Funzioni Pari e Dispari

- L'espressione booleana

$$E_n = A_1 \oplus A_2 \oplus \ldots \oplus A_n.$$

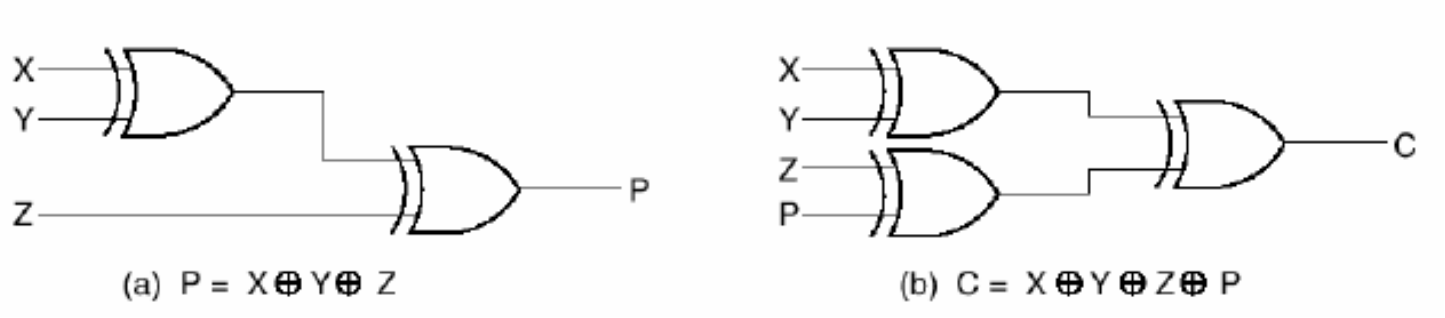
  vale 1 solo quando il numero di variabili cui si assegna il valore 1 è dispari.
- La funzione di verità catturata dall'espressione E_n è detta **funzione dispari**.

Funzioni Pari e Dispari

- Se si osserva la mappa di Karnaugh per la funzione dispari, si nota subito che la sua implementazione in somma di prodotti (o in prodotto di somme) risulta estremamente inefficiente.
  - Serve una quantità esponenziale di porte AND, OR e NOT.
  - Utilizzando la porta XOR, basta invece una quantità lineare di porte.
- Il complemento della funzione dispari è detto **funzione pari**.
- Le funzioni per pari e dispari risultano molto utili per la generazione e il controllo di parità.

Controllo di parità

- Il circuito che permette l'aggiunta del bit di parità prende il nome di generatore di parità (parity generator)
- Il circuito che permette il controllo di parità prende il nome parity checker
- Usando parità pari, si ha un errore se il parity checker genera 1 in uscita (perché un XOR di più variabili è 1 se il numero di bit 1 è dispari).



Multiplexer

- Un **multiplexer** è una rete combinatoria con:
  - 2^n ingressi per i dati.
  - n ingressi di controllo.
  - Una singola uscita.
- Gli input di controllo permettono di **selezionare** uno tra i possibili ingressi per i dati, il cui input diventerà il valore dell'unica uscita.
  - Ad ogni sequenza di valori di verità lunga n corrisponderà un numero binario compreso tra 0 e $2^n - 1$.

Multiplexer

- Assegnando valori costanti agli ingressi per i dati, si può costruire una rete combinatoria che implementi qualunque funzione booleana di n variabili.
- Un'altra applicazione del multiplexer è la conversione di dati da parallelo a seriale.
- Il **demultiplexer** redirige il suo unico ingresso per i dati verso una delle sue 2^n uscite in base ai valori degli ingressi di controllo.

Multiplexer (4-1)

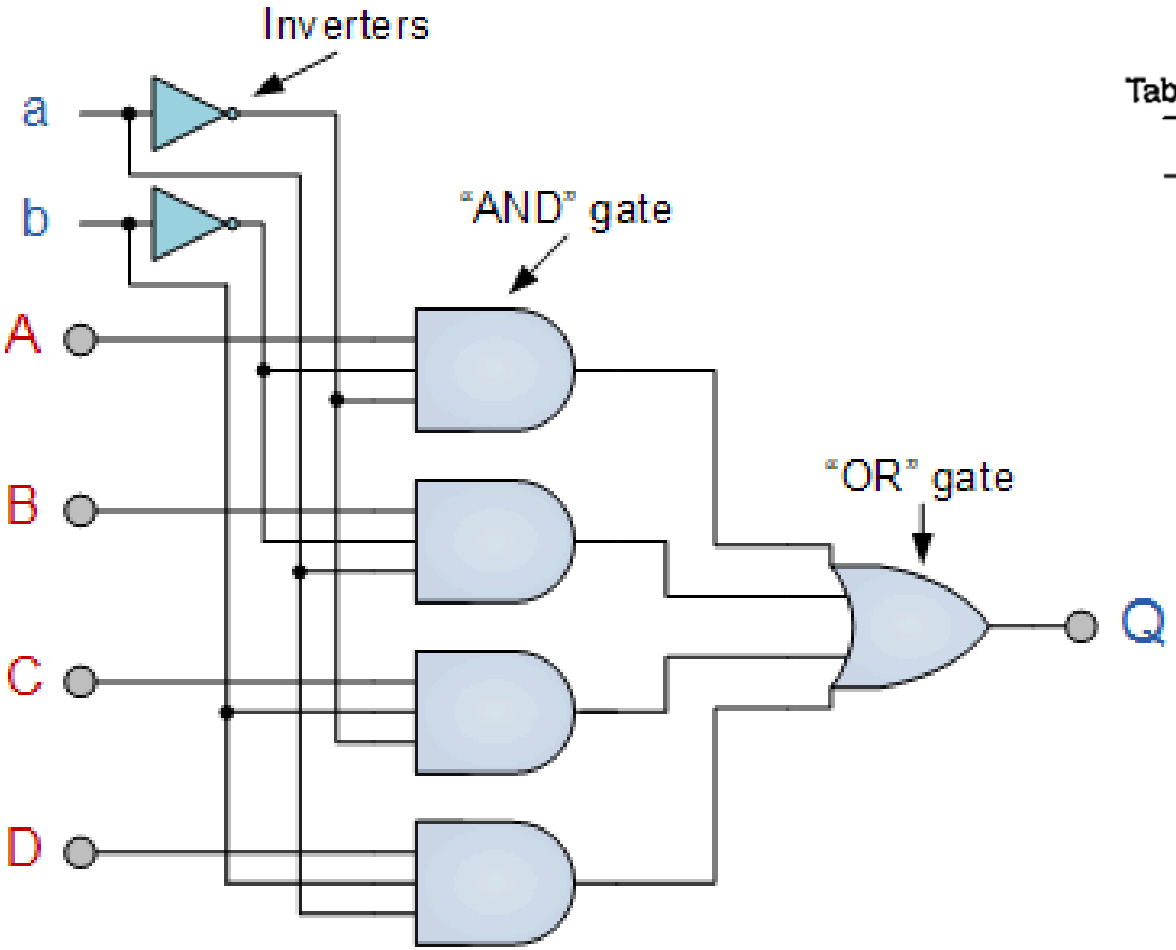
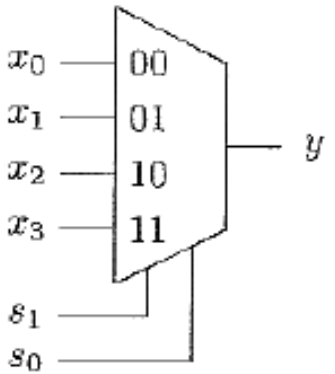


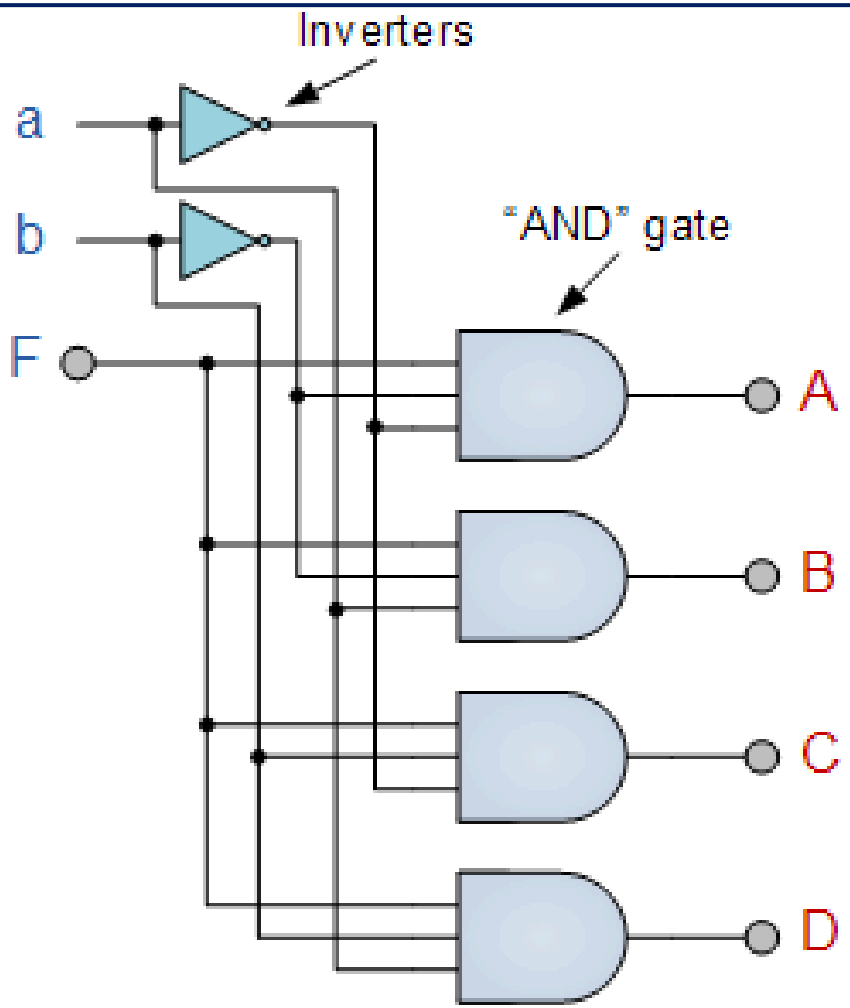
Tabella di funzionamento

S_1	S_0	Y
0	0	D_0
0	t	D_1
1	0	D_2
t	t	D_3

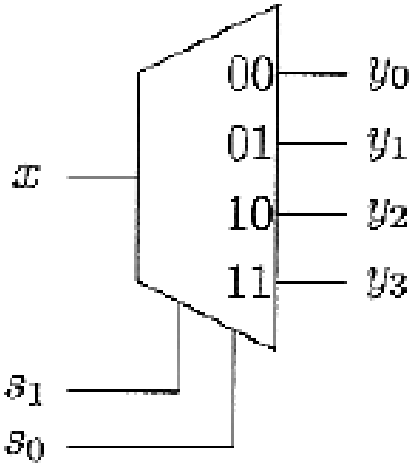


(b)

Demultiplexer (1-4)



s_0	s_1	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

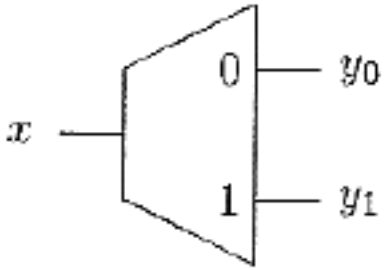


Decoder

- Un **decodificatore** è una rete combinatoria con:
  - n ingressi.
  - 2^n uscite.
- I valori in ingresso vengono utilizzati per selezionare una tra le uscite, che sarà l'unica ad essere impostata ad 1.

x	y_0	y_1
0	1	0
1	0	1

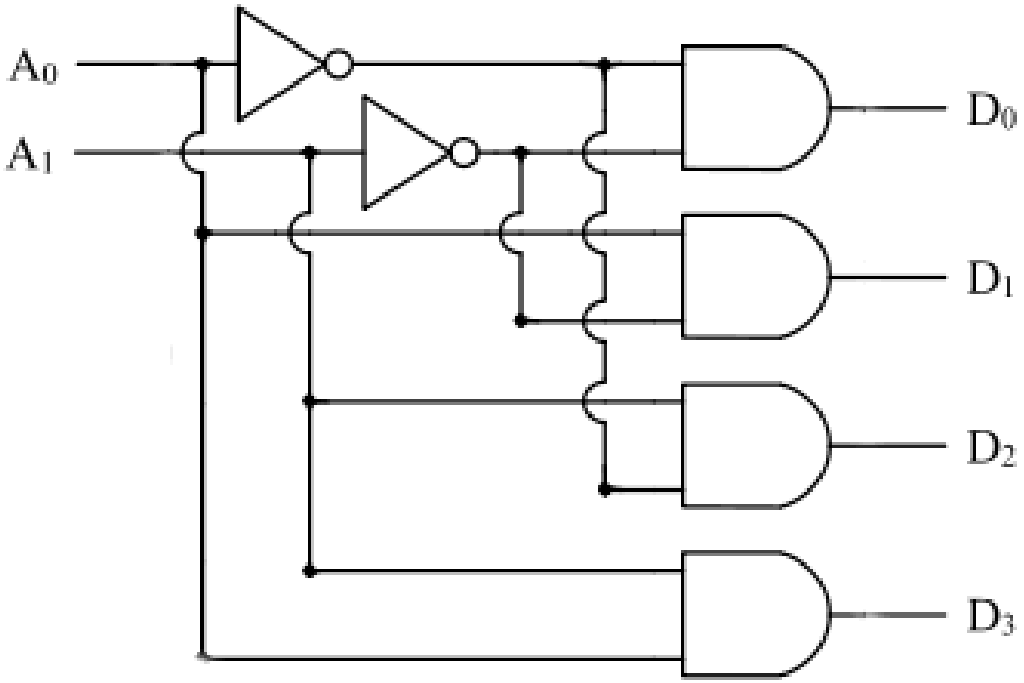
(a)



(b)

Figura 5.6 Decoder a un ingresso: tabella della verità e simbolo

Decoder



Truth Table

A_1	A_0	D_3	D_2	D_1	D_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

Equations

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

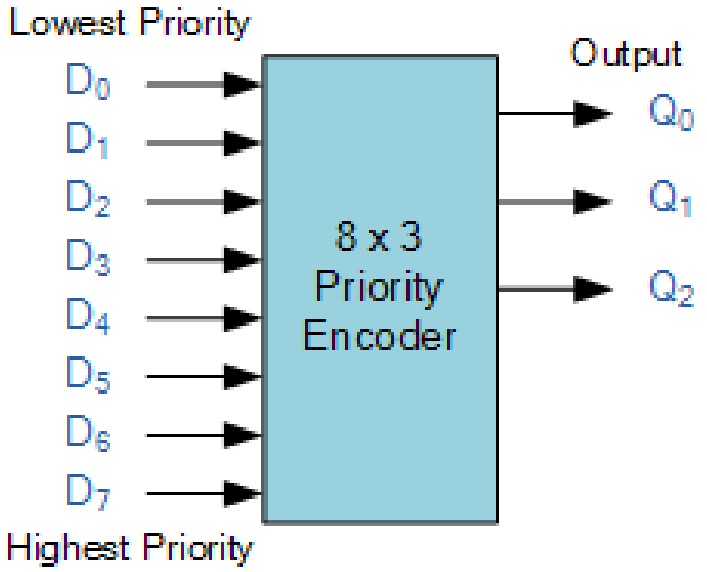
$$D_1 = \overline{A_1} \cdot A_0$$

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

$$D_3 = A_1 \cdot A_0$$

Encoder (codificatore)

- Dualmente al decodificatore avremo il **codificatore**, con 2^n ingressi e n uscite.
  - L'inerente ambiguità viene in genere risolta tramite un meccanismo a **priorità**.



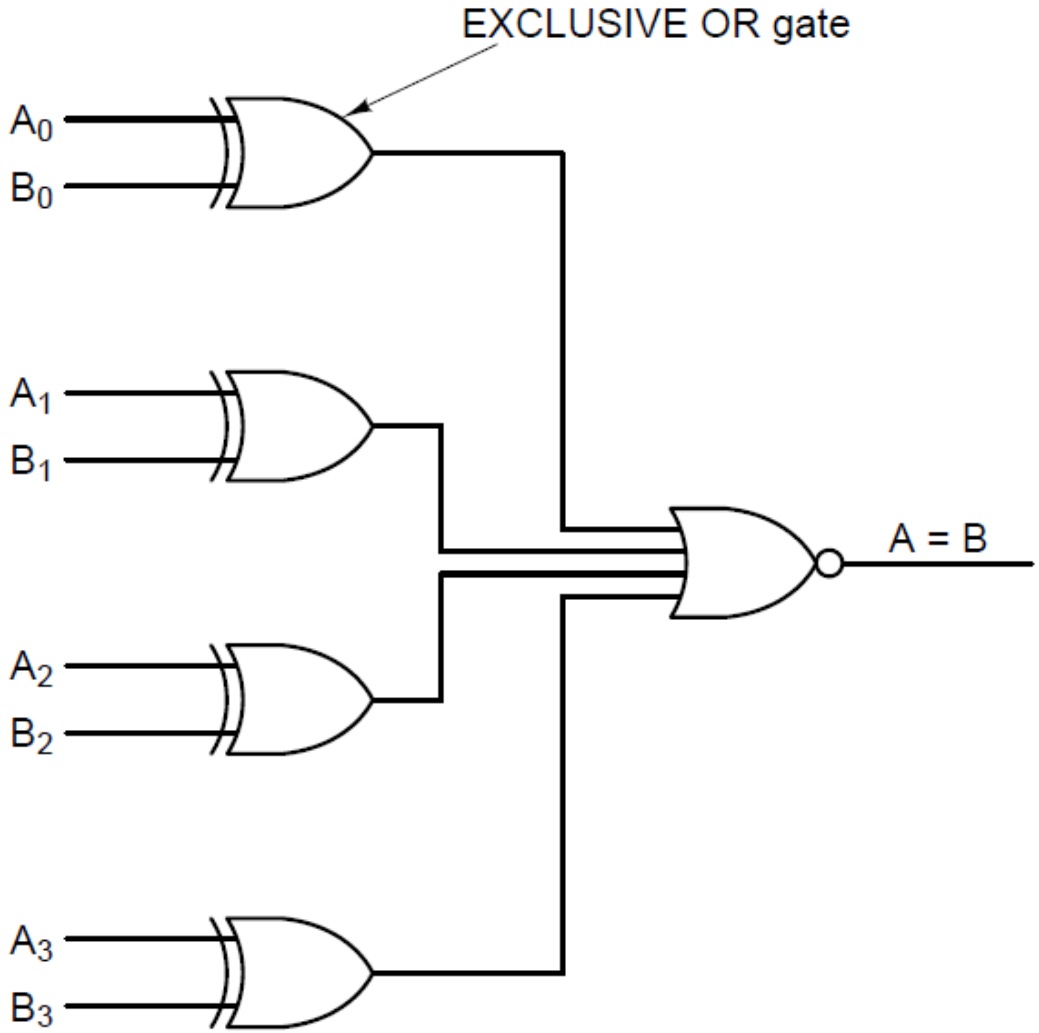
Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

Comparatore

- Un **comparatore** è una rete combinatoria con:
  - $2n$ ingressi, suddivisi in due sequenze lunghe n.
  - Un'uscita.
- Il valore dell'unica uscita sarà 1 se le due sequenze in input sono identiche, mentre sarà 0 se le due sequenze sono diverse.
  - In alternativa, si può fare in modo che l'uscita valga 0 se le due sequenze sono diverse e 1 se le due sequenze sono diverse.
- La porta logica XOR risulterà molto utile nella realizzazione del comparatore.
- Oltre all'uscita che indica se le due sequenze sono uguali, possono essere presenti **altre uscite**, che indicano se le due sequenze sono una maggiore dell'altra (qualora interpretate come numeri binari).

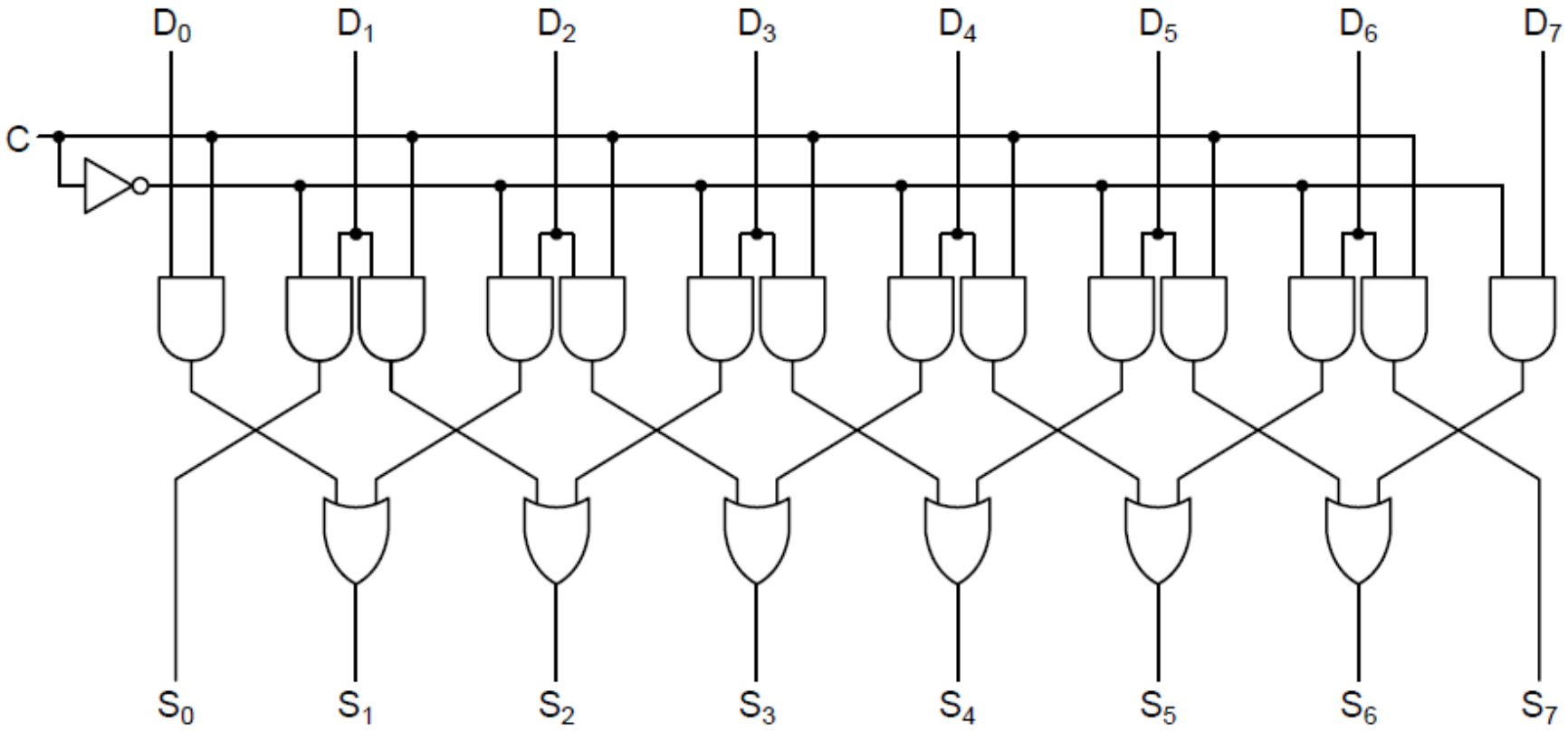
Comparatore



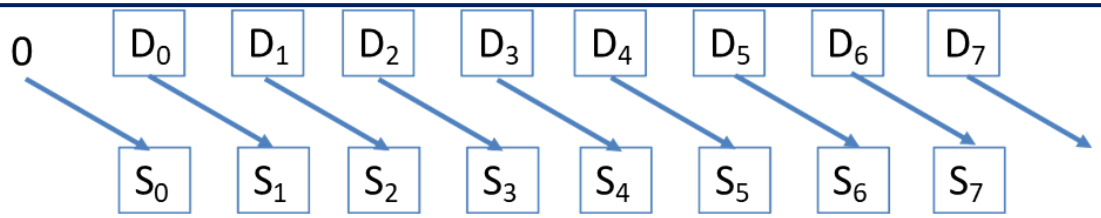
Registri a Scorrimento

- Il **Registro a Scorrimento** è una rete combinatoria con
  - *n* ingressi per i dati.
  - Un ingresso di controllo.
  - *n* uscite.
- L'idea è quella di far scorrere **verso destra** o **verso sinistra** i valori in input.
  - L'input di controllo serve proprio a determinare se lo scorrimento debba avvenire verso destra o verso sinistra.
- In ognuno dei due casi, il valore di uno tra gli ingressi andrà perso, mentre una tra le uscite assumerà un valore non proveniente dagli ingressi e fissato a priori.

Registri a Scorrimento

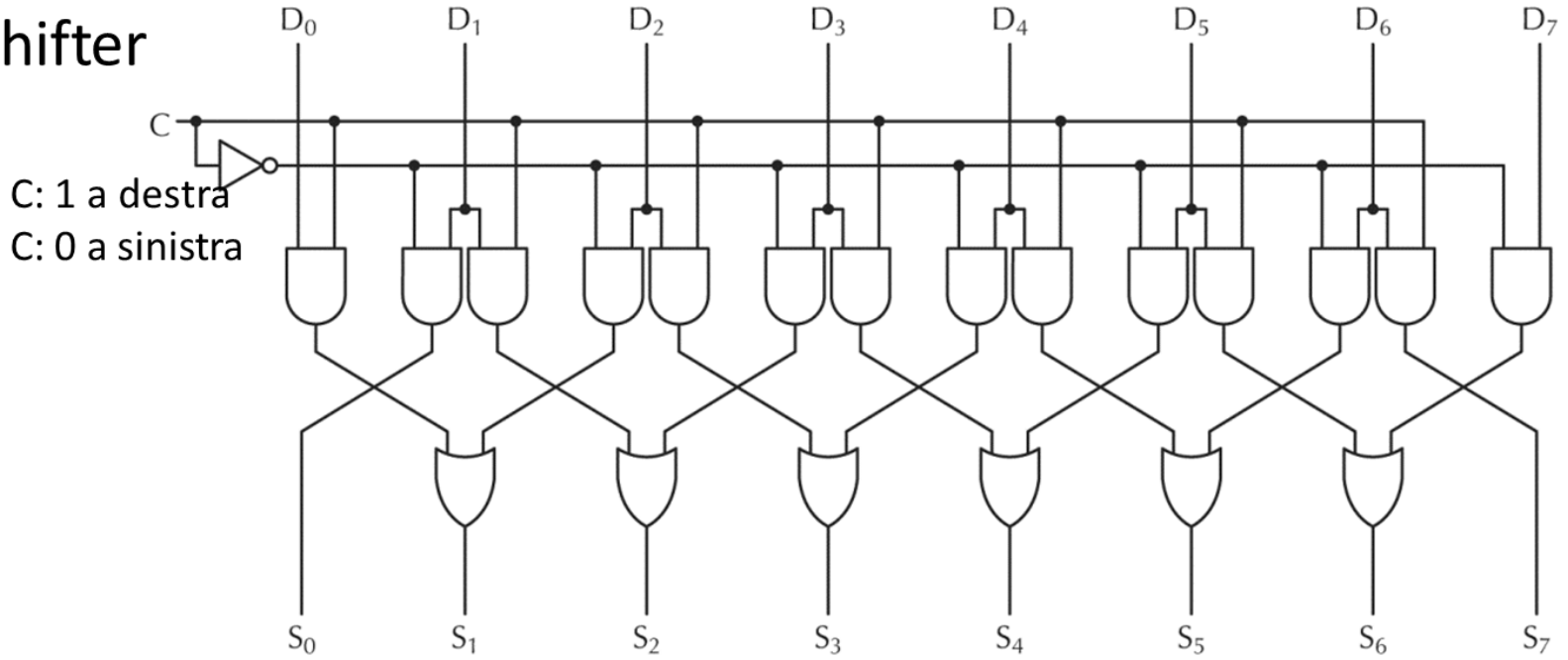


Registri a Scorrimento



CASO C = 1

Shifter



C=1 : 0 →S0, D0→S1, D1→S2,… D5→S6, D6→S7(dal più significativo verso il meno sign.)

C=0 : D1→S0, D2→S1, D3→S2, …D7→S6, 0 →S7 (dal meno significativo verso il più sign.)

Half-Adder

- Un **semisommatore** (o **half-adder**) è una rete combinatoria con:
  - Due ingressi.
  - Due uscite.
- La prima delle due uscite è valorizzata con la **somma** dei due bit in ingresso, mentre la seconda è valorizzata con il **riporto** generato dalla somma dei due bit in ingresso.
- Non è possibile utilizzare il semisommatore per calcolare la somma (e il riporto relativo al bit più significativo) di una sequenza di n bit (con $n \geq 1$).
  - In particolare, manca la possibilità di **propagare** il riporto ai bit più significativi.

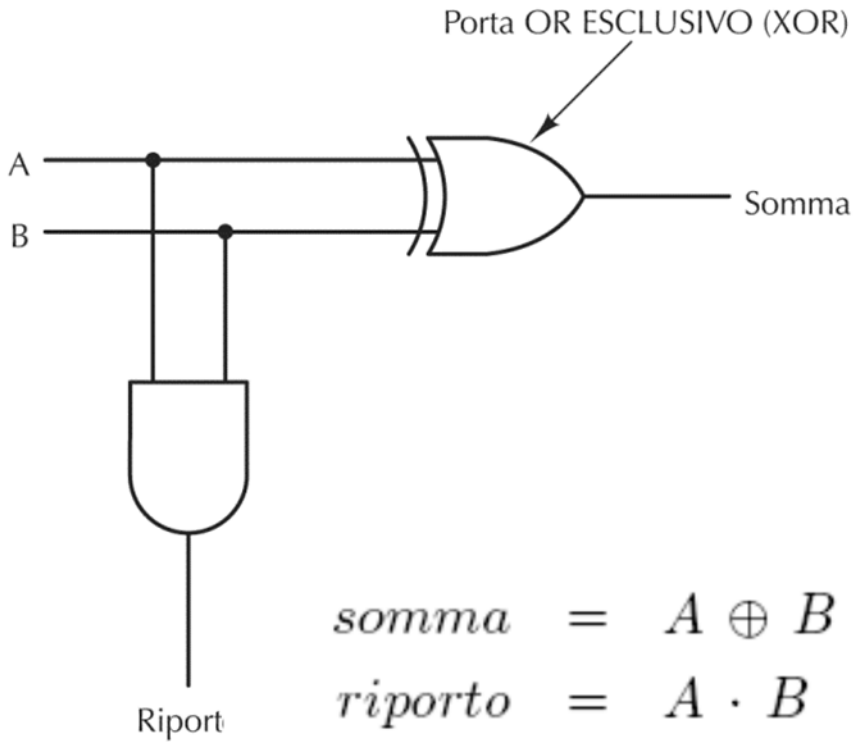
Half-Adder

Calcola somma e carry (riporto) su singoli bit

A	B	Somma	Riporto
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Somma = A xor B

Riporto = A and B



$$somma = A \oplus B$$

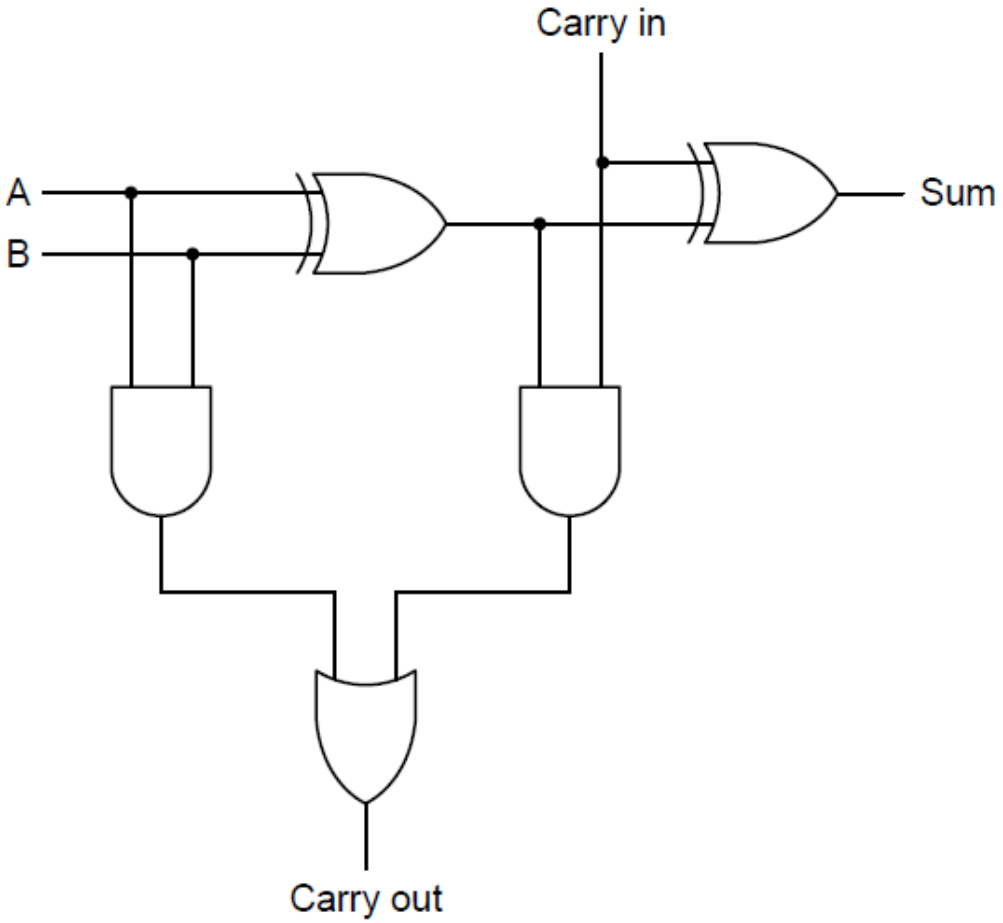
$$riporto = A \cdot B$$

Full-Adder

- Un **sommatore** (o **full-adder**) è una rete combinatoria con:
  - Tre ingressi.
  - Due uscite.
- Come nel caso del semisommatore, la prima delle due uscite è valorizzata con la **somma** dei bit in ingresso (che in questo caso sono tre, due operandi e un **riporto in entrata**), mentre la seconda è valorizzata con il **riporto** generato dalla somma dei bit in ingresso.
- È facile rendersi conto che un sommatore può essere realizzato a partire da due semisommatori.

Full-Adder

A	B	Carry in	Sum	Carry out
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



Propagazione del Riporto

- I sommatori, al contrario dei semisommatori, possono essere combinati a formare un sommatore a n bit.
  - Per ogni bit, il riporto in uscita viene utilizzato come riporto in entrata per il bit successivo.
- Parliamo in questo caso di **sommatore a propagazione di riporto**.
  - Se supponiamo che ogni porta logica aggiorni il suo output con un ritardo costante, la somma di n bit richiederà tempo proporzionale a n.
- Il ritardo diventa logaritmico in n se si utilizza un **sommatore a selezione di riporto**.
  - L'idea è quella di calcolare preventivamente le somme dei bit più significativi sia nel caso in cui il relativo riporto sia 1, sia nel caso in cui sia 0.

Full-Adder

A	B	C_{in}	Somma	Ci
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

$$
\begin{aligned}
somma &= \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C_{in}} + A\,\overline{B}\,\overline{C_{in}} + A\,B\,C_{in} \\
&\quad \text{si raccoglie primo-quarto e secondo-terzo} \\
&= C_{in}\,(\overline{A}\,\overline{B} + A\,B) + \overline{C_{in}}\,(\overline{A}\,B + A\,\overline{B}) \\
&= C_{in}\,(\overline{A \oplus B}) + \overline{C_{in}}\,(A \oplus B) = A \oplus B \oplus C_{in}
\end{aligned}
$$

Full-Adder

A	B	C_{in}	Somma	C_{out}
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

$$
\begin{aligned}
riporto &= \overline{A}\,B\,C_{in} + A\,\overline{B}\,C_{in} + A\,B\,\overline{C_{in}} + A\,B\,C_{in} \\
&\quad \text{si raccoglie primo-secondo e terzo-quarto} \\
&= C_{in} \cdot (A \oplus B) + A \cdot B
\end{aligned}
$$

Propagazione del Riporto

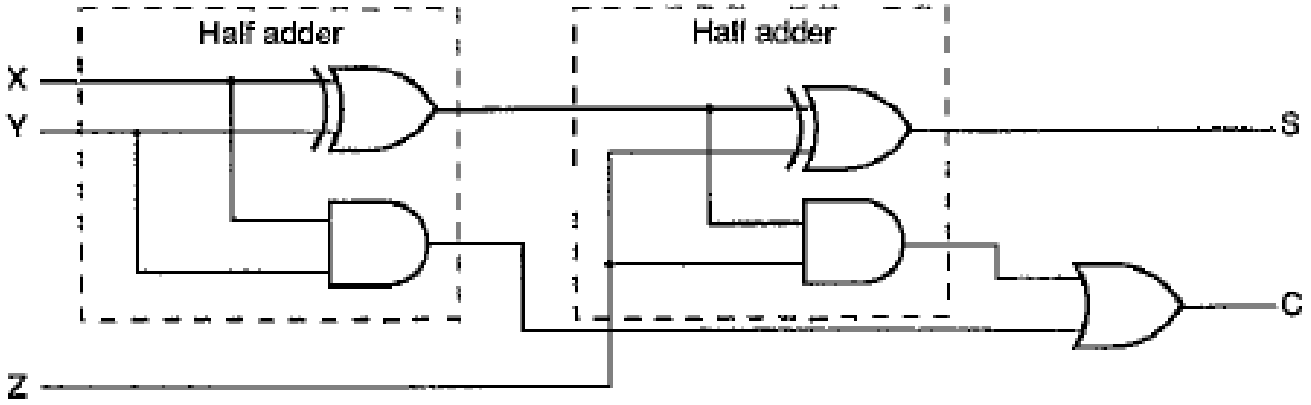


Figura 3 27 Diagramma logico di un circuito full adder

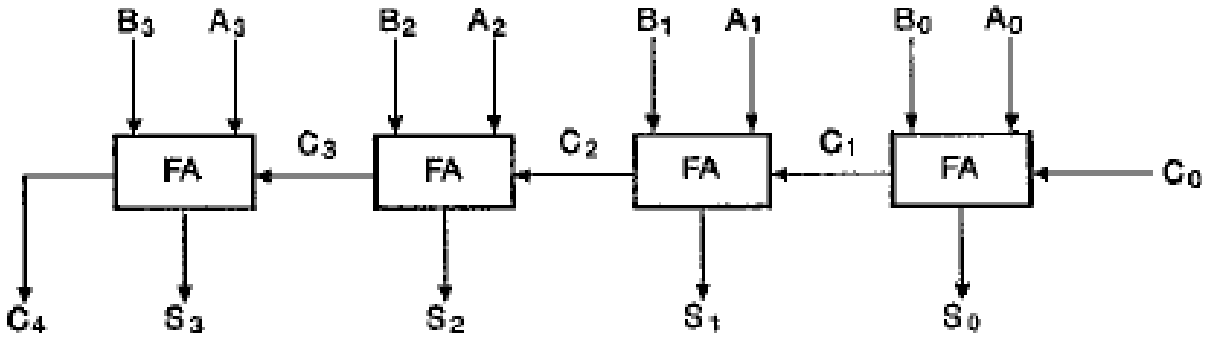
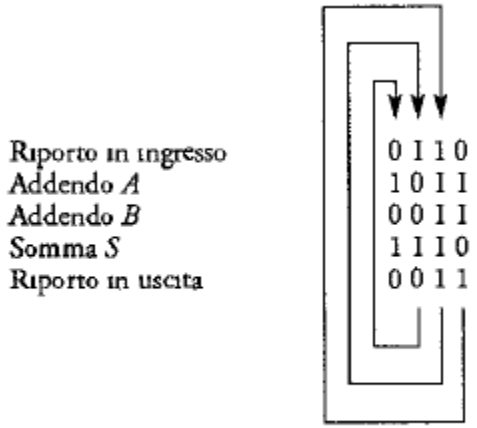


Figura 3 28 Sommatore a 4 bit con riporto in cascata (*ripple carry*)

Calcolo anticipato del Riporto



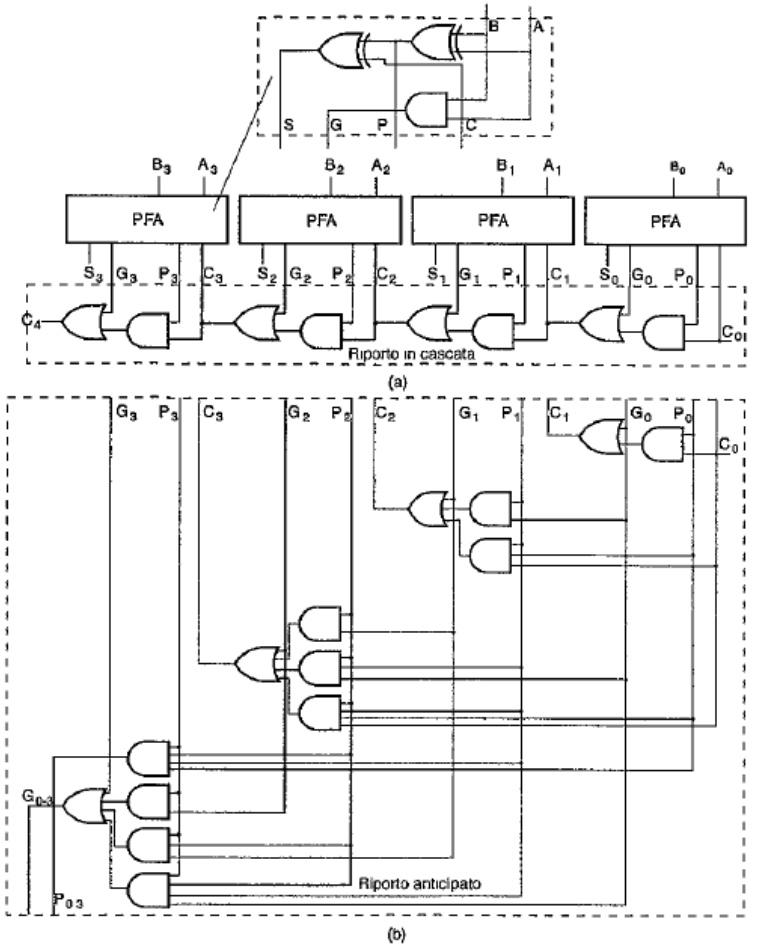
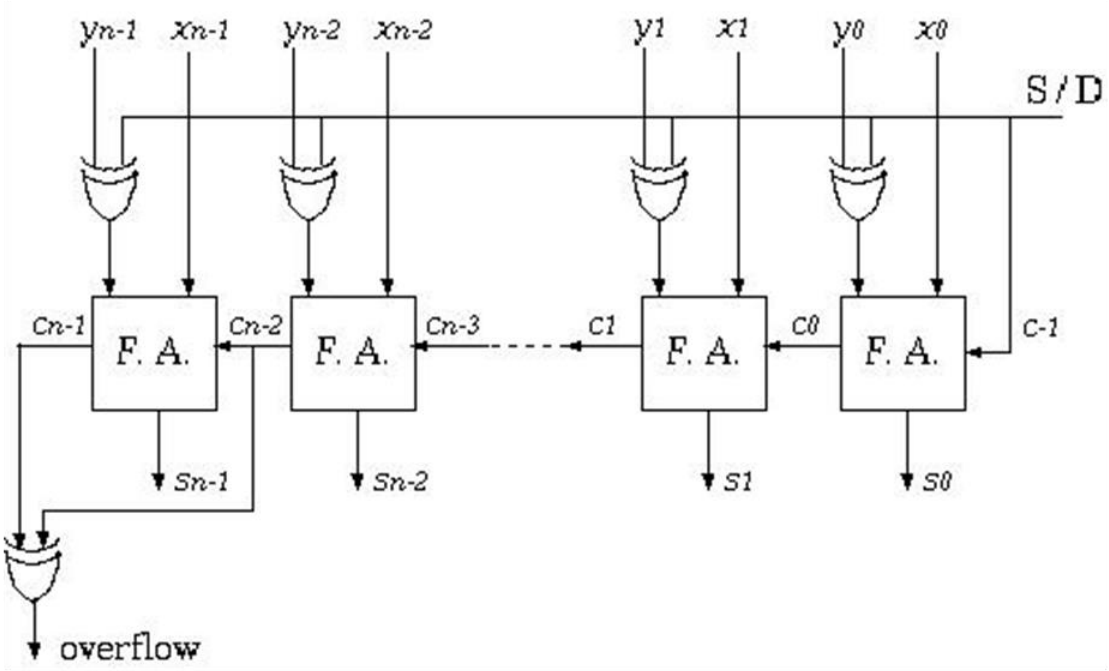


Figura 3 29 Circuito sommatore con logica di riporto anticipato (*carry lookahead*)

Sommatore completo in complemento a 2

Ricordando che il complemento a 2 di un numero è uguale al suo complemento a 1, +1: possiamo allora fare un circuito che calcoli correttamente la differenza di due numeri rappresentati in complemento a 2:



S/D è un segnale che indica se si deve calcolare la somma o la differenza.

ALU a 1 bit

I 6 input indicano il tipo di operazione da calcolare sui due operandi A e B.

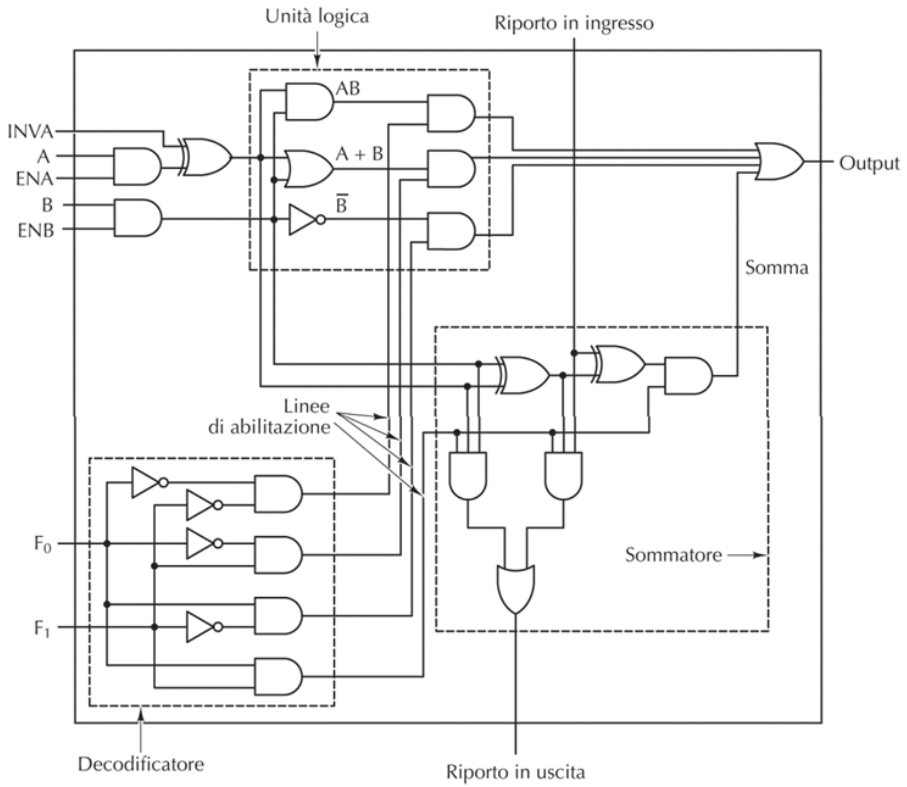
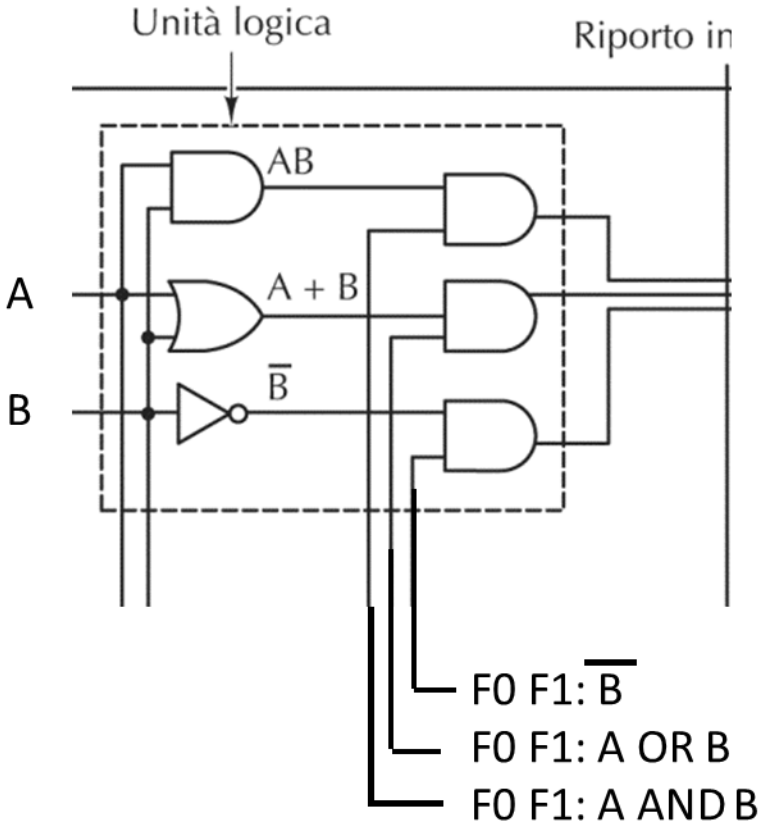


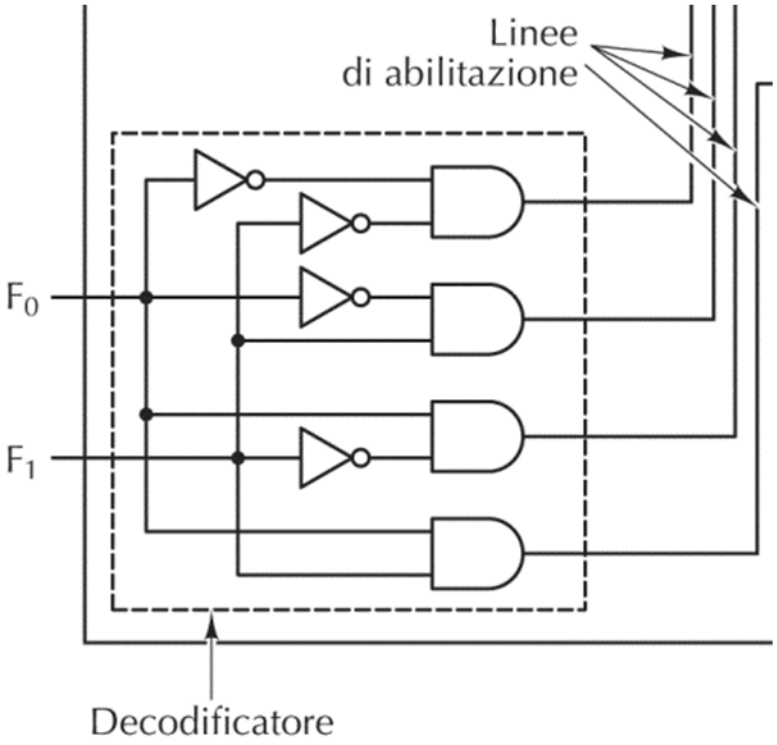
Figura 3.18 ALU a 1 bit.

ALU a 1 bit



Parte dell'ALU che calcola le funzioni logiche

ALU a 1 bit



F0	F1	Funzione
0	0	A AND B
0	1	A OR B
1	0	not(B)
1	1	A+B (con riporto)