Capitolo 5

Circuiti combinatori

5.1 Introduzione

Si chiamano *combinatori* quei circuiti dotati di uno o più ingressi e uno o più uscite, il cui funzionamento è descritto da una funzione logica; in questi circuiti gli ingressi e le uscite possono assumere solo uno di due valori binari previsti (0 e 1); inoltre in ogni istante l'uscita è funzione deterministica unicamente degli ingressi. La figura 5.1 ci illustra la struttura generale di un circuito combinatorio con n ingressi e m uscite, il cui funzionamento è

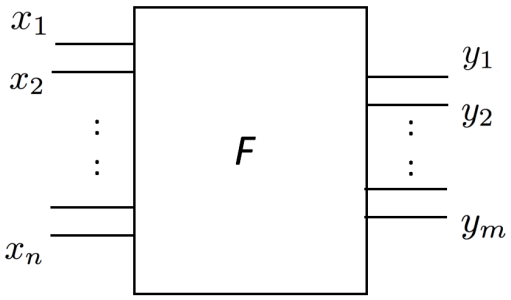


Figura 5.1: Struttura generale di un circuito combinatorio

descritto dalla funzione $F : \mathbf{2^n} \rightarrow \mathbf{2^m}$, che è una funzione Booleana da $\mathbf{2^n}$ a $\mathbf{2^m}$ realizzata mediante m funzioni Booleane $f_i : \mathbf{2^n} \rightarrow \mathbf{2}$

$$
\begin{aligned}
y_1 &= f_1(x_1, x_2, \ldots, x_n) \\
y_2 &= f_2(x_1, x_2, \ldots, x_n) \\
&\vdots \\
y_m &= f_m(x_1, x_2, \ldots, x_n)
\end{aligned}
$$

Nei circuiti elettronici i due stati 0 e 1 sono realizzati mediante due livelli caratteristici di tensione, detti livello alto (*h*) e livello basso (*l*). L'effettiva corrispondenza tra *h* e *l* e le costanti logiche 0 e 1 è convenzionale e va precisata di volta in volta.
E' detta *logica positiva* la convenzione secondo la quale il valore 1 viene associato al livello alto *h*; *logica negativa*

quello in cui il valore 1 è associato al livello basso l.
Si chiama *circuito logico elementare* o *porta logica* un circuito a 1 o 2 ingressi e un'uscita il cui valore è 1 in corrispondenza delle configurazioni degli ingressi descritte dalle funzioni logiche NOT (per 1 ingresso), OR, AND, NAND, NOR, XOR, XNOR (per 2 ingressi). Indipendentemente dalla loro realizzazione circuitale e dal tipo di logica (positiva o negativa), le porte logiche vengono indicate graficamente con i simboli che abbiamo già incontrato e che illustriamo nuovamente in figura 5.2.

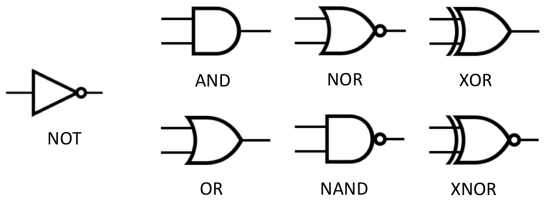


Figura 5.2: Simboli grafici delle principali porte logiche a uno e due ingressi

5.1.1 Itinerari e livelli

Ogni rete logica è formata da un certo numero di porte (AND, OR, ecc.) tra loro variamente interconnesse, da un certo numero di ingressi, contraddistinti in figura 5.3 con i simboli A_i, e da un certo numero di uscite, contraddistinte nella medesima figura con i simboli B_k. Si dice *itinerario* tra due elementi X e Y qualsiasi

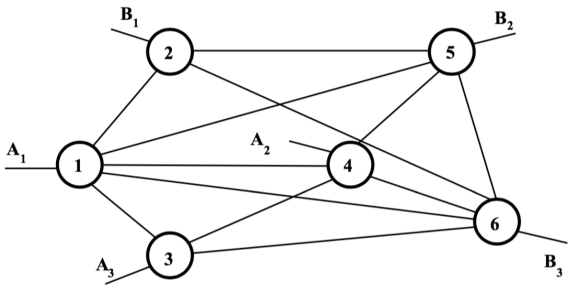
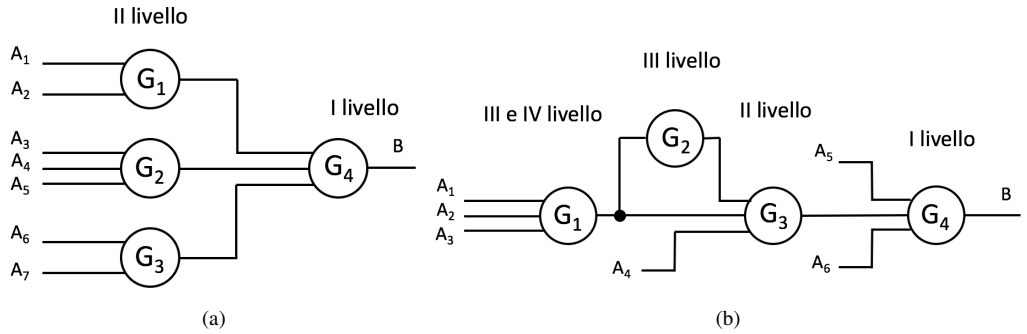


Figura 5.3: Schematizzazione di una rete logica complessa

percorso che colleghi X con Y. Si dice invece *livello di un elemento* X rispetto all'uscita B_j e a un determinato itinerario I, il numero di elementi, X compreso, disposti lungo l'itinerario a partire dall'uscita B_j. Si dice *livello di una variabile* rispetto all'uscita B_j e all'itinerario I il numero di elementi compresi tra il rispettivo ingresso e l'uscita B_j lungo l'itinerario I. In figura 5.4 sono riportati due esempi.



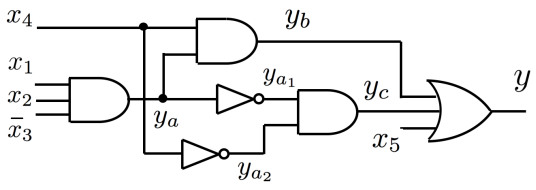
(a) (b)

Figura 5.4: Esempi di livelli in due reti logiche

Si noti che uno stesso elemento può avere più livelli a secondo dell'itinerario scelto. Ad esempio in figura 5.4b il gate G1 e' di III livello secondo l'itinerario G4, G3, G1 e di IV livello secondo l'itinerario G4, G3, G2, G1.
Per i circuiti combinatori, come per ogni altro tipo di circuito, si pongono due problemi opposti: da un lato quello dell'*analisi*, cioè della descrizione del funzionamento del circuito, una volta che sia nota la sua configurazione, il che significa la definizione della funzione che realizza; dall'altro quello della *sintesi*, cioè del progetto di un circuito che realizzi una certa funzione logica, comunque descritta.

5.2 Analisi dei circuiti combinatori

L'analisi di un circuito combinatorio tende a ottenere una rappresentazione della funzione d'uscita y o nella sua forma analitica, oppure sotto forma di tavola di verità. Poiché la rappresentazione circuitale è simbolica, l'analisi non è legata a considerazioni di logica positiva o negativa.
Per effettuare l'analisi, nel caso di circuiti AND-OR-NOT, è sufficiente partire dagli elementi su cui entrano le variabili e procedere verso il terminale di uscita secondo tutti i possibili itinerari, usando la funzione di uscita di ciascun elemento come variabile di ingresso dell'elemento successivo. Nella figura 5.5 vediamo un esempio di una rete e delle equazioni a essa associate, ricavate col procedimento sopra descritto. L'analisi dei circuiti basati



(a) Rete AND-OR-NOT da analizzare

$$
\begin{aligned}
y_a &= x_1\, x_2\, \overline{x_3} \\
y_{a_1} &= \overline{y_a} = \overline{x_1\, x_2\, \overline{x_3}} = \overline{x_1} + \overline{x_2} + x_3 \\
y_{a_2} &= \overline{x_4} \\
y_b &= x_4\, y_a = x_1\, x_2\, \overline{x_3}\, x_4 \\
y_c &= y_{a_1}\, y_{a_2} = \overline{x_4}\, (\overline{x_1} + \overline{x_2} + x_3) \\
y &= y_b + y_c + x_5 = x_1\, x_2\, \overline{x_3}\, x_4 \\
&\quad + \overline{x_4}\, (\overline{x_1} + \overline{x_2} + \overline{x_3}) + x_5
\end{aligned}
$$

(b) Equazioni dell'analisi

Figura 5.5: Rete AND-OR-NOT ed equazioni che si ottengono dall'analisi

su porte NAND e NOR è notevolemente più complessa, a causa della non associatività degli operatori. Lasciamo questo tipo di analisi al corso di *Reti Logiche*.

5.3 Sintesi dei circuiti combinatori

Eseguire la sintesi di un circuito combinatorio consiste, come già accennato, nel progettare un circuito a n ingressi che soddisfi una determinata funzione di uscita y di progetto. La funzione y che il circuito deve soddisfare

può essere assegnata in diverse forme. Più precisamente:

1. con la descrizione a parole del funzionamento del circuito. E' questa la forma di assegnazione più comune, ma anche la più imprecisa. E' necessario porre un'estrema attenzione alla corretta interpretazione di eventuali condizioni implicite e all'esistenza di vincoli di qualsiasi natura. Dalla descrizione verbale è necessario passare poi alla tavola di verità, assegnando il valore della funzione per ognuna delle 2^n configurazioni degli ingressi;
2. con una vera e propria tavola di verità, che è in definitiva l'effettivo punto di partenza della sintesi cui tutti gli altri tipi di assegnazione devono essere ricondotti;
3. con un'espressione analitica, che è il modo più conciso, anche se non univoco, di descrivere il funzionamento di un circuito;
4. con uno schema logico; è questa una procedura generalmente usata quando un determinato circuito logico debba esser riprogettato con componenti diversi. In tal caso, con le regole dell'analisi si ricava un'espressione analitica della funzione y.

Qualunque sia il metodo di assegnazione, la sintesi procede partendo dalla tavola di verità o da un'espressione analitica; applicando i metodi di semplificazione delle funzioni logiche si giunge alla forma più conveniente per gli scopi che ci si propone. Si noti che non sempre la forma più conveniente corrisponde alla forma minima della funzione. Ad esempio non è sempre opportuno realizzare circuitalmente la forma minima algebrica, in quanto vi possono essere dei vincoli sul numero massimo di livelli. Infatti, se è ben vero quanto esposto precedentemente, e cioè che nei circuiti combinatori l'uscita è, istante per istante, funzione unicamente degli ingressi, non significa che la variazione degli ingressi sia avvertita immediatamente in uscita; tale affermazione sta piuttosto a significare che ogni configurazione di ingresso dà luogo a una determinata uscita e che eventuali transitori di commutazione possono ritardare, ma non modificare quest'uscita.
Poiché il tempo di commutazione Δ di qualsiasi porta logica, per quanto piccolo, non è mai nullo, il tempo di risposta di un circuito a n livelli, al variare della configurazione d'ingresso, è $n \cdot \Delta$. In definitiva, il ritardo totale tra ingresso e uscita è proporzionale al numero di livelli e potendo la forma minima di una funzione contenere un numero di livelli molto elevato, la sua diretta realizzazione circuitale potrebbe dar luogo a ritardi intollerabili.
La forma in cui si ha il minimo ritardo è quella a due livelli, che d'altra parte è quella che si ottiene con i metodi di semplificazione che sono stati esposti nel capitolo 4. La convenienza di eventuali fattorizzazioni va valutata caso per caso.
Si può dunque concludere che la sintesi di un circuito combinatorio procede attraverso i seguenti passi:

1. Descrizione del funzionamento del circuito
2. Determinazione della tavola di verità
3. Sintesi della funzione Booleana
4. Semplificazione della funzione logica relativa
5. Determinazione della forma minima più conveniente
6. Disegno del circuito

Si osservi che il passo (5) non può essere attuato secondo un procedimento sistematico, e l'effettiva forma minima più conveniente andrà valutata di volta in volta, eventualmente facendo uso di tecniche basate sul concetto di decomponibilità che saranno illustrate nel successivo corso di *Reti Logiche*.

Esempio 5.1. Si voglia realizzare un circuito a tre ingressi e quattro uscite; sugli ingressi si può presentare un numero binario compreso tra 0 e 5. All'uscita di tale circuito si deve ottenere il prodotto per 3 del numero in ingresso. Il massimo numero di uscita rappresentabile con 4 bit è 15 e sarà necessario sintetizzare quattro funzioni logiche.

	$\boldsymbol{x}$			$\boldsymbol{y}$			
$3 \cdot \boldsymbol{x} = \boldsymbol{y}$	x_2	x_1	x_0	y_3	y_2	y_1	y_0
$3 \cdot 0 = 0$	0	0	0	0	0	0	0
$3 \cdot 1 = 3$	0	0	1	0	0	1	1
$3 \cdot 2 = 6$	0	1	0	0	1	1	0
$3 \cdot 3 = 9$	0	1	1	1	0	0	1
$3 \cdot 4 = 12$	1	0	0	1	1	0	0
$3 \cdot 5 = 15$	1	0	1	1	1	1	1
$3 \cdot 6 = -$	1	1	0	–	–	–	–
$3 \cdot 7 = -$	1	1	1	–	–	–	–

Figura 5.6: Tavola di verità del circuito che moltiplica per 3.

Le tavole di verità di ogni funzione sono riportate in figura 5.6, mentre in figura 5.7 si hanno le corrispondenti mappe di Karnaugh.

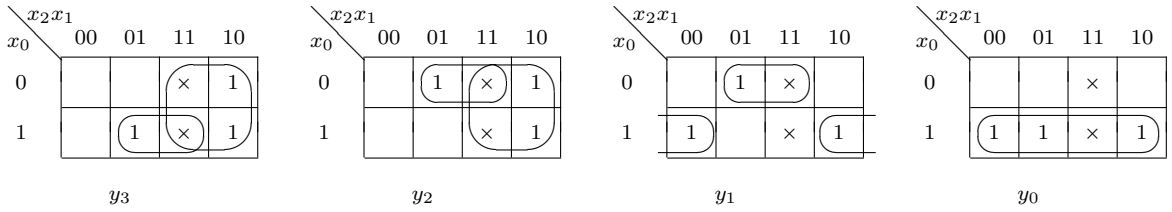


Figura 5.7: Mappe di Karnaugh per le quattro funzioni di figura 5.6

Utilizzando opportunamente le condizioni non specificate si ottiene:

$$y_3 = x_2 + x_0\, x_1 \qquad y_1 = \overline{x_0}\, x_1 + x_0\, \overline{x_1}$$
$$y_2 = x_2 + \overline{x_0}\, x_1 \qquad y_0 = x_0$$

cui corrisponde il circuito di figura 5.8, ottenuto mettendo in comune il termine $\overline{x_0}x_1$ tra le funzioni y_1 e y_2. ◯

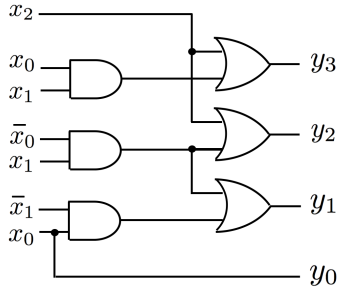
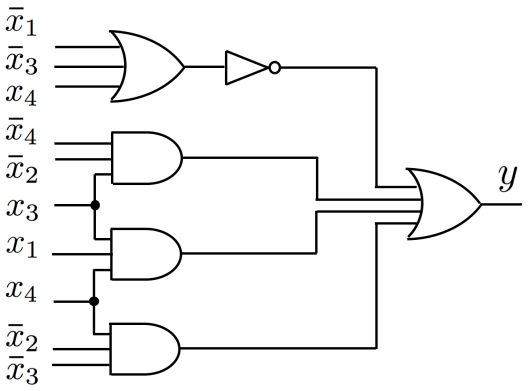
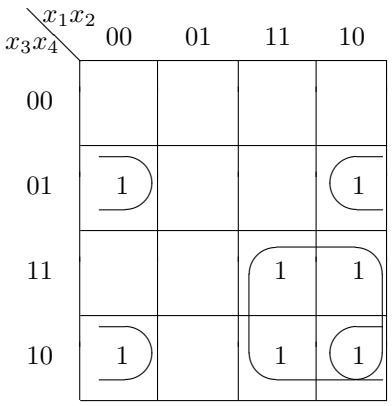


Figura 5.8: Moltiplicatore per 3 di un numero compreso tra 0 e 5



(a) Circuito assegnato da semplificare



(b) Mappa di Karnaugh della funzione 5.1

Figura 5.9: Semplificazione di un circuito mediante mappa di Karnaugh

Esempio 5.2. Si voglia sintetizzare un circuito con la stessa funzione Booleana di quello illustrato in figura 5.9a, ma possibilmente più economico.

L'espressione analitica della funzione di trasmissione è

$$\begin{aligned} y &= \overline{\overline{x_1} + \overline{x_3} + x_4} + \overline{x_2}\, x_3\, \overline{x_4} + x_1\, x_3\, x_4 + \overline{x_2}\, \overline{x_3}\, x_4 = \\ &= x_1\, x_3\, \overline{x_4} + \overline{x_2}\, x_3\, \overline{x_4} + x_1\, x_3\, x_4 + \overline{x_2}\, \overline{x_3}\, x_4 \end{aligned} \tag{5.1}$$

Dalla mappa di Karnaugh di figura 5.9b si ricava l'espressione minima a due livelli

$$y = x_1\, x_3 + \overline{x_2}\, \overline{x_3}\, x_4 + \overline{x_2}\, x_3\, \overline{x_4} \tag{5.2}$$

e fattorizzando si ottiene:

$$y = x_1\, x_3 + \overline{x_2}\, (\overline{x_3}\, x_4 + x_3\, \overline{x_4}) \tag{5.3}$$

Per la realizzazione della (5.2) sono sufficienti 4 porte, mentre per la (5.3) ne sono necessarie 6. Il circuito che

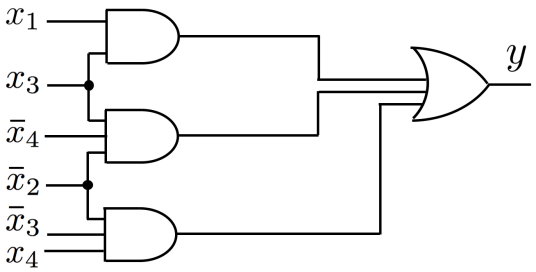


Figura 5.10: Circuito più economico che realizza la stessa funzione Booleana del circuito di figura 5.9a

conviene realizzare è allora quello relativo all'espressione (5.2), ed è riportato in figura 5.10. ◯

Si noti che negli esempi fatti si è supposto di avere a disposizione all'ingresso del circuito sia le variabili dirette che la loro negazione. Quando questa situazione non si verifica anche il numero di invertitori va minimizzato. Ad esempio la funzione

$$y = \overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4}$$

realizzabile con quattro invertitori e un gate OR, può essere realizzata molto più convenientemente tenendo presente che

$$y = \overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4} = \overline{x_1\, x_2\, x_3\, x_4}$$

richiedendo in tal caso solamente un AND e un NOT.
Facciamo un altro esempio di gran lunga più impegnativo dei precedenti, tratto da [3].

Esempio 5.3. Si voglia costruire un circuito che esegue la moltiplicazione tra due numeri espressi in notazione posizionale in base 2. Per mantenere una complessità gestibile in un contesto didattico, limitiamoci a due numeri da due bit, e dunque un moltiplicatore binario a 2×2 bit. Siano $\boldsymbol{x} = x_1x_0$ i bit del moltiplicando e $\boldsymbol{y} = y_1y_0$ i bit del moltiplicatore; ciascun fattore varia dunque nell'intervallo $[0, 3]$, mentre il prodotto è compreso nell'intervallo $[0, 9]$. Il progetto richiede la costruzione della tavola di verità per tutte le possibili 16 combinazioni, tenendo conto che per rappresentare il valore del prodotto serviranno $\lceil \log_2 9 \rceil = 4$ bit. Se, per esempio, si ha $\boldsymbol{x} = 10\ (2)$ e $\boldsymbol{y} = 11\ (3)$, a questa configurazione d'ingresso deve corrispondere $\boldsymbol{z} = 0110\ (6)$, considerato che $2 \cdot 3 = 6$. Operando in questo modo si ottiene la tavola di verità di figura 5.11, nella quale ogni colonna i-esima rappresenta i valori assunti dalla funzione z_i in corrispondenza delle varie quaterne d'ingresso. Dobbiamo allora valutare quattro

	$\boldsymbol{x}$		$\boldsymbol{y}$		$\boldsymbol{z}$			
$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{z}$	x_1	x_0	y_1	y_0	z_3	z_2	z_1	z_0
$0 \cdot 0 = 0$	0	0	0	0	0	0	0	0
$0 \cdot 1 = 0$	0	0	0	1	0	0	0	0
$0 \cdot 2 = 0$	0	0	1	0	0	0	0	0
$0 \cdot 3 = 0$	0	0	1	1	0	0	0	0
$1 \cdot 0 = 0$	0	1	0	0	0	0	0	0
$1 \cdot 1 = 1$	0	1	0	1	0	0	0	1
$1 \cdot 2 = 2$	0	1	1	0	0	0	1	0
$1 \cdot 3 = 3$	0	1	1	1	0	0	1	1
$2 \cdot 0 = 0$	1	0	0	0	0	0	0	0
$2 \cdot 1 = 2$	1	0	0	1	0	0	1	0
$2 \cdot 2 = 4$	1	0	1	0	0	1	0	0
$2 \cdot 3 = 6$	1	0	1	1	0	1	1	0
$3 \cdot 0 = 0$	1	1	0	0	0	0	0	0
$3 \cdot 1 = 3$	1	1	0	1	0	0	1	1
$3 \cdot 2 = 6$	1	1	1	0	0	1	1	0
$3 \cdot 3 = 9$	1	1	1	1	1	0	0	1

Figura 5.11: Tavola di verità per il moltiplicatore binario a 2×2 bit

funzioni Booleane del tipo $\mathbf{2^2 \rightarrow 2}$. Poichè ci sono pochi 1 e molti 0, conviene usare la I forma canonica, basata

sui termini minimi. Facendo i calcoli si ottiene

$$
\begin{aligned}
z_0 &= \overline{x_1}\, x_0\, \overline{y_1}\, y_0 + \overline{x_1}\, x_0\, y_1\, y_0 + x_1\, x_0\, \overline{y_1}\, y_0 + x_1\, x_0\, y_1\, y_0 \\
&= \overline{x_1}\, x_0\, y_0(\overline{y_1} + y_1) + x_1\, x_0\, y_0(\overline{y_1} + y_1) \\
&= \overline{x_1}\, x_0\, y_0 + x_1\, x_0\, y_0 \\
&= x_0\, y_0(\overline{x_1} + x_1) \\
&= x_0\, y_0
\end{aligned}
$$

$$
\begin{aligned}
z_1 &= \overline{x_1}\, x_0\, y_1\, \overline{y_0} + \overline{x_1}\, x_0\, y_1\, y_0 + x_1\, \overline{x_0}\, \overline{y_1}\, y_0 + x_1\, \overline{x_0}\, y_1\, y_0 + x_1\, x_0\, \overline{y_1}\, y_0 + x_1\, x_0\, y_1\, \overline{y_0} \\
&= \overline{x_1}\, x_0\, y_1(\overline{y_0} + y_0) + x_1\, \overline{x_0}\, y_0(\overline{y_1} + y_1) + x_1\, x_0\, \overline{y_1}\, y_0 + x_1\, x_0\, y_1\, \overline{y_0} \\
&= \overline{x_1}\, x_0\, y_1 + x_1\, \overline{x_0}\, y_0 + x_1\, x_0\, \overline{y_1}\, y_0 + x_1\, x_0\, y_1\, \overline{y_0} \\
&= x_0\, y_1(\overline{x_1} + x_1\, \overline{y_0}) + x_1\, y_0(\overline{x_0} + x_0\, \overline{y_1}) \\
&= x_0\, y_1(\overline{x_1} + \overline{y_0}) + x_1\, y_0(\overline{x_0} + \overline{y_1}) \\
&= \overline{x_1}\, x_0\, y_1 + x_0\, y_1\, \overline{y_0} + x_1\, \overline{x_0}\, y_0 + x_1\, \overline{y_1}\, y_0
\end{aligned}
$$

$$
\begin{aligned}
z_2 &= x_1\, \overline{x_0}\, y_1\, \overline{y_0} + x_1\, \overline{x_0}\, y_1\, y_0 + x_1\, x_0\, y_1\, \overline{y_0} \\
&= x_1\, \overline{x_0}\, y_1(\overline{y_0} + y_0) + x_1\, x_0\, y_1\, \overline{y_0} \\
&= x_1\, y_1(\overline{x_0} + x_0\overline{y_0}) \\
&= x_1\, y_1(\overline{x_0} + \overline{y_0}) \\
&= x_1\overline{x_0}\, y_1 + x_1\, y_1\, \overline{y_0}
\end{aligned}
$$

$$
z_3 = x_1\, x_0\, y_1\, y_0
$$

Risulta però molto più agevole ricavare gli implicanti direttamente dalle mappe di Karnaugh delle quattro funzioni z_3, z_2, z_1, z_0, come rappresentato in figura 5.12.

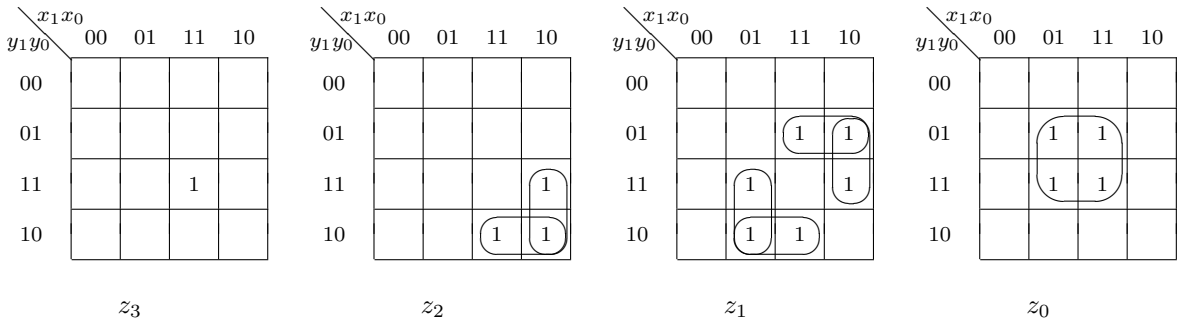


Figura 5.12: Mappe di Karnaugh per le quattro funzioni di figura 5.11

Il circuito completo del moltiplicatore binario è rappresentato in figura 5.13; esso è ricavato direttamente dalle espressioni di z_i usando porte AND, OR e NOT.

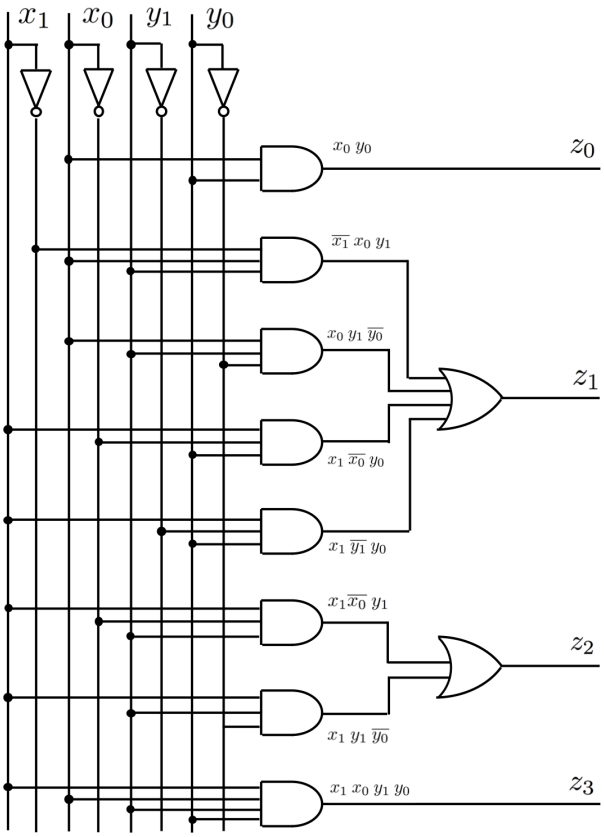


Figura 5.13: Moltiplicatore binario a 2×2 bit

5.4 Moduli combinatori

L'algebra Booleana permette di analizzare e progettare qualunque tipo di rete combinatoria, di qualunque complessità. In precedenza abbiamo mostrato anche le tecniche per semplificare le reti ottenute direttamente dalle espressioni delle forme canoniche, basate sulla somma di prodotti o sui prodotti di somme. Sebbene in linea di principio quei metodi possono essere utilizzati per reti combinatorie di qualunque estensione, nella pratica si preferisce affrontare il progetto di una rete complessa secondo una tecnica di scomposizione della rete in blocchi funzionali, detti *moduli combinatori*. In questo modo si rinuncia alla soluzione teoricamente ottima, a favore di una maggiore comprensibilità, gestibilità e soprattutto modularità del progetto. Ciò porta anche a un effettivo risparmio economico, poiché i moduli combinatori, se prodotti in larga scala, hanno prezzi via via sempre più bassi; dunque non sempre un numero minore di porte implica un risparmio economico.
Già a partire dagli anni '70 vennero prodotti moduli integrati, che svolgendo precise funzionalità di carattere combinatorio trovarono impiego nello sviluppo di progetti complessi. Nella parte che segue vengono esaminati alcuni moduli combinatori di uso corrente.

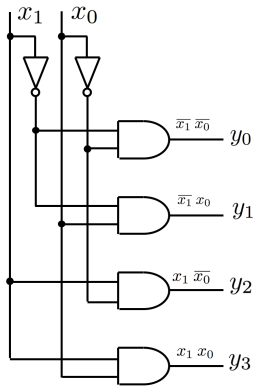
5.4.1 Decodificatori

I decodificatori convertono un numero espresso in notazione posizionale in base 2 in un numero intero in base 10. Un decodificatore accetta in ingresso n bit e presenta in uscita $m = 2^n$ linee, numerate da 0 a $2^n - 1$, in modo tale che va a 1 la sola linea y_j che corrisponde all'intero j codificato in notazione posizionale dagli n bit d'ingresso. Se dunque indichiamo con y_0, y_1, ..., y_m le uscite del decodificatore, la generica uscita y_j è ottenuta come AND delle n variabili che compongono il *minterm* j. Nelle figure 5.14a e 5.14b vengono riportati, rispettivamente, la tavola di verità e il circuito di un decodificatore a 2 bit; analogamente, nelle figure 5.15a e 5.15b abbiamo la tavola di verità e il circuito di un decodificatore a 3 bit. In figura 5.14c viene invece rappresentato il simbolo circuitale per un generico decodificatore a n bit.

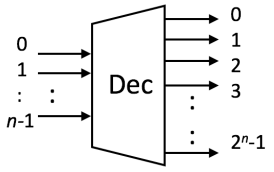
$\boldsymbol{x}$		$\boldsymbol{y}$			
x_1	x_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$y_0 = \overline{x_1}\ \overline{x_0}$$
$$y_1 = \overline{x_1}\ x_0$$
$$y_2 = x_1\ \overline{x_0}$$
$$y_3 = x_1\ x_0$$

(a) Tavola di verità di un decodificatore a 2 bit



(b) Circuito del decodificatore a 2 bit



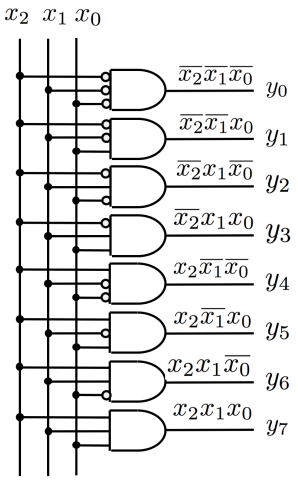
(c) Simbolo circuitale di un decodificatore a n bit

Figura 5.14: Decodificatore a 2 bit e simbolo circuitale di un codificatore a n bit

$\boldsymbol{x}$			$\boldsymbol{y}$							
x_2	x_1	x_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$$y_0 = \overline{x_2}\ \overline{x_1}\ \overline{x_0} \qquad y_4 = x_2\ \overline{x_1}\ \overline{x_0}$$
$$y_1 = \overline{x_2}\ \overline{x_1}\ x_0 \qquad y_5 = x_2\ \overline{x_1}\ x_0$$
$$y_2 = \overline{x_2}\ x_1\ \overline{x_0} \qquad y_6 = x_2\ x_1\ \overline{x_0}$$
$$y_3 = \overline{x_2}\ x_1\ x_0 \qquad y_7 = x_2\ x_1\ x_0$$

(a) Tavola di verità di un decodificatore a 3 bit



(b) Circuito del decodificatore a 3 bit

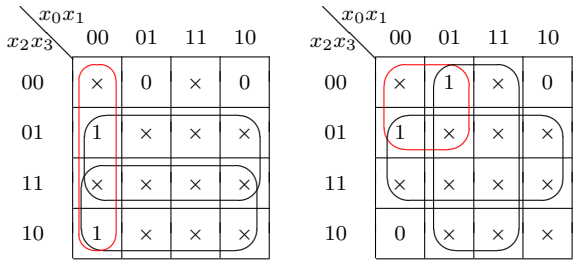
Figura 5.15: Tavola di verità e circuito di un decodificatore a 3 bit

5.4.2 Codificatori

I codificatori convertono un numero intero in base 10 in un numero espresso in notazione posizionale in base 2. Un codificatore svolge dunque la funzione inversa di un decodificatore, nel senso che esso prevede $m = 2^n$ ingressi e n uscite. Le uniche configurazioni ammesse per gli ingressi sono quelle in cui c'è esattamente un solo 1 in corrispondenza della linea x_j; dunque $x_j = 1$ e $x_i = 0$ per ogni $i \neq j$. Indicando con $x_0, x_1, ..., x_{m-1}$ gli ingressi, la corrispondente configurazione di uscita sulle variabili $y_{n-1}, y_{n-2}, ..., y_0$ è tale che esse codificano j in notazione posizionale in base 2. Nelle figure 5.16a e 5.16b vengono riportate, rispettivamente, la tavola di verità e le due mappe di Karnaugh associate alle variabili y_1 e y_0. Si noti che nella tabella di verità sono state riportate le sole configurazioni di ingresso definite; le altre danno luogo a condizioni non specificate, che consentono un amplissimo margine di libertà nella realizzazione effettiva del circuito. Ciò è evidente dalle mappe di Karnaugh di figura 5.16b, che consentono almeno 2 soluzioni; la prima implica l'uso di due porte OR e la seconda due porte AND e due NOT. Nella figura 5.17a e 5.17b vengono riprodotti, rispettivamente, il circuito del codificatore a 2 bit

$\boldsymbol{x}$				$\boldsymbol{y}$	
x_0	x_1	x_2	x_3	y_1	y_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

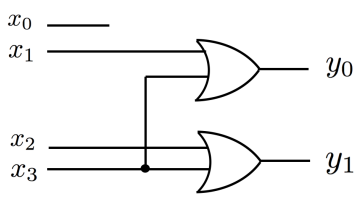
(a) Tavola di verità di un codificatore a 2 bit



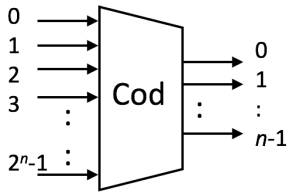
(b) Mappe di Karnaugh per le funzioni di figura 5.16a

Figura 5.16: Tavola di verità e mappe di Karnaugh del codificatore a 2 bit

associato alla soluzione con le porte OR e il simbolo circuitale di un generico codificatore a n bit. Analogamente, nelle figure 5.18a e 5.18b abbiamo la tavola di verità e il circuito di un codificatore a 3 bit.



(a) Circuito del codificatore a 2 bit



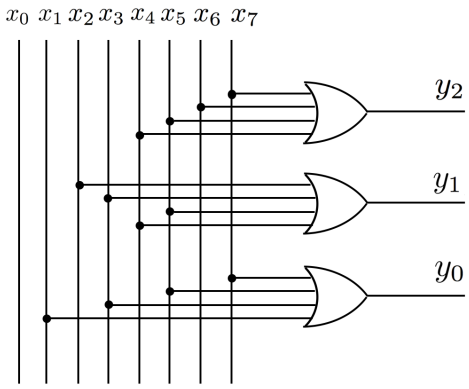
(b) Simbolo circuitale di un codificatore a n bit

Figura 5.17: Codificatore a 2 bit e simbolo circuitale di un decodificatore a n bit

$\boldsymbol{x}$								$\boldsymbol{y}$		
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$y_2 = x_4 + x_5 + x_6 + x_7$$
$$y_1 = x_2 + x_3 + x_6 + x_7$$
$$y_0 = x_1 + x_3 + x_5 + x_7$$

(a) Tavola di verità di un codificatore a 3 bit



(b) Circuito del codificatore a 3 bit

Figura 5.18: Codificatore a 3 bit

5.4.3 Selettori

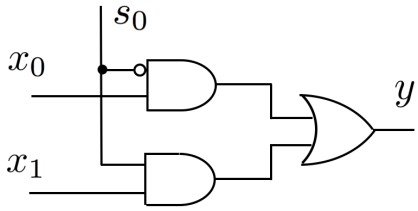
Un *selettore d'ingresso* (o *Multiplexer* o *Mux*) è un modulo che permette di selezionare uno tra 2^n ingressi e presentarlo sull'unica uscita. La selezione si effettua attraverso n linee di comando.

$\boldsymbol{s}$	$\boldsymbol{x}$		$\boldsymbol{y}$
s_0	x_0	x_1	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a) Tavola di verità di un *Multiplexer* a 2 vie

$$\begin{aligned} y &= \overline{s_0} x_0 \overline{x_1} + \overline{s_0} x_0 x_1 \\ &+ s_0 \overline{x_0} x_1 + s_0 x_0 x_1 \\ &= \overline{s_0} x_0 + s_0 x_1 \end{aligned}$$

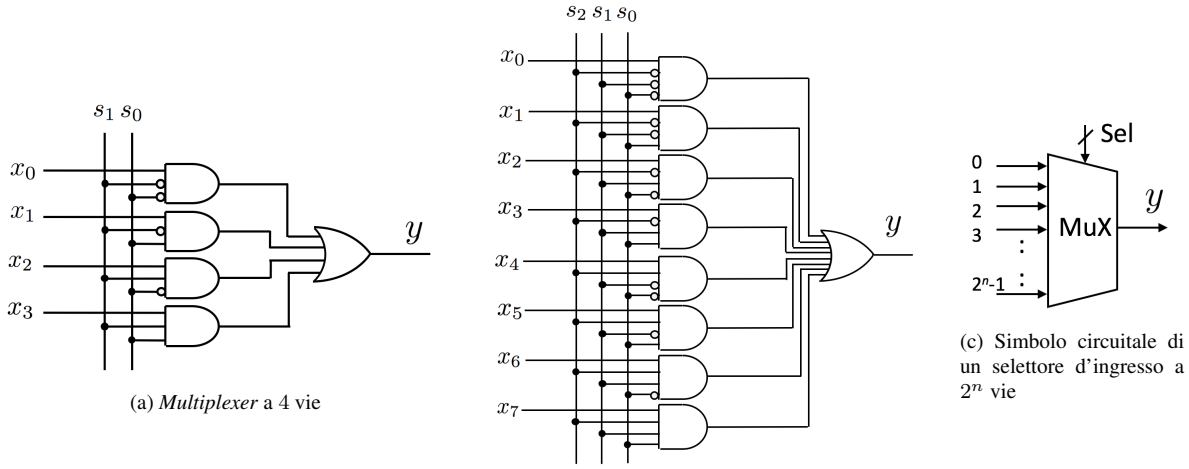
(b) *minterm* per la tavola di verità 5.19a



(c) *Multiplexer* a 2 vie basato sui *minterm* di figura 5.19b

Figura 5.19: Selettore d'ingresso (o *Multiplexer*) a 2

Nella figura 5.19a viene fornita la tavola di verità del *Multiplexer* a 2 vie (2:1 *Mux*), che si ricava tenendo conto che per $s_0 = 0$ viene riportato in uscita il contenuto di x_0, mentre per $s_0 = 1$ si porta in uscita il contenuto di x_1; la semplificazione tramite *minterm* di figura 5.19b porta alla soluzione circuitale di figura 5.19c. Nelle figure 5.20a, 5.20b sono invece rappresentati i *Multiplexer* a 4 e 8 vie, mentre la figura 5.20c illustra il simbolo circuitale di un generico *Multiplexer* a n vie.
Un *selettore d'uscita* (o *Demultiplexer* o *Demux*) è al contrario un dispositivo che permette di dirottare l'unico ingresso su una delle possibili 2^n uscite.

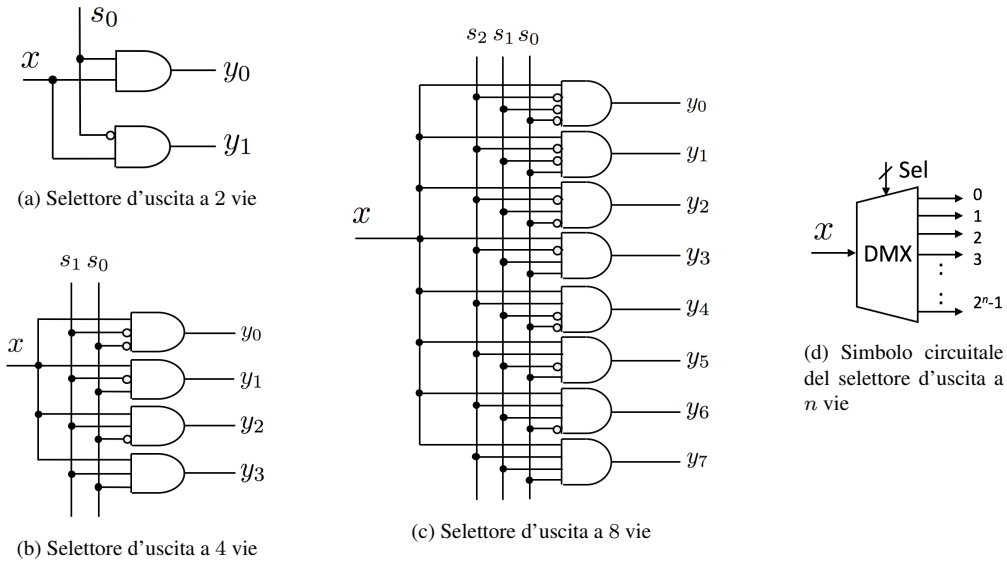


(a) *Multiplexer* a 4 vie

(b) Selettore d'ingresso a 8 vie

(c) Simbolo circuitale di un selettore d'ingresso a 2^n vie

Figura 5.20: Selettori d'ingresso a 4 e 8 vie e simbolo circuitale di un generico selettore a n vie



(a) Selettore d'uscita a 2 vie

(b) Selettore d'uscita a 4 vie

(c) Selettore d'uscita a 8 vie

(d) Simbolo circuitale del selettore d'uscita a n vie

Figura 5.21: Selettori d'uscita (o *demultiplexer*) a 2, 4 e 8 vie e relativo simbolo circuitale

La figura 5.21 mostra un *Demultiplexer* a 2 vie (1:2 *Demux*), uno a 4 vie (1:4 *Demux*) e uno a 8 vie (1:8 *Demux*), oltre al simbolo grafico di un generico selettore d'uscita a 2^n vie. Si noti che nel caso di selettore di uscita l'ingresso viene presentato sulla via selezionata, mentre tutti gli altri ingressi sono a 0.
In entrambi i selettori (d'ingresso e d'uscita) l'n-pla binaria della linea Sel seleziona quale, tra le 2^n linee, è interessata alla connessione con l'uscita (l'ingresso).

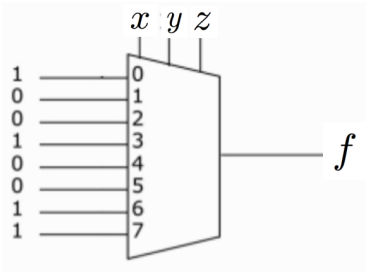
5.4.4 Costruzione modulare di una funzione Booleana

Illustriamo ora un impiego particolarmente interessante dei selettori d'ingresso, o *multiplexer*: la costruzione di qualunque funzione Booleana di n variabili attraverso un selettore d'ingresso a 2^n vie.
Una qualunque funzione di n variabili può essere ricondotta a una somma di *minterm*, come si è visto nell'equa-

zione (4.16). Usiamo come esempio la funzione Booleana di figura 5.22a.

		x	y	z	f	
m_0	$\bar{x}\bar{y}\bar{z}$	0	0	0	1	μ_0
m_1	$\bar{x}\bar{y}z$	0	0	1	0	μ_1
m_2	$\bar{x}y\bar{z}$	0	1	0	0	μ_2
m_3	$\bar{x}yz$	0	1	1	1	μ_3
m_4	$x\bar{y}\bar{z}$	1	0	0	0	μ_4
m_5	$x\bar{y}z$	1	0	1	0	μ_5
m_6	$xy\bar{z}$	1	1	0	1	μ_6
m_7	xyz	1	1	1	1	μ_7

(a) Funzione Booleana da realizzare col *Multiplexer*



(b) *Multiplexer* che realizza la funzione Booleana di figura 5.22a

Figura 5.22: Funzione Booleana e selettore idoneo a realizzarla

La funzione può essere scritta come

$$f(x,y,z) = \sum_{i\in\{0,3,6,7\}} m_i = m_0 + m_3 + m_6 + m_7 = \bar{x}\bar{y}\bar{z} + \bar{x}yz + xy\bar{z} + xyz$$
$$= 1\cdot\bar{x}\bar{y}\bar{z} + 0\cdot\bar{x}\bar{y}z + 0\cdot\bar{x}y\bar{z} + 1\cdot\bar{x}yz + 0\cdot x\bar{y}\bar{z} + 0\cdot x\bar{y}z + 1\cdot xy\bar{z} + 1\cdot xyz \qquad (5.4)$$

poiché $0, 3, 6$ e 7 sono le codifiche in base due di 000, 011, 110 e 111; nella 5.4 si mette 1 in corrispondenza dei *minterm* che compaiono nella funzione e 0 in corrispondenza di quelli che non compaiono. Con un selettore a otto vie la funzione f si ottiene impiegando x, y, z come linee di selezione e ponendo le otto linee di ingresso a 0 o a 1 a seconda che il coefficiente del corrispondente *minterm*, che corrisponde al valore della funzione per la terna binaria associata, sia a 0 o a 1. In figura 5.23 si vede il circuito per l'attuazione della funzione in oggetto. Poiché ogni

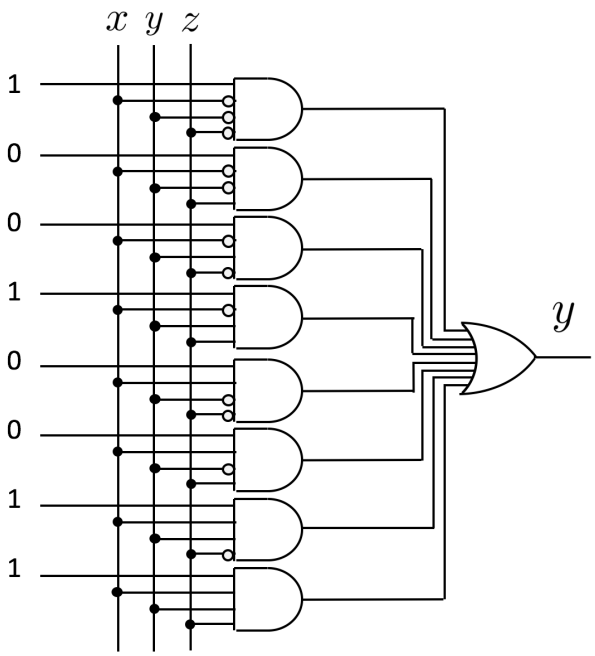
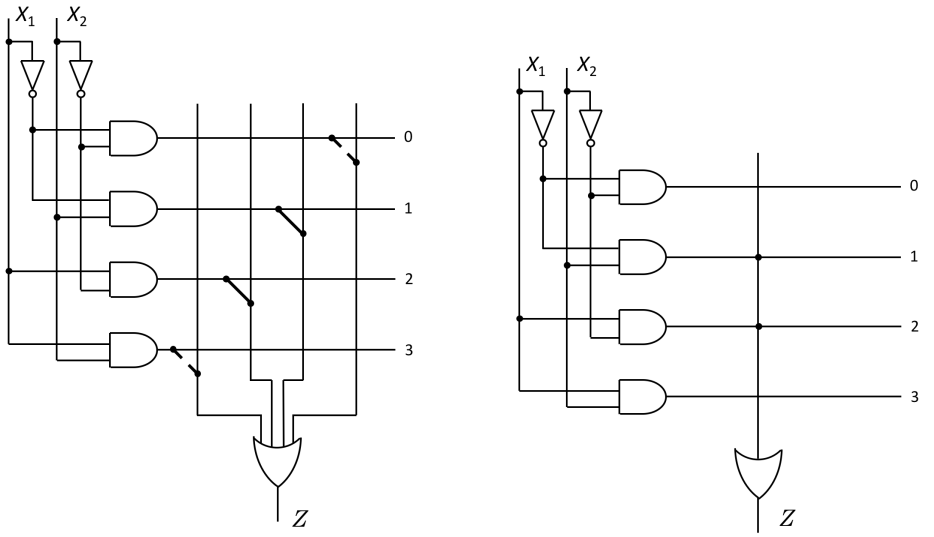


Figura 5.23: Modulo selettore per la realizzazione della funzione 5.4

funzione logica può essere ricondotta alla sua prima forma canonica, ne deriva che è possibile realizzare qualunque

funzione con il metodo sopra esposto.
E' ragionevole domandarsi perché usare tale metodo. La risposta sta nel fatto che con i circuiti integrati la soluzione con *multiplexer*, pur comportando solitamente un numero maggiore di porte, può portare a una riduzione del costo complessivo grazie alla produzione su larga scala di tali moduli. Inoltre, l'impiego del *multiplexer* dà maggiore flessibilità al progetto; infatti gli ingressi corrispondenti ai coefficienti possono essere considerati come ingressi di programmazione e possono essere aggiustati sulla piastra elettronica anche attraverso collegamenti ad hoc (verso $+V_{CC}$ o verso massa, che in logica positiva corrispondono rispettivamente a 1 e 0 logico). C'è tuttavia un metodo

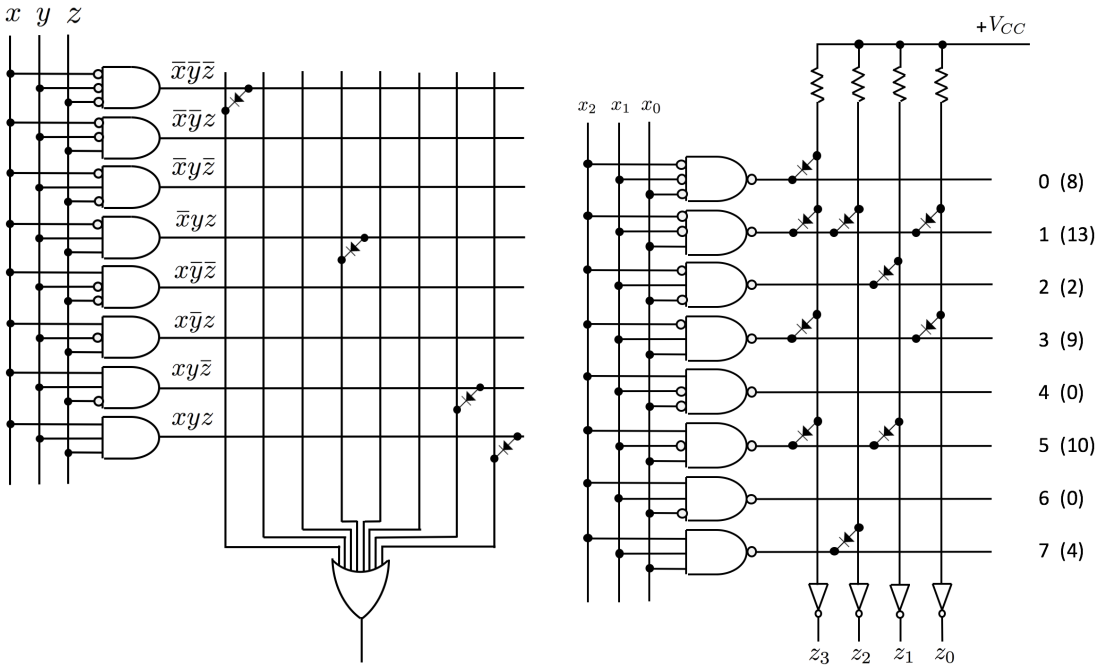


(a) Matrice di contatti a 2 bit

(b) Circuito effettivo per la matrice di contatti a 2 bit

Figura 5.24: Matrice di contatti a 2 bit

più economico per costruire, in modo modulare, una qualunque funzione Boolena di n variabili. Facendo sempre riferimento alla I forma canonica, la sua realizzazione richiede un numero di porte AND a n ingressi pari al numero di *minterm* (cioè 2^n) e una porta OR con un numero di ingressi pari al numero di porte AND (cioè sempre 2^n). Si tratta allora di "prolungare" le uscite delle porte AND agli ingressi della porta OR per i soli *minterm* che entrano nella funzione. La figura 5.24a mostra un circuito di questo genere, chiamato *matrice di contatti*, nel quale si evidenzia la presenza di un contatto per i soli *minterm* che servono per realizzare la funzione $z = \overline{x_1}\, x_2 + x_1\, \overline{x_2}$; nella pratica circuitale la configurazione diventa quella di figura 5.24b. Se ora vogliamo usare questa tecnica per costruire la funzione (5.4) otteniamo il circuito di figura 5.25a. La sua struttura è estremamente vantaggiosa se viene realizzata direttamente dal costruttore di integrati, che privilegia un'alta uniformità nella struttura circuitale. Il confronto con la soluzione a selettori di figura 5.23 è chiarificatore: in quel caso, per realizzare una qualunque funzione di tre variabili occorre un integrato che abbia almeno $3 + 8 + 1 = 12$ piedini (a parte l'alimentazione e la massa). Con la soluzione ora illustrata il numero di piedini sarebbe pari a soli $3 + 1 = 4$. La struttura di figura 5.25a fa uso di diodi per realizzare le connessioni della matrice; inoltre essa può essere estesa in modo tale da fornire non una, ma più funzioni di uscita, passando p.es. a una rete con n ingressi e k uscite. Una simile rete fornisce, per ogni configurazione degli n ingressi, una configurazione sulle k uscite e viene a costituire quella che si chiama una memoria *ROM*, acronimo di *Read Only Memory* (memoria di sola lettura); la memoria in questione ha $M = 2^n$ celle da k bit ciascuna, che vengono indirizzate dalle n linee di indirizzamento. Tenendo conto della realizzazione circuitale effettiva illustrata in figura 5.24b, riportiamo in figura 5.25b la struttura di una memoria ROM con 2^3 celle di memoria da 4 bit ciascuna.
Nella parte destra della figura 5.25b viene riportato l'indirizzo di ciascuna cella e, tra parentesi, il suo contenuto. Per capirne il funzionamento supponiamo che si voglia leggere il contenuto della memoria all'indirizzo 3; ciò significa che deve essere $x_2x_1x_0 = 011$. In tal caso ci sarà una sola porta NAND che ha un'uscita bassa, la porta della riga 3, mentre tutte le altre porte NAND avranno uscita alta. I catodi dei diodi relativi alle due colonne z_3 e



(a) Matrice di contatti a 4 bit che attua la funzione (5.4)

(b) Memoria ROM con 2^3 celle di memoria da 4 bit ciascuna

Figura 5.25: Matrice di contatti e memoria ROM derivabile da essa

z_0 vanno allora a massa, inducendo una polarizzazione diretta per i corrispondenti diodi; sui rispettivi anodi si ha allora una tensione virtualmente nulla (in pratica ci sarà solo la V_{AK} di saturazione, pari $0,3 \div 0,6\ V$). Le linee z_3 e z_0 sono allora basse, e a seguito della complementazione finale diventano alte. Gli altri diodi sulle colonne z_3 e z_0 risultano invece interdetti, le linee rimangono allo stato alto ($+V_{CC}$) e quindi c'è 0 in uscita, a seguito della complementazione finale. Dunque $z_3 = 1$ e $z_0 = 1$, mentre le altre due linee z_2 e z_1 si trovano a zero, e dunque $\boldsymbol{z} = 1001$.

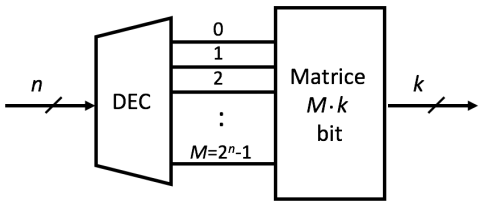
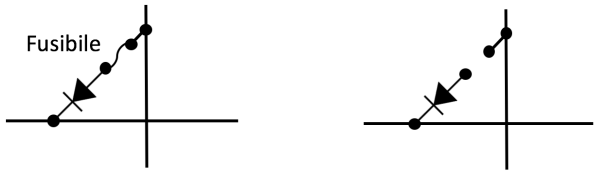


Figura 5.26: Schema a blocchi di una memoria ROM

In figura 5.26 viene dato lo schema a blocchi di una memoria ROM; di fatto si tratta di un decodificatore seguito da una matrice di contatti. Il numero binario corrispondente alla combinazione degli n ingressi rappresenta l'indirizzo della corrispondente cella.
Si noti che fissare i contatti della matrice equivale a programmare il comportamento della rete. Nelle ROM propriamente dette, la matrice non ha inizialmente alcun punto di contatto tra righe e colonne. E' il costruttore che fa le connessioni in base alla matrice di bit desiderata dal committente. Per le memorie ROM, che sono appunto di sola lettura, non è possibile variare il contenuto della memoria dopo la programmazione fatta in fabbrica. Le ROM risultano economicamente convenienti per volumi molto grandi.
Nel caso debba essere l'utente a programmare la ROM si ricorre alle cosiddette *PROM*(*Programmable Read Only Memory*); la matrice ha inizialmente tutti i punti di contatto tra righe e colonne, attraverso un diodo e un fusibile,

come schematizzato a destra in figura 5.27a, e dunque inizialmente tutti i bit sono a 1. Programmare un bit a 0 ri-



(a) Connessione con diodo prima della programmazione

(b) Connessione con diodo dopo la programmazione del bit a 0

Figura 5.27: Modalità di interconnessione tra righe e colonne per le PROM prima e dopo la programmazione di un bit a 0

chiede la fusione del relativo fusibile. Spetta all'utente inserire gli 0 negli incroci desiderati, attraverso un apparato di programmazione. Chiaramente non è possibile modificare il contenuto della memoria dopo la programmazione. Le PROM risultano economicamente convenienti per volumi medio/grandi.
Esiste anche la possibilità di modificare la programmazione ricorrendo alle *EPROM* (*Erasable* PROM), basate sulla tecnologia dei *Floating Gate MOSFET*; la memoria è in un contenitore che presenta una piastrina di quarzo, attraverso la quale possono passare raggi ultravioletti che ripristinano la programmabilità cancellando la programmazione precedente (si veda la figura *fig:EPROM*). Le EPROM risultano convenienti per prototipi di laboratorio o per bassissimi volumi di produzione. Sulla base della tecnologia EPROM, nel 1978 vennero sviluppate delle me-

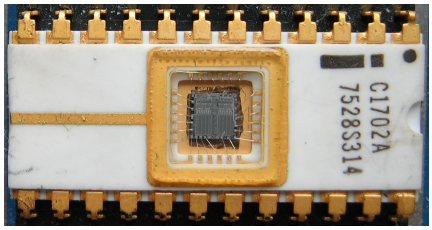


Figura 5.28: La prima EPROM realizzata da INTEL nel 1971, da 256 byte

morie cancellabili e riprogrammabili elettricamente, le cosiddette *EEPROM* (*Electrically Erasable* PROM), che fanno uso di MOSFET con uno strato molto più sottile di ossido per il gate. Queste memorie sono indicate per un impiego che prevede la possibilità di modificarne il contenuto direttamente dall'apparato in cui vengono usate.

5.5 Moduli per la realizzazione dell'unità logico-aritmetica

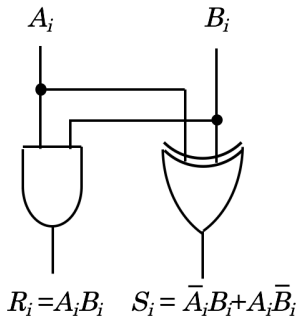
Come vedremo nel capitolo 7, uno degli elementi centrali nell'architettura di un calcolatore è l'*Unità Logico Aritmetica*, meglio nota con l'acronimo *ALU* - che sta per *Arithmetic Logic Unit*; essa lavora a stretto contatto con i registri di memoria, guidata dall'azione dell'*Unità di Controllo*. Nella ALU si realizzano tipicamente operazione su numeri interi quali somma, sottrazione, incremento, decremento, scorrimento di bit, ma si attuano anche operazioni logiche sui dati, quali AND, OR, XOR o complementazione; le ALU più recenti contengono anche moduli per eseguire direttamente prodotti e moltiplicazioni.

5.5.1 Il semisommatore e il sommatore completo

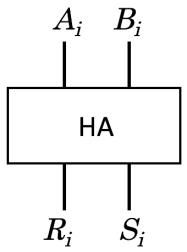
Il nucleo di partenza per costruire una ALU è il modulo *semisommatore* (o *Half Adder*), che realizza la somma di due bit con riporto. In figura 5.29a è riprodotta la tabella aritmetica della somma bit per bit con riporto,

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

(a) Tavola di verità del semisommatore



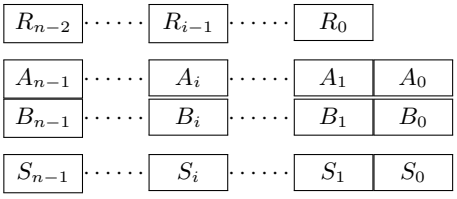
(b) Circuito del semisommatore



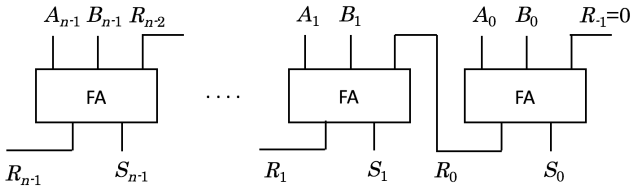
(c) Simbolo circuitale del semisommatore

Figura 5.29: Il semisommatore

che corrisponde alla tavola di verità di due funzioni Booleane, una che realizza la somma S_i e una che realizza il riporto R_i, per ogni coppia A_i, B_i dei bit d'ingresso. Si riconosce immediatamente che la S_i corrisponde a uno XOR, mentre la R_i è un AND; di conseguenza il modulo ha la realizzazione circuitale di figura 5.29b, mentre la figura 5.29c rappresenta il suo simbolo circuitale.
Se ora vogliamo effettuare la somma completa tra due numeri interi $\boldsymbol{A} = [A_{n-1}\ A_{n-2}\ ...\ A_1\ A_0]$ e $\boldsymbol{B} = [B_{n-1}\ B_{n-2}\ ...\ B_1\ B_0]$, espressi in notazione posizionale con n bit, dobbiamo costruire una rete che accetti in ingresso $2n$ bit e generi la somma binaria dei due, $\boldsymbol{S} = [S_{n-1}\ S_{n-2}\ ...\ S_1\ S_0]$, secondo il classico procedimento di somma con riporto evidenziato in figura 5.30a. Partendo da destra si inizia a sommare A_0 con B_0; detto R_0 il



(a) Procedimento di somma con riporto tra due numeri interi $\boldsymbol{A} = [A_{n-1}\ A_{n-2}\ ...\ A_1\ A_0]$ e $\boldsymbol{B} = [B_{n-1}\ B_{n-2}\ ...\ B_1\ B_0]$



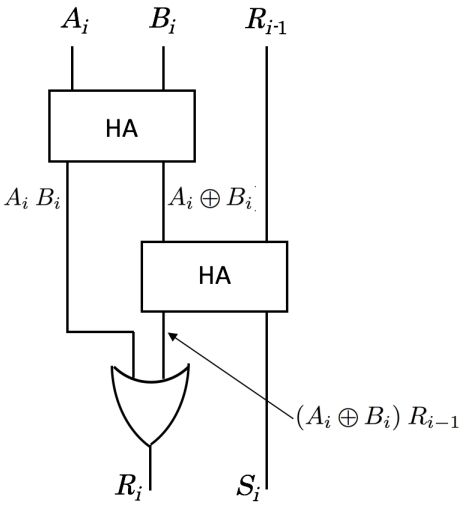
(b) Sommatore completo

Figura 5.30: Somma con riporto e relativo circuito sommatore

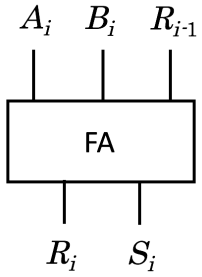
riporto, questi dovrà essere sommato con la somma di A_1 e B_1, che genera R_1; questo va sommato con la somma di A_2 e B_2 e così via. E' evidente che il riporto R_{-1} della colonna $A_0 \| B_0$ vale 0 e che se l'ultimo riporto, R_{n-1} dovesse valere 1, siamo di fronte a una situazione di *overflow*, che richiede il passaggio alla notazione a virgola mobile. Per realizzare un tale circuito bisogna modificare il semisommatore, in modo da includere il contributo R_{i-1} del riporto del passo precedente. In questo modo si ottiene il *sommatore completo* (o *full adder*), la cui tavola di verità è riportata in figura 5.31a. La somma di A_i, B_i e R_{i-1} vale 1 solo quando c'è un numero dispari di 1 nella somma, ed è quindi lo XOR dei tre bit; R_i vale 1 quando A_i e B_i sono entrambi a 1 (qualunque sia il valore di R_{i-1}), oppure quando $R_{i-1} = 1$ e $A_i \oplus B_i = 1$. Per ricavare in modo formale l'espressione risolutiva scriviamo i *minterm* di entrambe le funzioni che si ricavano dalla tavola di verità di figura 5.31a e procediamo con

A_i	B_i	R_{i-1}	S_i	R_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Tavola di verità del sommatore completo



(b) Circuito del sommatore completo



(c) Simbolo circuitale del sommatore completo

Figura 5.31: Il sommatore completo

le semplificazioni, ricordando che $x \oplus y = x\bar{y} + \bar{x}y$ e che $\overline{x \oplus y} = \bar{x}\bar{y} + xy$

$$\begin{aligned} S_i &= \overline{A_i}\,\overline{B_i}R_{i-1} + \overline{A_i}B_i\overline{R_{i-1}} + A_i\overline{B_i}\,\overline{R_{i-1}} + A_iB_iR_{i-1} \\ &= (\overline{A_i}\,\overline{B_i} + A_iB_i)R_{i-1} + (\overline{A_i}B_i + A_i\overline{B_i})\overline{R_{i-1}} \\ &= (\overline{A_i \oplus B_i})R_{i-1} + (A_i \oplus B_i)\overline{R_{i-1}} \\ &= (A_i \oplus B_i) \oplus R_{i-1} \\ R_i &= \overline{A_i}B_iR_{i-1} + A_i\overline{B_i}R_{i-1} + A_iB_i\overline{R_{i-1}} + A_iB_iR_{i-1} \\ &= (\overline{A_i}B_i + A_i\overline{B_i})R_{i-1} + A_iB_i \\ &= (A_i \oplus B_i)\ R_{i-1} + A_i\ B_i \end{aligned} \tag{5.5}$$

e da queste equazioni si ricava direttamente la struttura circuitale di figura 5.31b, che viene rappresentata simbolicamente come in figura 5.31c. Tale realizzazione ha anche il pregio di sfruttare la modularità del semisommatore.

5.5.2 Calcolo della differenza mediante sommatore

Nel paragrafo 2.3.2 si è visto che i numeri negativi vengono rappresentati con la notazione mediante complemento a 2. Ciò significa che la differenza $\boldsymbol{A} - \boldsymbol{B}$ si realizza come $\boldsymbol{A} + (-\boldsymbol{B})$, dove $-\boldsymbol{B}$ si ottiene complementando $\boldsymbol{B}$ e sommando 1. Si ha dunque

$$\boldsymbol{A} - \boldsymbol{B} = \boldsymbol{A} + (-\boldsymbol{B}) = \boldsymbol{A} + \overline{\boldsymbol{B}} + 1 \tag{5.6}$$

Il circuito di figura 5.32 offre la possibilità di fare la differenza tra $\boldsymbol{A}$ e $\boldsymbol{B}$ secondo il principio appena esposto; il funzionamento è il seguente. Il valore assunto dalla linea di controllo C_B determina se il valore $\boldsymbol{B}$ viene o meno complementato prima di entrare nel sommatore; infatti se $C_B = 0$ si attivano gli AND sulla sinistra, che fanno passare $\boldsymbol{B}$; se invece $C_B = 1$ si attivano gli AND sulla destra, che fanno passare il suo complementare $\overline{\boldsymbol{B}}$. Se vogliamo fare la differenza $\boldsymbol{A} - \boldsymbol{B} = \boldsymbol{A} + (-\boldsymbol{B}) = \boldsymbol{A} + \overline{\boldsymbol{B}} + 1$, dobbiamo allora porre $C_B = 1$ e aggiungere 1 tramite R_{-1}. La linea di controllo C_A serve invece per azzerare l'ingresso $\boldsymbol{A}$; infatti se $C_A = 1$, in ingresso al sommatore viene mandato $\boldsymbol{A}$, altrimenti si hanno degli 0. La tabella 5.33 mostra il risultato delle varie combinazioni di C_B, C_A, R_{-1}. Le tre porte di sinistra di figura 5.32 servono invece per dare un segnale di allarme quando si è in presenza di *overflow*; infatti dalla tavola 2.14 si può osservare che tale condizione si realizza quando si verificano le seguenti condizioni:

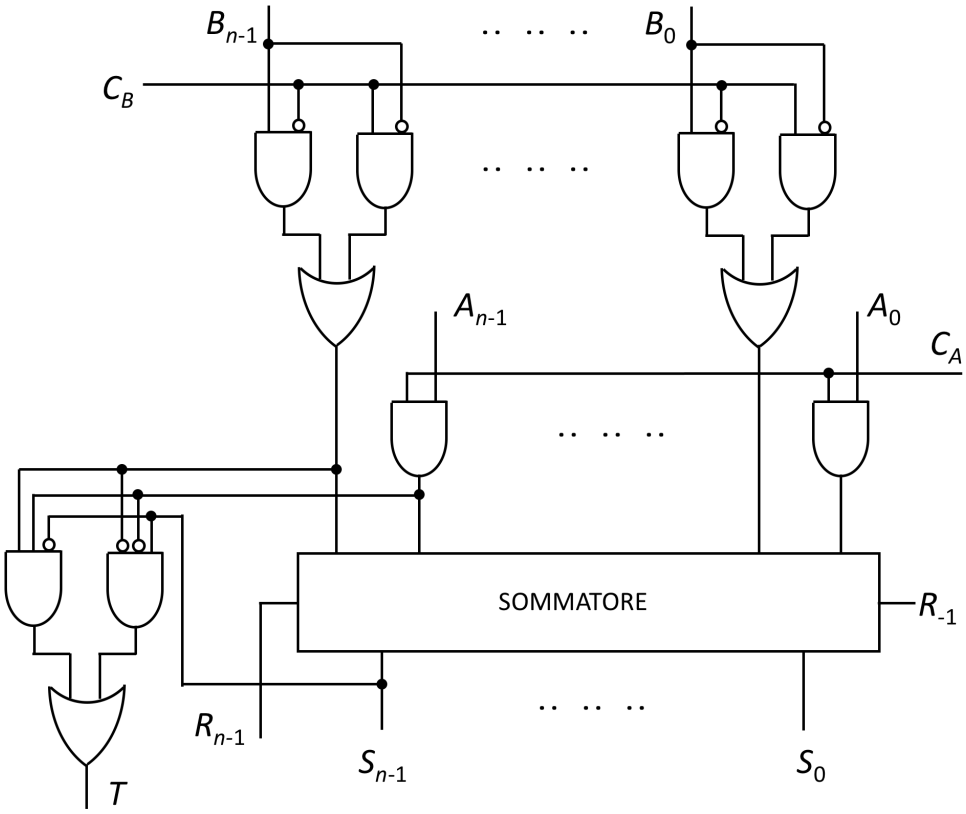


Figura 5.32: Circuito sommatore che può eseguire anche la differenza

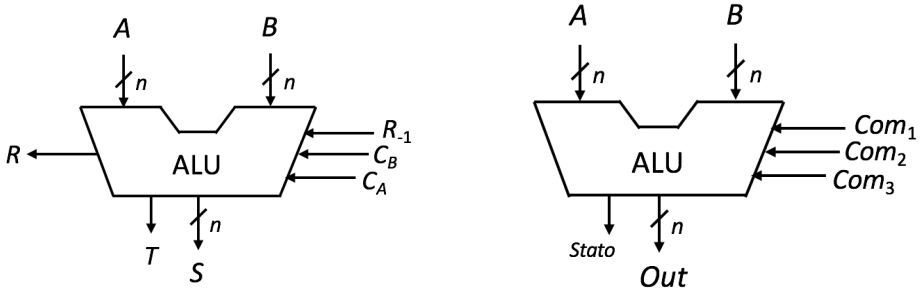
- si sommano due numeri negativi (cioè con 1 nella prima posizione) e il risultato è positivo (0 nella prima posizione);

- si sommano due numeri positivi (0 nella prima posizione) e il risultato è negativo (cioè con 1 nella prima posizione)

La prima delle due condizioni si ha con $A_{n-1} = 1$, $B_{n-1} = 1$, $S_{n-1} = 0$, e quando si realizza fornisce un uscita 1 nella porta AND di sinistra; la seconda condizione si ha con $A_{n-1} = 0$, $B_{n-1} = 0$, $S_{n-1} = 1$ e quando si realizza fornisce un uscita 1 nella porta AND di destra. La porta OR raccoglie dunque l'unione logica dei due eventi. Il segnale di allarme viene usato per commutare nella rappresentazione a virgola mobile.
In figura 5.34a viene illustrato il simbolo schematico della ALU descritta dalla rete 5.32. Con una piccola modifica della circuiteria è possibile fare in modo da realizzare, oltre alla somma e alla differenza tra $\boldsymbol{A}$ e $\boldsymbol{B}$, anche l'AND e l'OR tra i due ed eventuali altre operazioni logiche; lo schema di figura 5.34b mostra una ALU completa, nella quale i comandi C_A, C_B, R_{-1} e tutti gli altri per attivare le varie operazioni logiche sono rappresentati con la notazione $\text{Com}_1, \text{Com}_2, ..., \text{Com}_n$. Invitiamo il lettore interessato ad approfondire la questione sul testo di Bucci [2].

C_A	C_B	R_{-1}	Operazione	Descrizione
0	0	0	$\boldsymbol{S} = 0 + \boldsymbol{B}$	Selezione di $\boldsymbol{B}$
0	0	1	$\boldsymbol{S} = 0 + \boldsymbol{B} + 1 = \boldsymbol{B} + 1$	Incremento di $\boldsymbol{B}$
0	1	0	$\boldsymbol{S} = 0 + \overline{\boldsymbol{B}} = \overline{\boldsymbol{B}}$	Complementazione di $\boldsymbol{B}$
0	1	1	$\boldsymbol{S} = 0 + \overline{\boldsymbol{B}} + 1 = -\boldsymbol{B}$	Cambio segno di $\boldsymbol{B}$
1	0	0	$\boldsymbol{S} = \boldsymbol{A} + \boldsymbol{B}$	Somma $\boldsymbol{A} + \boldsymbol{B}$
1	0	1	$\boldsymbol{S} = \boldsymbol{A} + \boldsymbol{B} + 1$	
1	1	0	$\boldsymbol{S} = \boldsymbol{A} + \overline{\boldsymbol{B}} = \boldsymbol{A} - \boldsymbol{B} - 1$	
1	1	1	$\boldsymbol{S} = \boldsymbol{A} + \overline{\boldsymbol{B}} + 1 = \boldsymbol{A} - \boldsymbol{B}$	Differenza $\boldsymbol{A} - \boldsymbol{B}$

Figura 5.33: Operazioni effettuate dalla rete di figura 5.32 a seconda dei valori assunti dagli ingressi di controllo C_A, C_B, R_{-1}



(a) Simbolo schematico della ALU descritta dalla rete 5.32

(b) Simbolo schematico di una generica ALU dotata di linee di comando per generare operazioni logiche tra gli ingressi

Figura 5.34: Simbolo schematico di una ALU

Le moderne ALU vengono integrate con circuiti che realizzano la moltiplicazione e la divisione tra numeri interi, anche per questi circuiti dedicati invitiamo a consultare [2].