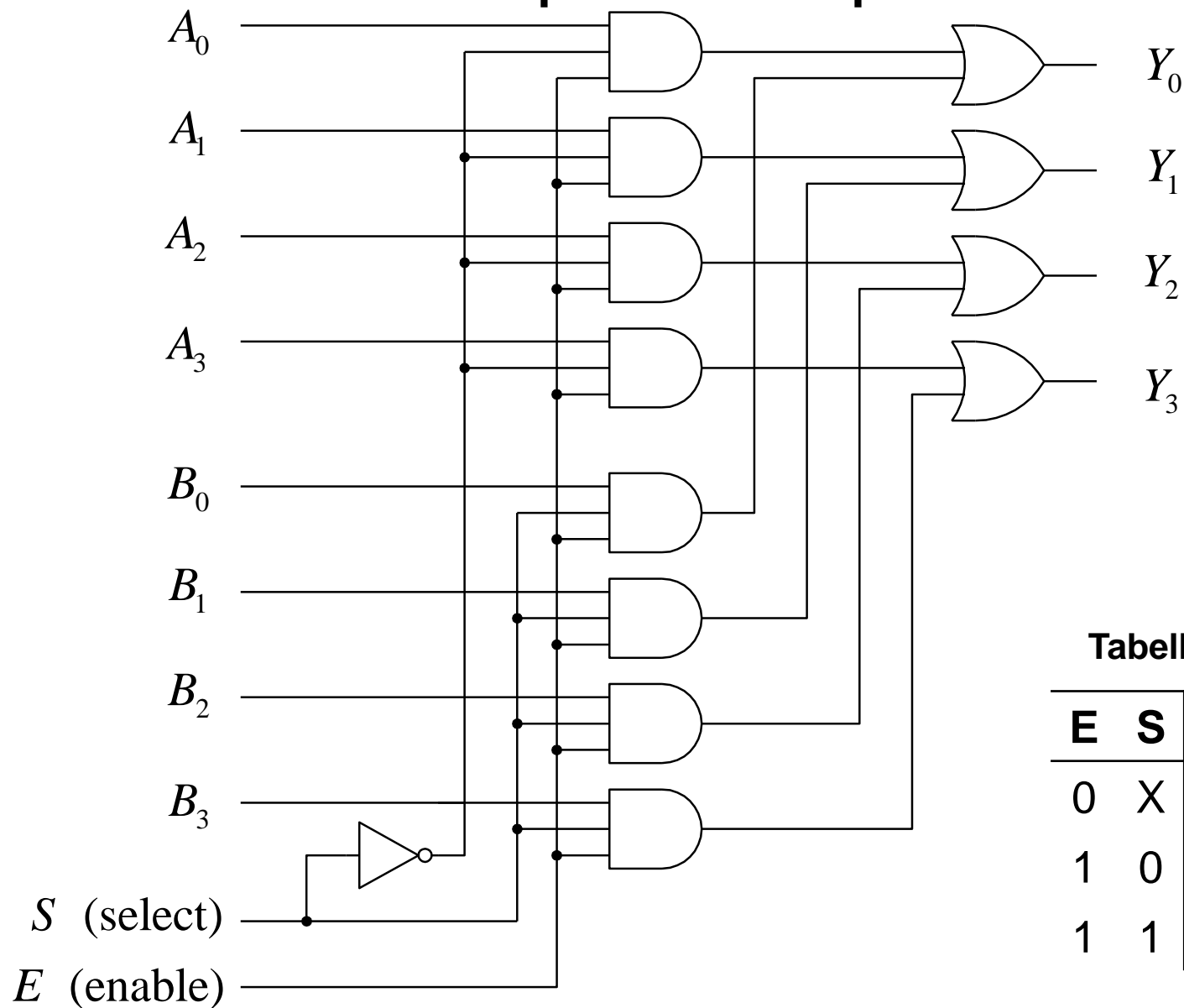


Tabella di funzionamento

E	S	Y_0	Y_1	Y_2	Y_3
0	X	0	0	0	0
1	0	A_0	A_1	A_2	A_3
1	1	B_0	B_1	B_2	B_3

Realizzazione di Circuiti Combinatori con Multiplexer

Una funzione Booleana a n variabili (X_1, X_2, \dots, X_n) può essere realizzata con un multiplexer a $n-1$ ingressi di selezione



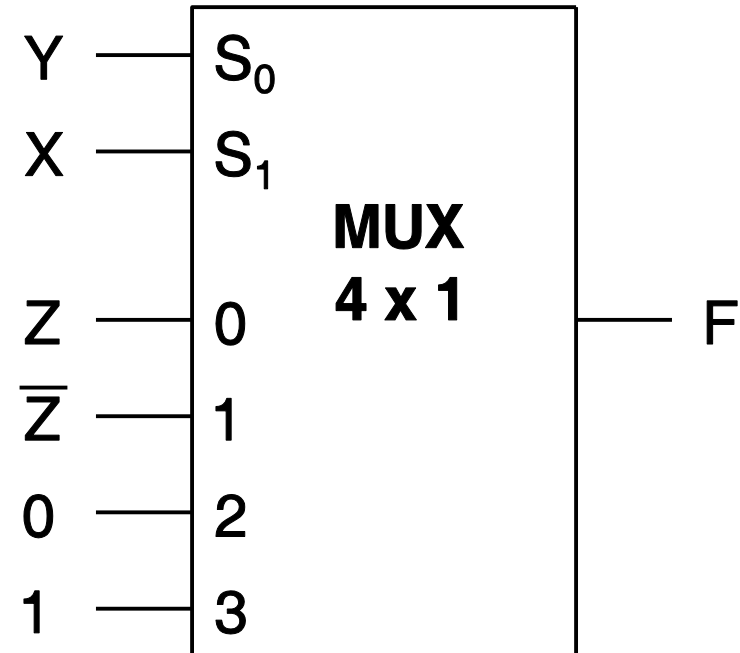
- prime $n-1$ variabili (X_1, X_2, \dots, X_{n-1}) \rightarrow ingressi di selezione
- ultima variabile (X_n) \rightarrow ingresso dati come $X_n, \overline{X_n}, 0, 1$

Esempi:

$$F(X, Y, Z) = \sum_{XYZ} m(1, 2, 6, 7)$$

Tabella di verità

	X	Y	Z	F	
0	0	0	0	0	$F = Z$
	0	0	1	1	
1	0	1	0	1	$F = \overline{Z}$
	0	1	1	0	
2	1	0	0	0	$F = 0$
	1	0	1	0	
3	1	1	0	1	$F = 1$
	1	1	1	1	



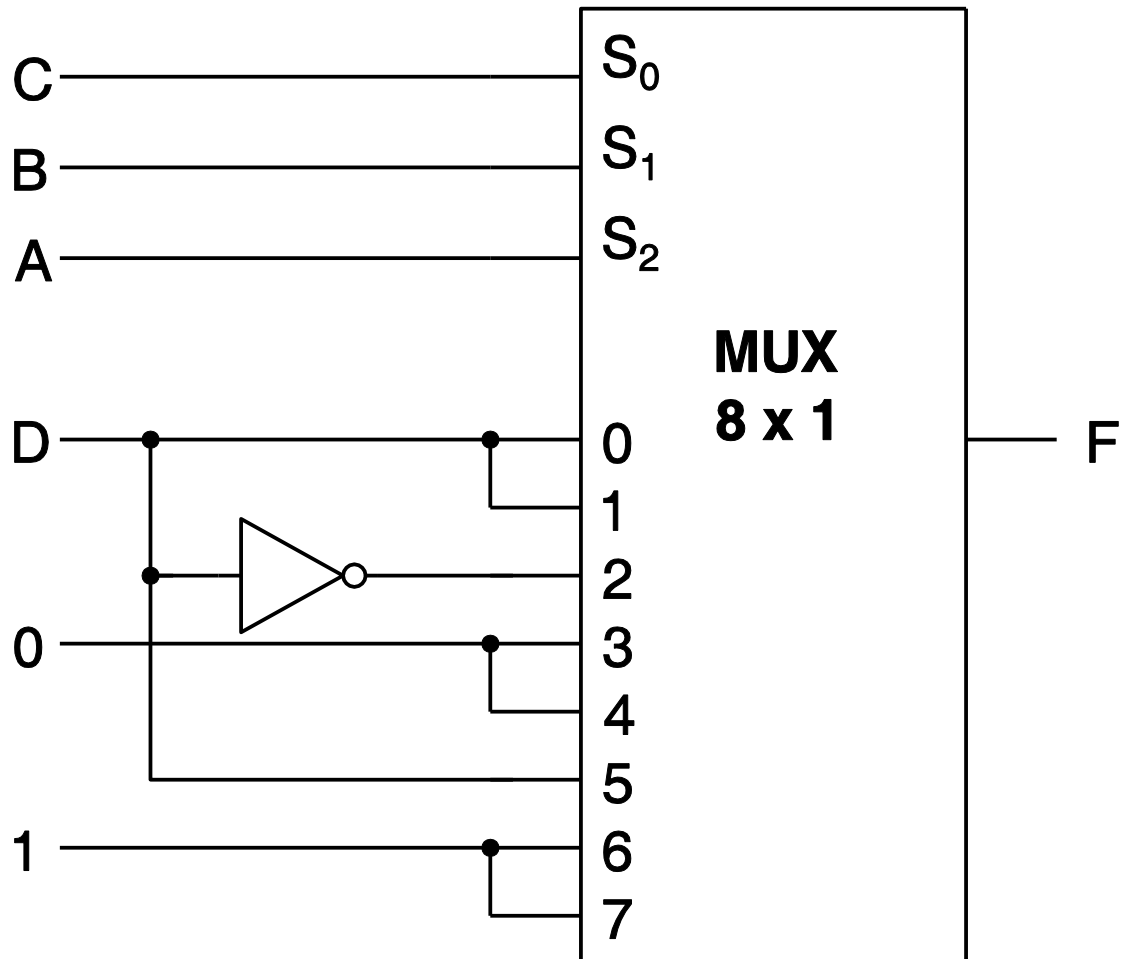
Implementazione con Multiplexer

Realizzazione di Circuiti Combinatori con Multiplexer

Tabella di verità

	A	B	C	D	F	
0	0	0	0	0	0	F = D
	0	0	0	1	1	
1	0	0	1	0	0	F = D
	0	0	1	1	1	
2	0	1	0	0	1	— F = D
	0	1	0	1	0	
3	0	1	1	0	0	F = 0
	0	1	1	1	0	
4	1	0	0	0	0	F = 0
	1	0	0	1	0	
5	1	0	1	0	0	F = D
	1	0	1	1	1	
6	1	1	0	0	1	F = 1
	1	1	0	1	1	
7	1	1	1	0	1	F = 1
	1	1	1	1	1	

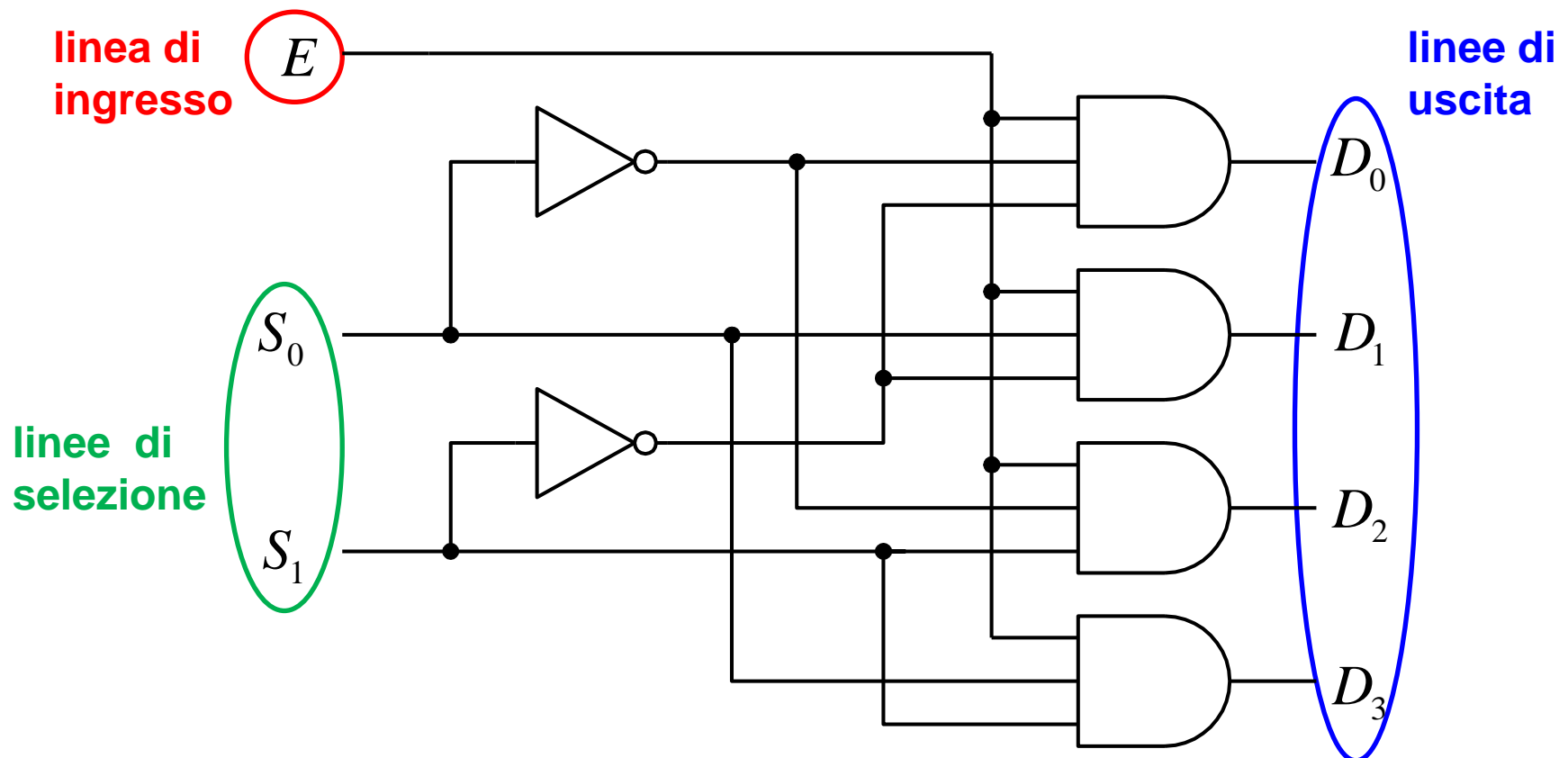
$$F(A, B, C, D) = \sum_{ABCD} m(1, 3, 4, 11, 12, 13, 14, 15)$$



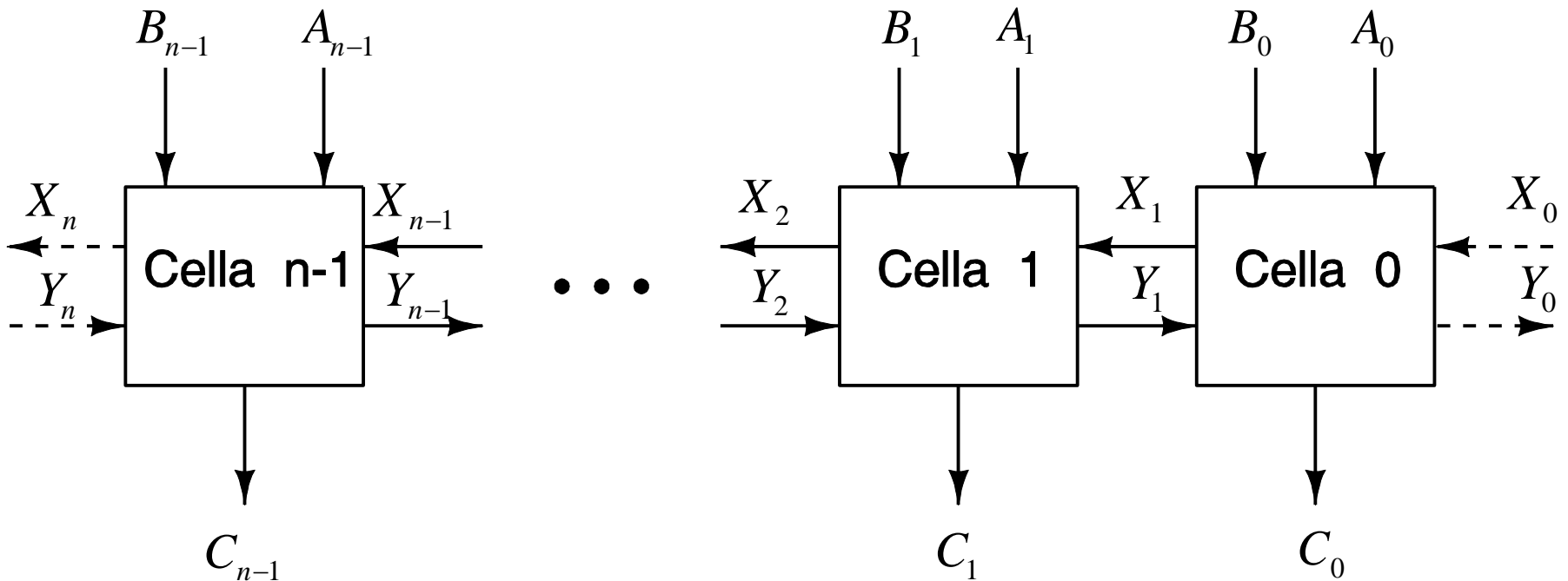
Demultiplexer

Un **demultiplexer (distributore)** è un circuito combinatorio che esegue l'operazione inversa rispetto a quella svolta da un multiplexer, ovvero riceve informazioni da una singola linea e le trasmette a una delle possibili 2^n linee di uscita

Esempio: Demultiplexer 1x4



Circuiti Combinatori Iterativi



**... possono essere visti come
circuiti combinatori iterativi !**

Blocco Funzionale: Half-Adder

- Un sommatore binario a singolo bit con 2 ingressi che esegue le seguenti operazioni:

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- Un half adder somma due bit per produrre una somma a due bit
- La somma è espressa come un bit somma, S ed un bit riporto, C
- L' half adder può essere definito dalla tabella di verità (per S e C) \Rightarrow

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half-Adder: Semplificazione Logica

- Le K-Map per S e C sono:

S

		X	
		0	1
Y	0		1
	1	1	

C

		X	
		0	1
Y	0		
	1		1

- Banalmente si deriva:

$$S = X \cdot \bar{Y} + \bar{X} \cdot Y = X \oplus Y$$

$$S = (X + Y) \cdot (\bar{X} + \bar{Y})$$

- e

$$C = X \cdot Y$$

$$C = \overline{\overline{X \cdot Y}}$$

- Da queste equazioni si deducono differenti implementazioni.

Half-Adder: Differenti Implementazioni

- Possiamo derivare I seguenti 5 insiemi di equazioni per un half-adder:

$$(a) \quad S = X \cdot \bar{Y} + \bar{X} \cdot Y = X \oplus Y \\ C = X \cdot Y$$

$$(d) \quad S = (X + Y) \cdot \bar{C} \\ \bar{C} = (\bar{X} + \bar{Y})$$

$$(b) \quad S = (X + Y) \cdot (\bar{X} + \bar{Y}) \\ C = X \cdot Y$$

$$(e) \quad S = X \oplus Y \\ C = X \cdot Y$$

$$(c) \quad S = \overline{(C + \bar{X} \cdot \bar{Y})} \\ C = X \cdot Y$$

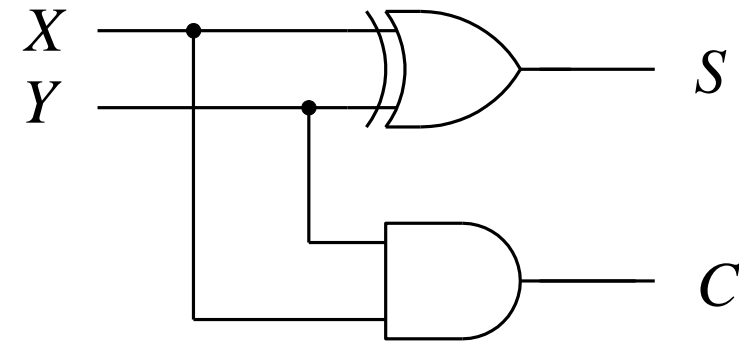
- (a), (b), ed (e) sono implementazioni SOP, POS, e XOR per S.
- In (c), la funzione C è utilizzata come un termine nell'implementazione AND-NOR di S; in (d), la funzione \bar{C} è utilizzata in un termine POS di S.

Half-Adder: Circuiti Logici

- L'implementazione più comune di un half adder è:

$$S = X \oplus Y$$

$$C = X \cdot Y$$

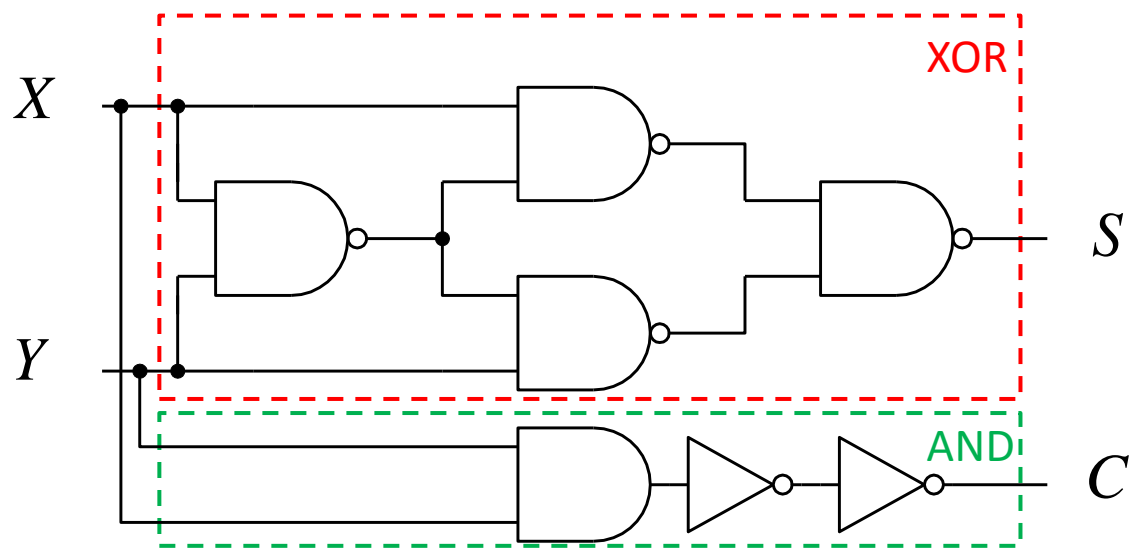


- Un'altra implementazione a sole porte NAND è:

$$S = X \oplus Y = X \cdot \bar{Y} + \bar{X} \cdot Y = X \cdot (\bar{X} + \bar{Y}) + Y \cdot (\bar{X} + \bar{Y}) = X \cdot (\overline{X \cdot Y}) + Y \cdot (\overline{X \cdot Y})$$

$$= \overline{\overline{X \cdot (\overline{X \cdot Y})}} \cdot \overline{\overline{Y \cdot (\overline{X \cdot Y})}}$$

$$C = \overline{\overline{X \cdot Y}}$$

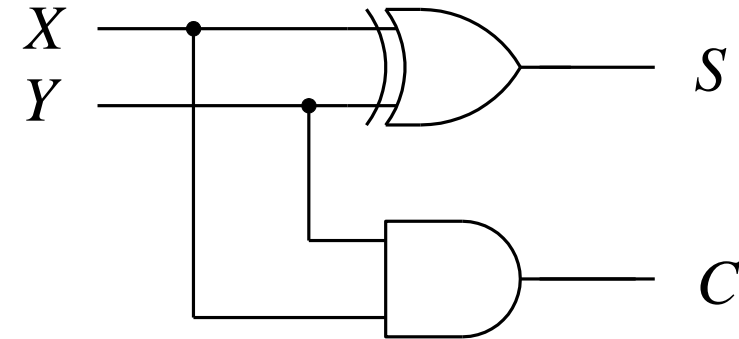


Half-Adder: Circuiti Logici

- L'implementazione più comune di un half adder è:

$$S = X \oplus Y$$

$$C = X \cdot Y$$

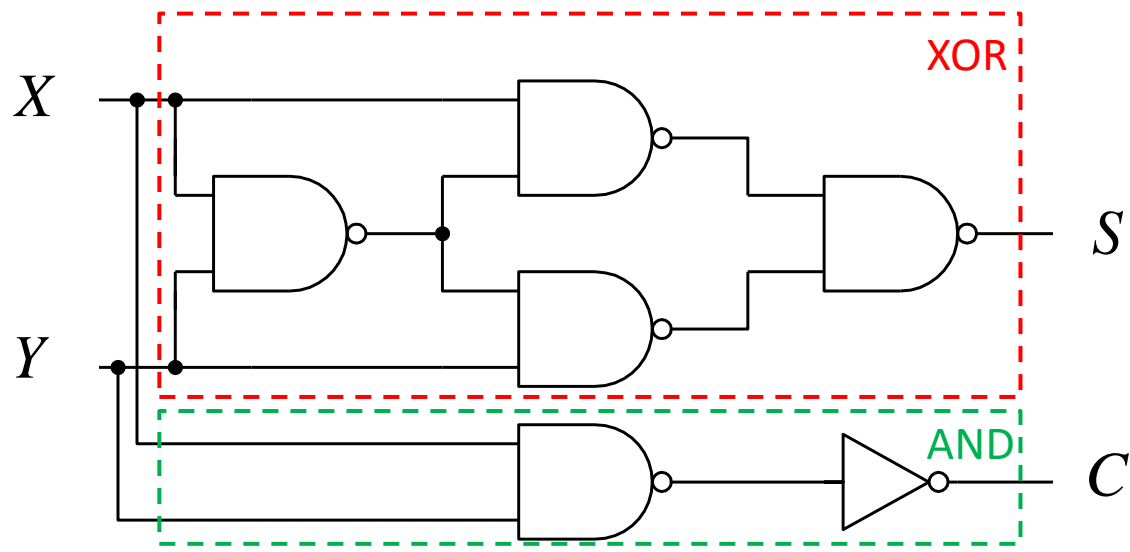


- Un'altra implementazione a sole porte NAND è:

$$S = X \oplus Y = X \cdot \bar{Y} + \bar{X} \cdot Y = X \cdot (\bar{X} + \bar{Y}) + Y \cdot (\bar{X} + \bar{Y}) = X \cdot (\overline{X \cdot Y}) + Y \cdot (\overline{X \cdot Y})$$

$$= \overline{\overline{X \cdot (\overline{X \cdot Y})}} \cdot \overline{\overline{Y \cdot (\overline{X \cdot Y})}}$$

$$C = \overline{\overline{X \cdot Y}}$$

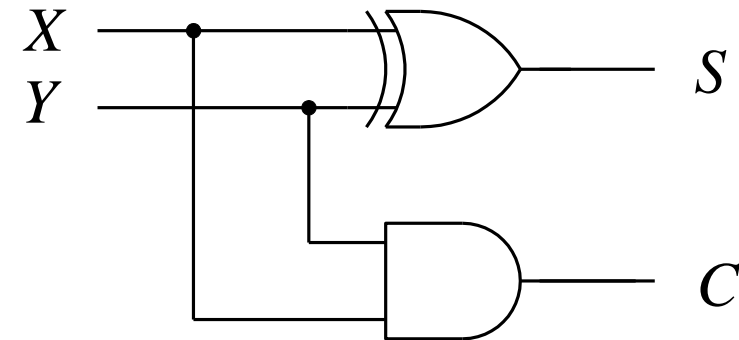


Half-Adder: Circuiti Logici

- L'implementazione più comune di un half adder è:

$$S = X \oplus Y$$

$$C = X \cdot Y$$

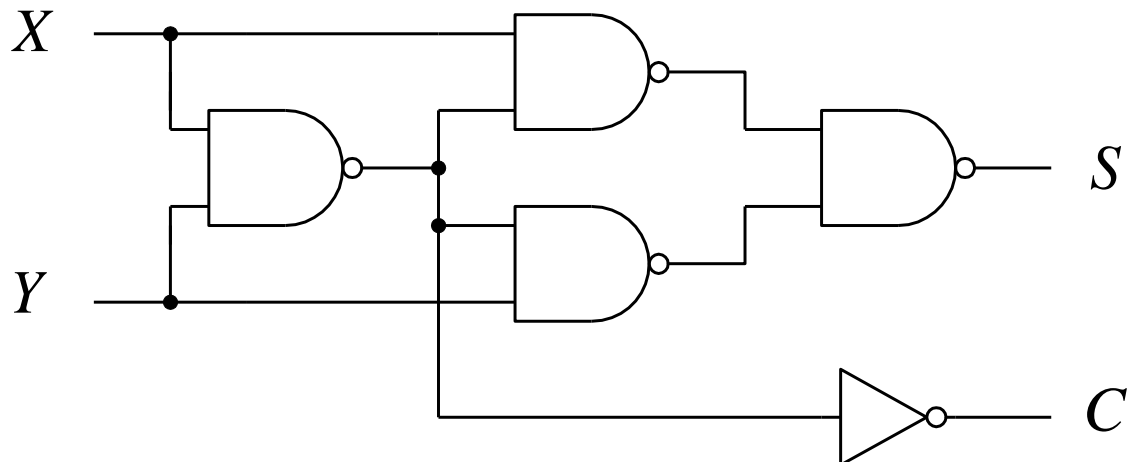


- Un'altra implementazione a sole porte NAND è:

$$S = X \oplus Y = X \cdot \bar{Y} + \bar{X} \cdot Y = X \cdot (\bar{X} + \bar{Y}) + Y \cdot (\bar{X} + \bar{Y}) = X \cdot (\overline{X \cdot Y}) + Y \cdot (\overline{X \cdot Y})$$

$$= \overline{\overline{X \cdot (\overline{X \cdot Y})}} \cdot \overline{\overline{Y \cdot (\overline{X \cdot Y})}}$$

$$C = \overline{X \cdot Y}$$



Blocco Funzionale: Full Adder

- Il full adder è simile all'half adder, eccetto che include il bit di riporto (carry-in) Z. Come l'half-adder, calcola un bit di somma, S, e un bit di riporto (carry-out), C.

- Per un carry-in (Z) di 0
è come per l'half-adder:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- Per un carry-in (Z) di 1:

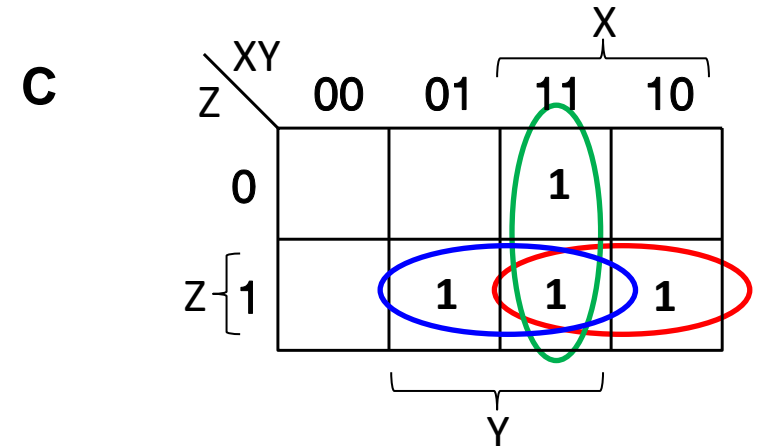
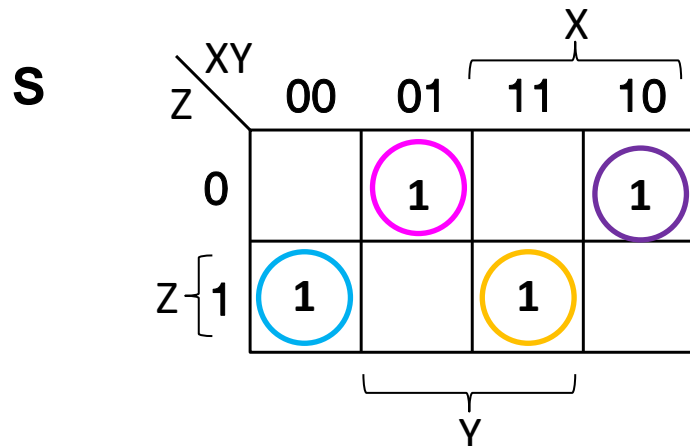
Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

Full Adder: Ottimizzazione Logica

- Tabella di Verità del Full-Adder :

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Mappe di Karnaugh del Full-Adder :



- Dalle K-Map:

$$S = X \cdot \bar{Y} \cdot \bar{Z} + \bar{X} \cdot Y \cdot \bar{Z} + \bar{X} \cdot \bar{Y} \cdot Z + X \cdot Y \cdot Z$$

$$C = X \cdot Y + X \cdot Z + Y \cdot Z$$

- La funzione S è una funzione XOR a 3 bit (Funzione dispari):

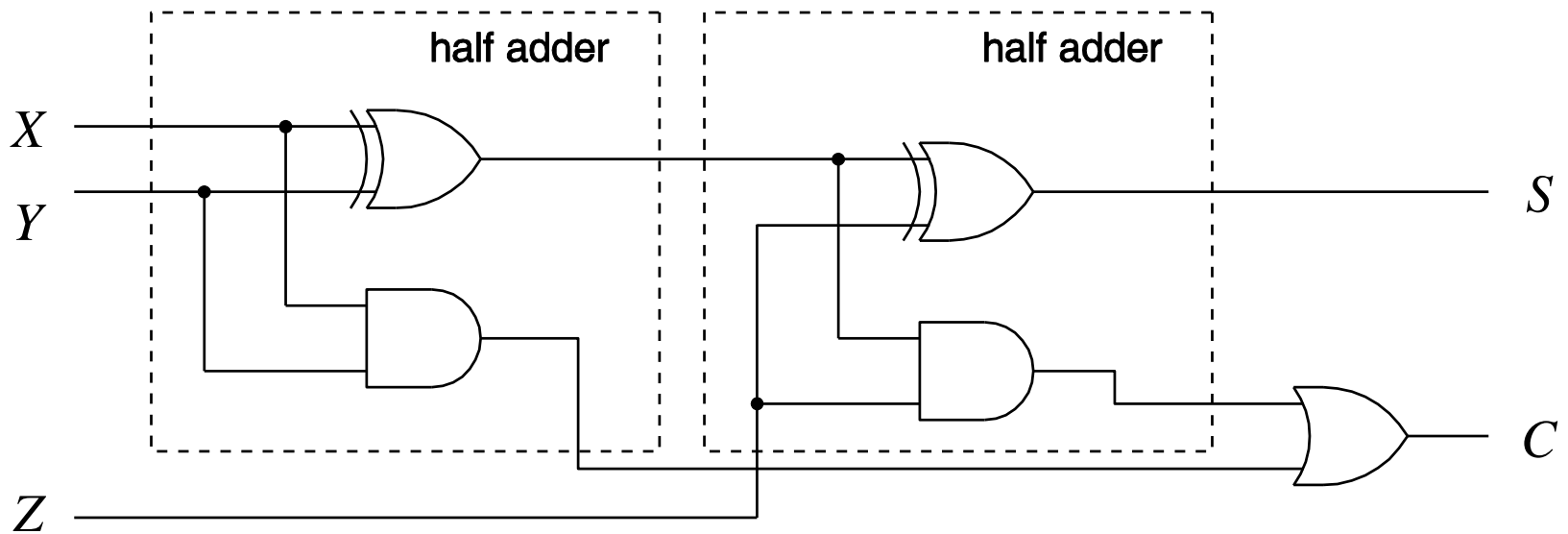
$$S = X \oplus Y \oplus Z$$

- Il bit di riporto C è 1 se sia X che Y sono 1 (la somma è 2), oppure se la somma è 1 e si ha un carry-in (Z) di 1 . Pertanto C si può riscrivere come:

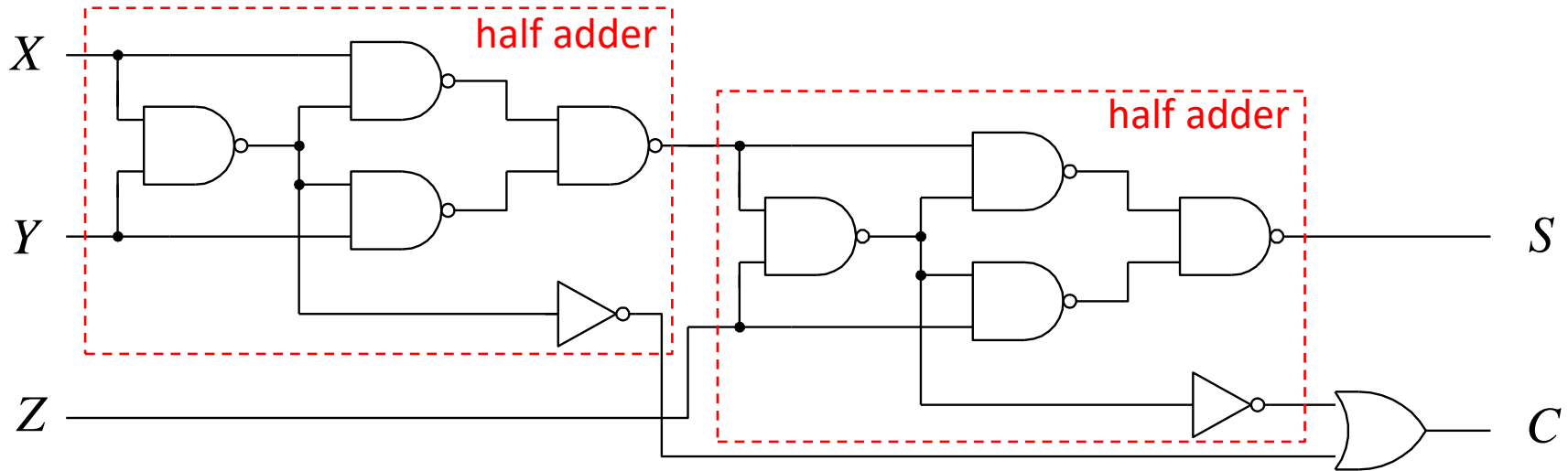
$$C = X \cdot Y + (X \oplus Y) \cdot Z$$

- Il termine $X \cdot Y$ è il *carry generate*.
- Il termine $X \oplus Y$ è il *carry propagate*.

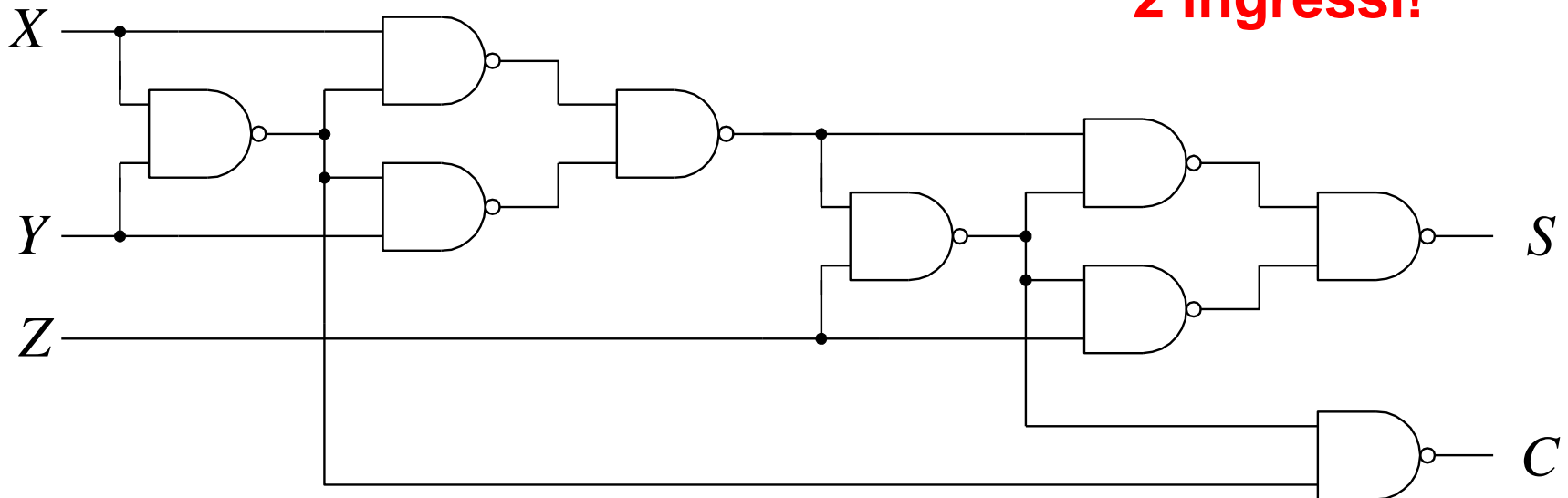
Full Adder: Circuito Logico



Full Adder: Circuito Logico a NAND

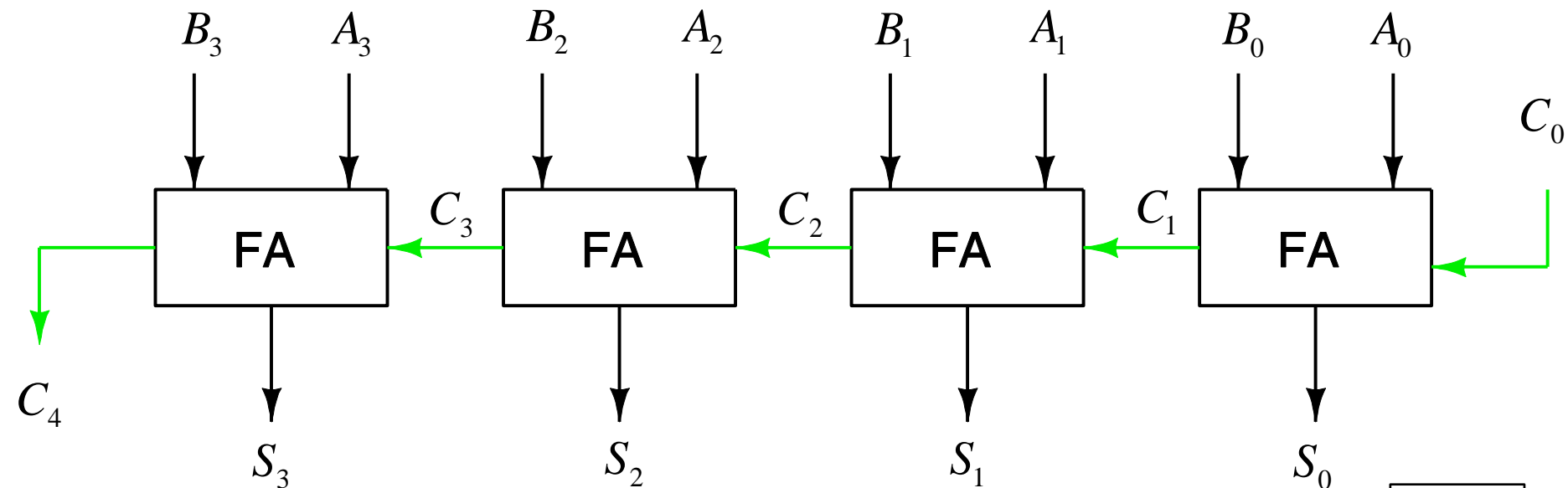


**9 porte NAND a
2 ingressi!**

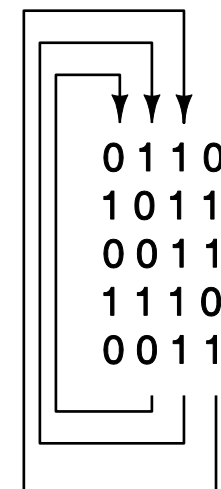


Circuito Sommatore Binario con Riporto in Cascata

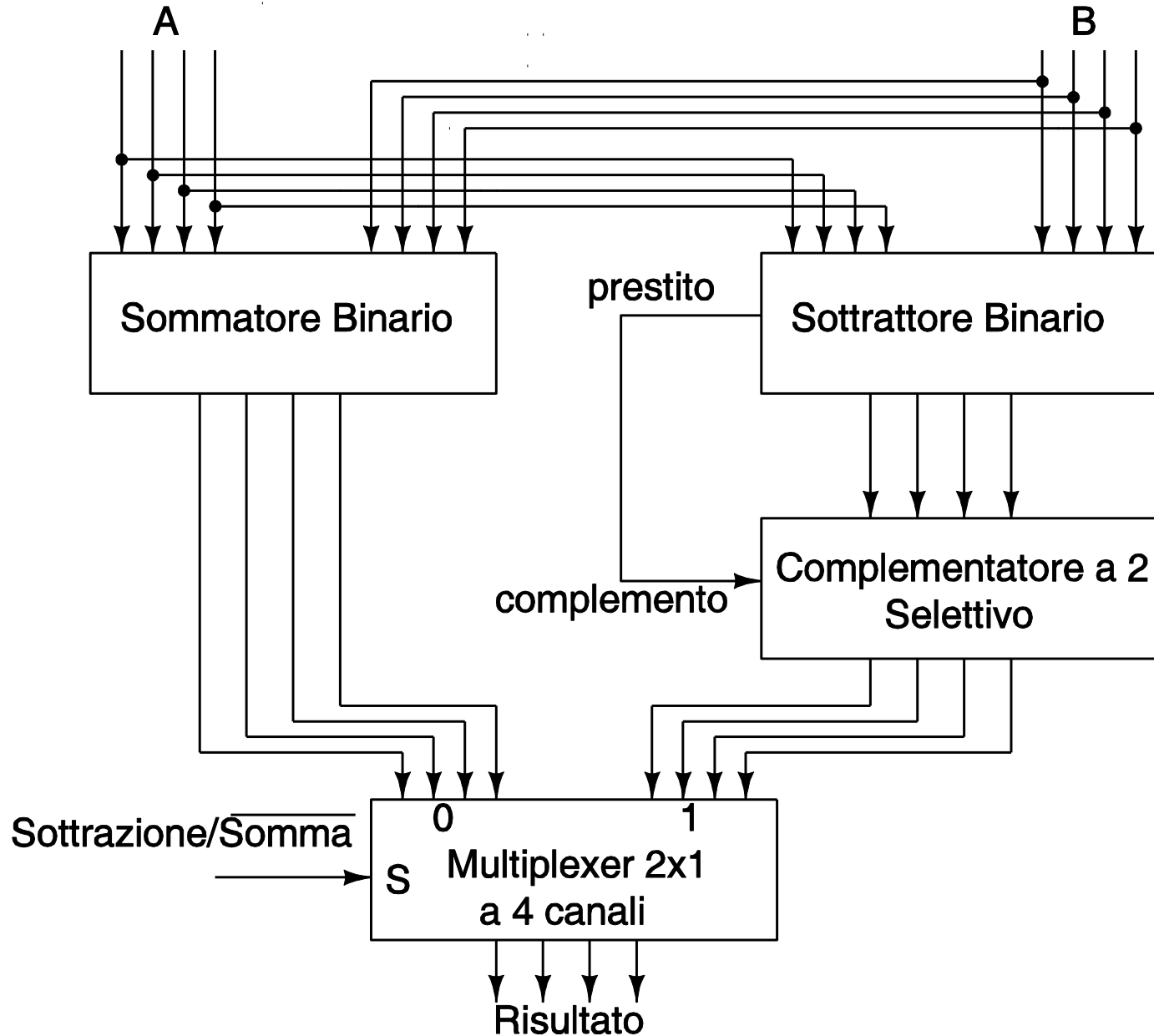
Sommatore a 4-bit con riporto in cascata (4-bit Ripple Carry Adder) realizzato con quattro Full Adder:



Riporto in ingresso
Addendo A
Addendo B
Somma S
Riporto in uscita



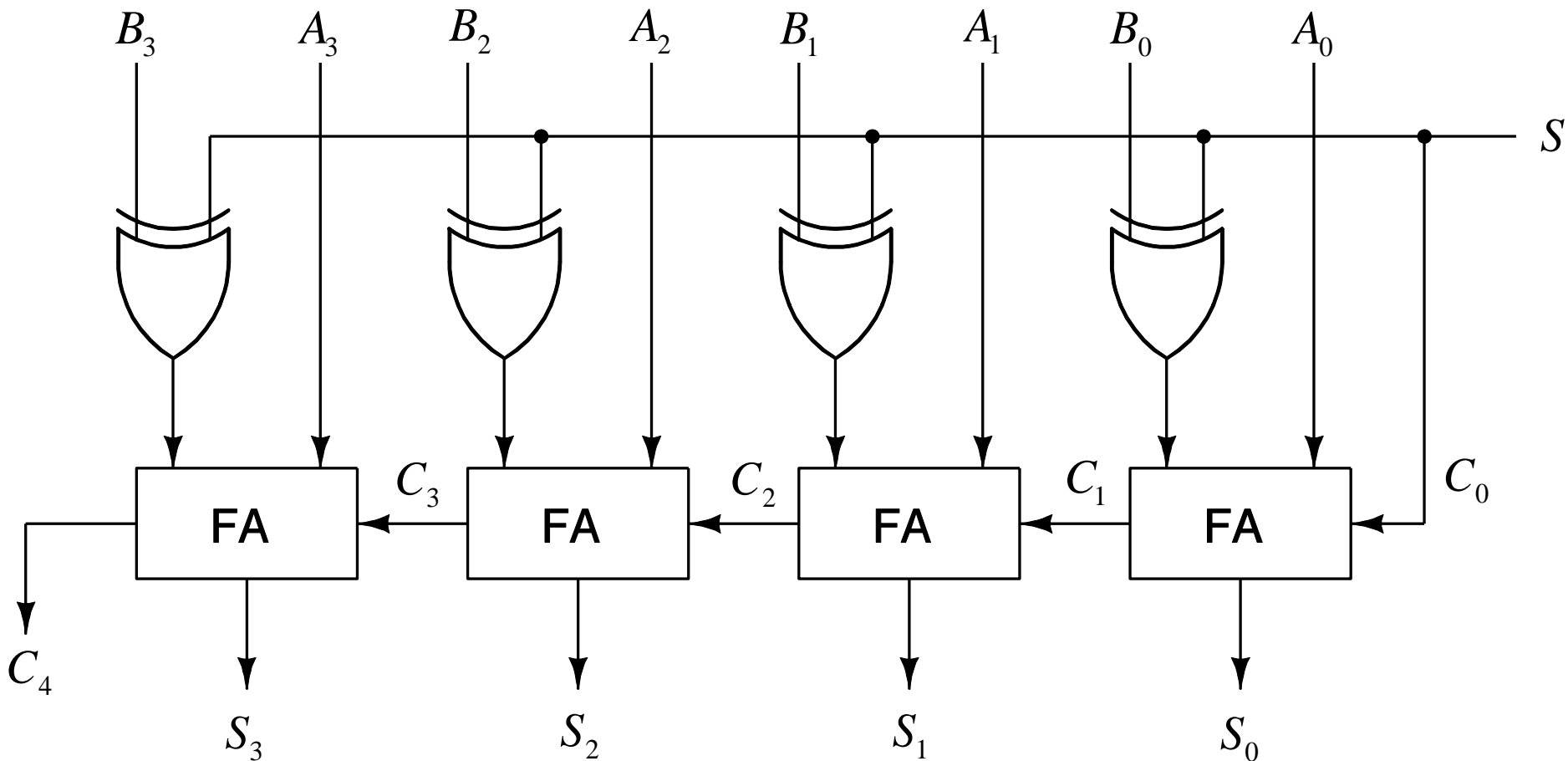
Sommatore/Sottrattore Binario



Sommatore / Sottrattore Binario

$$S = 0 \longrightarrow A + B$$

$$S = 1 \longrightarrow A - B$$



Numeri Negativi

Decimale	Complemento a 2 con segno	Complemento a 1 con segno	Modulo e segno
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	----	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	----	----

Overflow (Addizione Compl. a 2)

Nell'addizione in complemento a 2, si ha overflow quando la somma eccede il range consentito:

ciò avviene se il segno dei due addendi è lo stesso, e se il segno della somma è differente dal segno degli addendi.

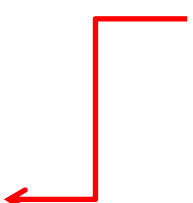
$$\begin{array}{rcl} -7 & 1001 & \\ + -6 & +1010 & \\ \hline -13 & +3 = \overline{0011} & \end{array}$$

$$\begin{array}{rcl} +5 & 0101 & \\ + +6 & +0110 & \\ \hline +11 & -5 = \overline{1011} & \end{array}$$

Overflow (Sottrazione Compl. a 2)

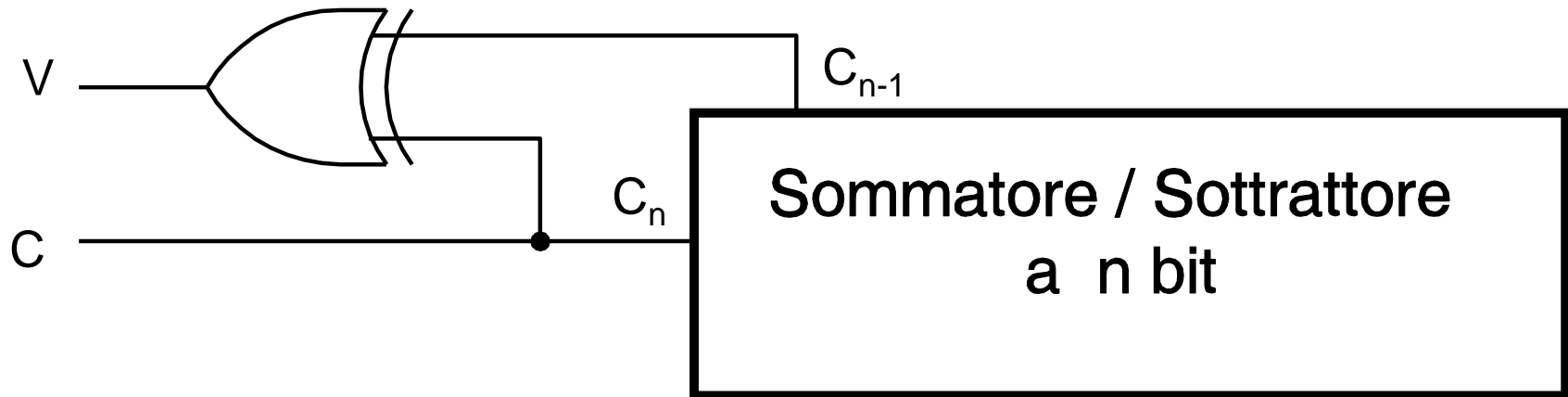
L'overflow nella sottrazione di numeri in complemento a 2 si può rilevare esaminando il segno del minuendo e del sottraendo complementato usando la stessa regola dell'addizione.

$$\begin{array}{r} 1 \leftarrow C_{in} \\ - 6 \qquad 1010 \qquad 1010 \\ + 7 \qquad - 0111 \qquad + 1000 \\ \hline = -13 \qquad \qquad 0011 = 3 \end{array}$$

1 

Circuito di Overflow

Nei calcolatori il numero di bit per rappresentare i numeri è fisso: bisogna segnalare la condizione di overflow!



Numeri senza segno:

$C = 1$ overflow per la somma (la sottrazione non necessita di correzione)

$C = 0$ correzione per la sottrazione (la somma non presenta overflow)

Numeri con segno:

$V = 0$ nessun overflow per la somma (o la sottrazione)

$V = 1$ overflow per la somma (o la sottrazione)

Moltiplicazione: Scorrimento e Somma

$$\begin{array}{r} 9 \\ \times 11 \\ \hline 99 \end{array}$$

$$\begin{array}{r} 1001 \\ \times 1011 \\ \hline 1001 \\ 1001 \\ 0000 \\ 1001 \\ \hline 1100011 \end{array}$$

$$\begin{array}{r} 1001 \\ \times 1011 \\ \hline 01001 \\ 10010 \\ \hline 011011 \\ 000000 \\ \hline 0011011 \\ 1001000 \\ \hline 1100011 \end{array}$$

Overflow (Addizione Compl. a 2)

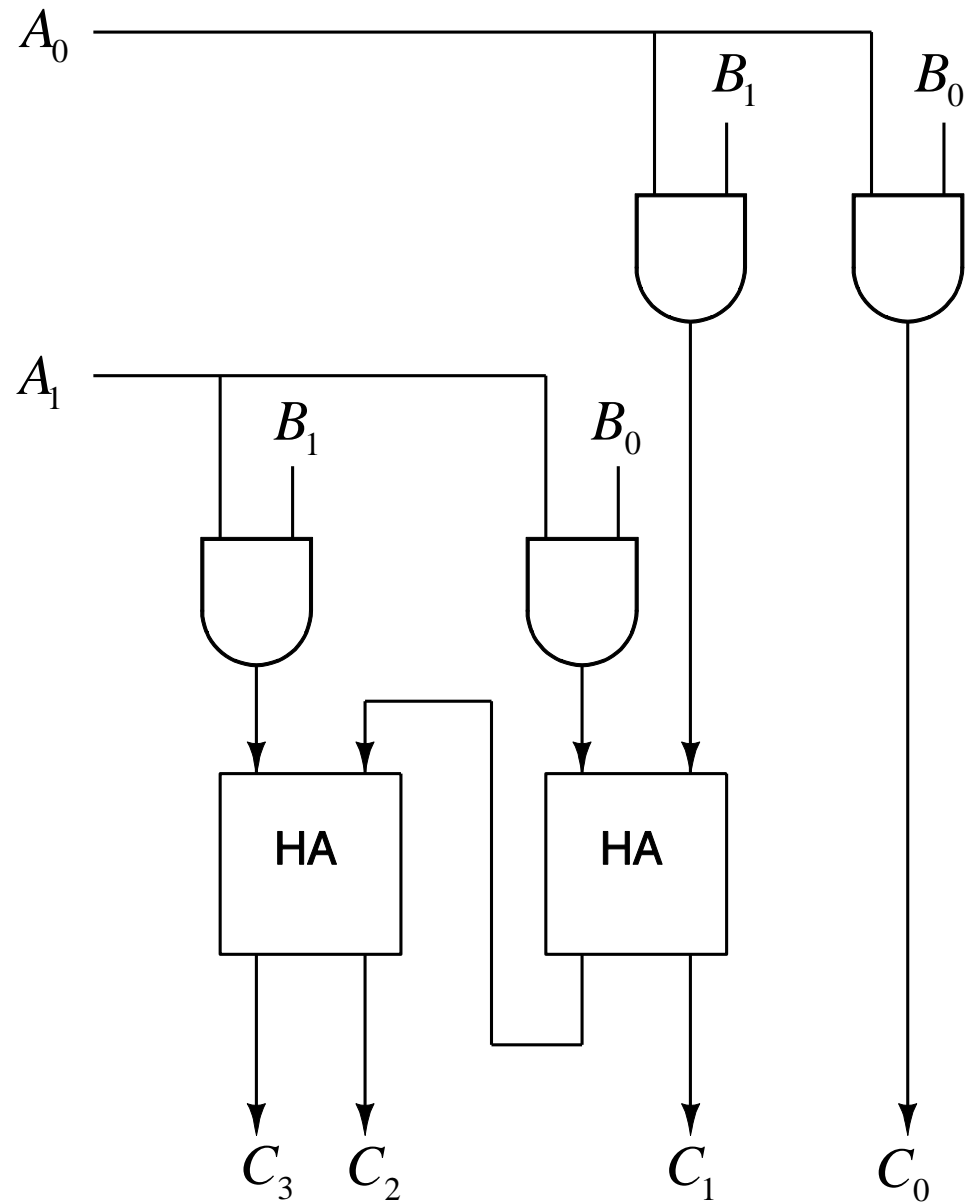
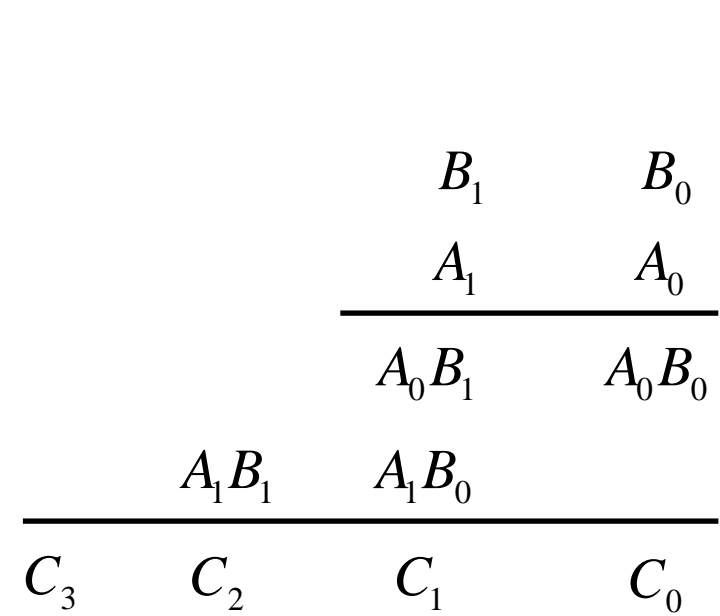
Nell'addizione in complemento a 2, si ha overflow quando la somma eccede il range consentito:

ciò avviene se il segno dei due addendi è lo stesso, e se il segno della somma è differente dal segno degli addendi.

$$\begin{array}{r} -5 \quad 1011 \\ + -6 \quad + 1010 \\ \hline = -11 \quad +5 = 0101 \end{array}$$

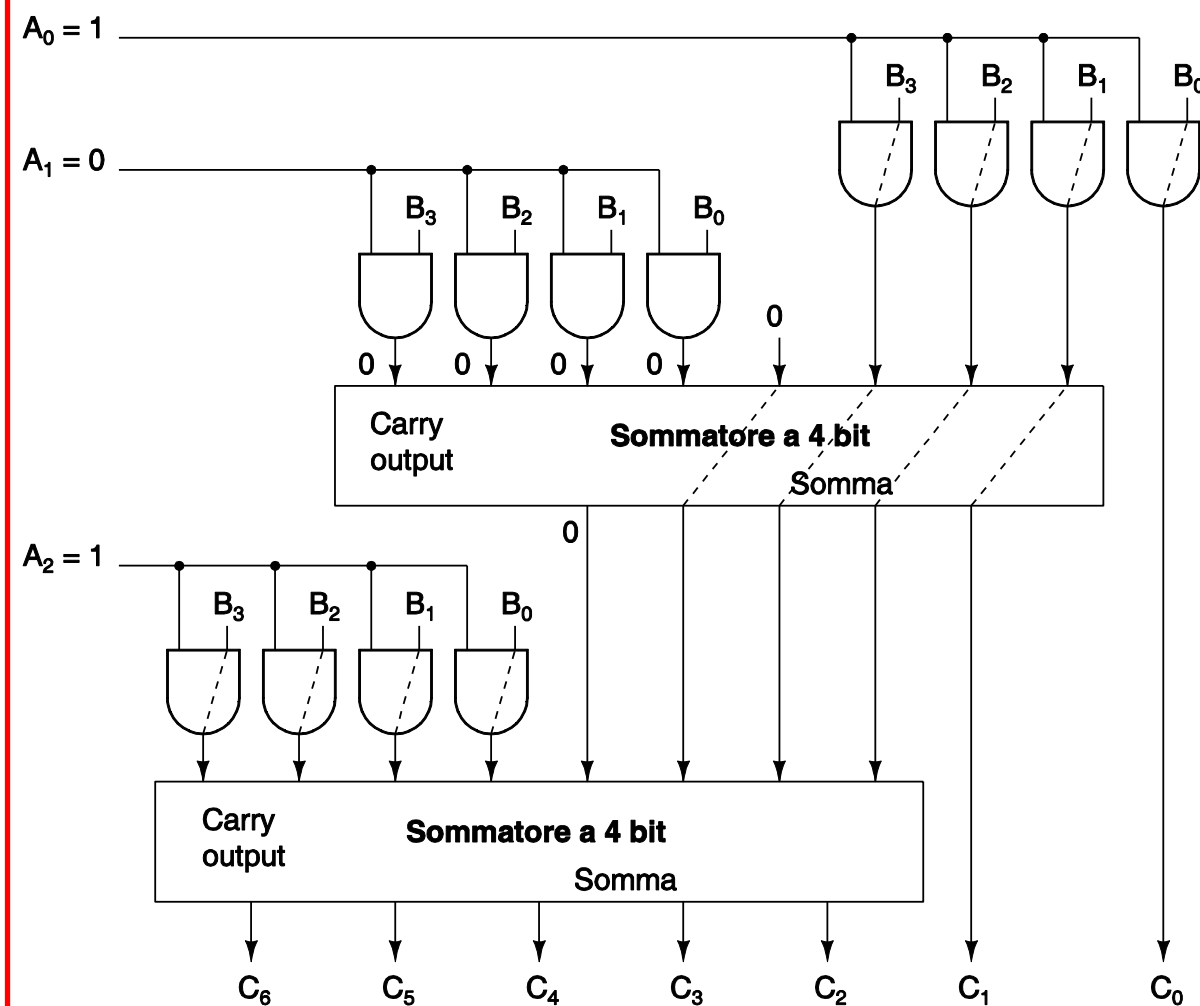
$$\begin{array}{r} +4 \quad 0100 \\ + +5 \quad + 0101 \\ \hline = +9 \quad -7 = 1001 \end{array}$$

Moltiplicatore Binario

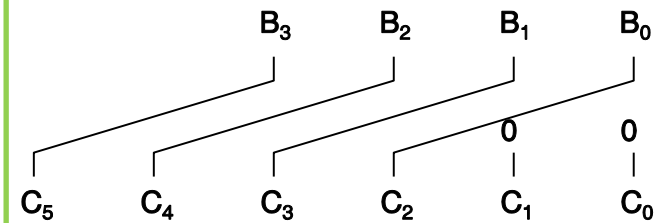


Moltiplicatore Binario

101 x B



100 x B



B : 100

