Capitolo 3

PROGETTO LOGICO DELLE RETI COMBINATORIE

II Parte - Reti a più di due livelli di logica e Reti modulari

3.1 - Introduzione

Le reti a due livelli di logica, pur essendo particolarmente apprezzabili per le doti di velocità, non si prestano ad essere utilizzate in qualunque situazione per due ragioni principali:

a) non sempre queste reti presentano valori minimi dei parametri N_g ed N_d rispetto a reti con più di due livelli di logica;

b) esistono vincoli di natura tecnologica sul numero massimo di segnali di ingresso e di uscita che limitano la pratica realizzabilità delle reti a due livelli per qualsiasi applicazione.

Esaminiamo brevemente queste due categorie di problemi.

a) Per il primo punto, un semplice esempio risulta significativo per dimostrare come reti a due livelli possano essere più costose di quelle a più livelli, in termini di valori di N_g ed N_d. Consideriamo la funzione "generatore di parità", ossia una funzione di n variabili che assume valore 1 se il numero di variabili con valore 1 è dispari e 0 altrimenti. Tale funzione è descritta da una mappa di Karnaugh configurata a scacchiera, nella quale una casella di 1 è circondata da n caselle di 0 e viceversa, come viene mostrato nella Fig. 3.1 per $n = 4$.

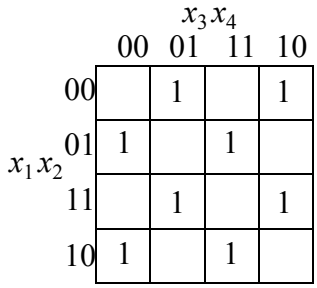


Fig. 3.1

E' facile verificare che l'espressione che si ricava da questa mappa è l'OR esclusivo delle n variabili, ovvero:

$$f(x_0, x_1, ..., x_{n-1}) = x_0 \oplus x_1 \oplus ... \oplus x_{n-1}$$

Nella realizzazione a due livelli di logica, poichè non esistono adiacenze tra le caselle di 1, tutti gli implicanti primi (che sono anche essenziali) hanno n letterali ed il loro numero è pari a metà delle caselle, ovvero 2^{n-1}. La realizzazione richiede quindi 2^{n-1} porte AND ciascuna con n ingressi, seguite da una porta OR con 2^{n-1} ingressi; i parametri N_g ed N_d pertanto valgono:

$$N_g = 2^{n-1}+1$$
$$N_d = n \cdot 2^{n-1}+2^{n-1} = (n+1) \cdot 2^{n-1}$$

Nel caso di $n = 8$ risulta $N_g = 129$ e $N_d = 1152$, ossia valori chiaramente non realistici.

Tuttavia la funzione in esame può essere realizzata anche collegando tra loro n-1 blocchi XOR a due ingressi secondo una struttura ad albero, come illustrato nella figura seguente:

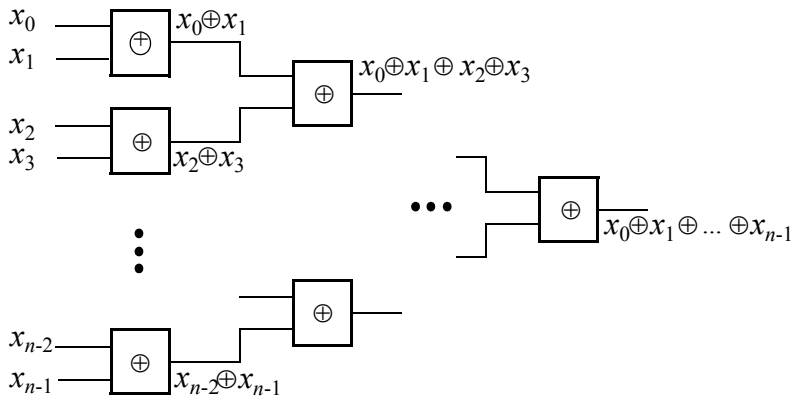
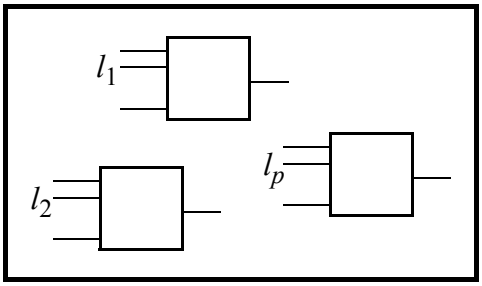


Fig. 3.2

Poichè ogni blocco XOR richiede tre porte (due AND ed un OR) con due ingressi ciascuna, avremo $N_g = 3 \cdot (n-1)$ ed $N_d = 3 \cdot 2 \cdot (n-1)$, con una dipendenza lineare da n, anzichè esponenziale. Nel caso di $n = 8$ i valori di N_g ed N_d scendono rispettivamente a 21 e 42.

b) Per quanto riguarda il secondo tipo di problema, in una rete combinatoria con n ingressi, m uscite e p porte chiamiamo **fan-out** di un ingresso della rete x_i , $0 \le i \le n$-1 , o di un'uscita z_j di una porta, $0 \le j \le p$-1, il numero di porte alimentate da x_i o da z_j. Chiamiamo quindi **fan-out della rete** il massimo tra i fan-out dei suoi ingressi e delle uscite delle sue porte.

Si dice invece **fan-in** di una porta il numero dei suoi ingressi; **fan-in di una rete** è il massimo fan-in delle sue porte (Fig. 3.3).



fan-in = max$\{l_1, l_2, ..., l_p\}$

Fig. 3.3

Per questioni di natura tecnologica il fan-in ed il fan-out di una porta logica non possono superare determinati limiti. Poichè nelle reti a due livelli di logica il fan-in (ed anche il fan-out) possono assumere valori anche molto elevati, come nell'esempio di Fig. 3.1, tali reti non possono essere sempre realizzate, ma occorre ripiegare su strutture a più livelli di logica, derivate da quelle a due livelli attraverso l'uso della proprietà distributiva dell'algebra e in grado di mantenere il fan-in ed il fan-out dei singoli blocchi a livelli accettabili; ad esempio possono essere utilizzate strutture ad albero, come quella indicata nella Fig. 3.4a, detta **libera da fan-in** o quella della Fig. 3.4b, detta **libera da fan-out**.

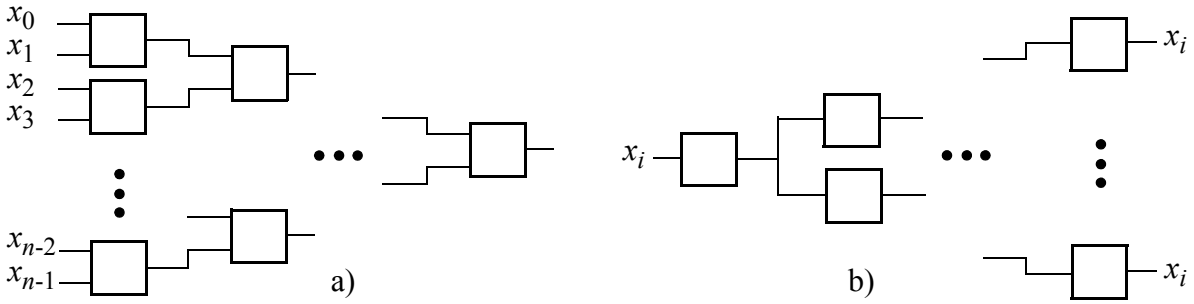


Fig. 3.4

Si osservi che in Fig. 3.4b i vari blocchi dell'albero non svolgono alcuna funzione logica, tranne riportare in uscita il segnale di ingresso convenientemente amplificato in modo da eliminare il problema del fan out. Essi prendono il nome di **buffer**.

Purtroppo per le reti a più di due livelli di logica non si conoscono metodi sistematici di sintesi ottima e di conseguenza, nella ricerca della soluzione praticabile, si devono seguire vari approcci, più o meno basati su criteri euristici.

3.2 - Partizionamento

Nel caso di reti complesse un utile approccio alla sintesi può essere quello di cercare di suddividere l'intera rete richiesta per un dato problema in più sottoreti da collegare opportunamente tra loro; nella scomposizione si cerca di trovare un compromesso accettabile tra fan-in e/o fan-out e velocità di risposta. In pratica si tenta di individuare nella formulazione del problema combinatorio una serie di funzioni logiche abbastanza semplici, le quali concorrono a definire l'intera funzione richiesta per la soluzione. Per ciascuna funzione si progetta, possibilmente in forma minima, la rete corrispondente e infine tutte le reti vengono interconnesse secondo la logica definita dal problema. Vedremo nel prossimo paragrafo che molto spesso, accanto a funzioni generiche, è possibile riconoscere un numero - abbastanza ridotto - di funzioni tipiche ricorrenti frequentemente, alle quali corrispondono altrettante reti logiche utilizzabili come blocchi standard secondo un criterio modulare. E' chiaro che più è fine il grado di partizionamento e più semplici e con bassi valori di fan-in e fan-out risultano le reti componenti, ma spesso a scapito della velocità.

Per esempio si consideri il seguente problema:

> Progettare una rete combinatoria la quale, dati tre numeri interi di n bit, A, B, C, con $A > B$, produce in uscita C se risulta $\min\{B, A\text{-}B\} < C < \max\{B, A\text{-}B\}$; altrimenti produce in uscita 0.

Per risolvere il problema, isoliamo le funzioni: a) calcolo della differenza tra A e B; b) calcolo del minimo tra B e la differenza A-B; c) calcolo del massimo tra B e la differenza A-B; d) confronto tra il minimo e C; e) confronto tra C ed il massimo; f) calcolo del valore logico che indica se entrambe le condizioni sono vere o almeno una delle due è falsa; g) scelta tra il numero C ed il numero 0 da far uscire dalla rete a seconda di quel valore. Associando un blocco logico a ciascuna funzione individuata e collegando i vari blocchi secondo l'ordine appropriato, si arriva al seguente schema a blocchi della rete (Fig. 3.5).

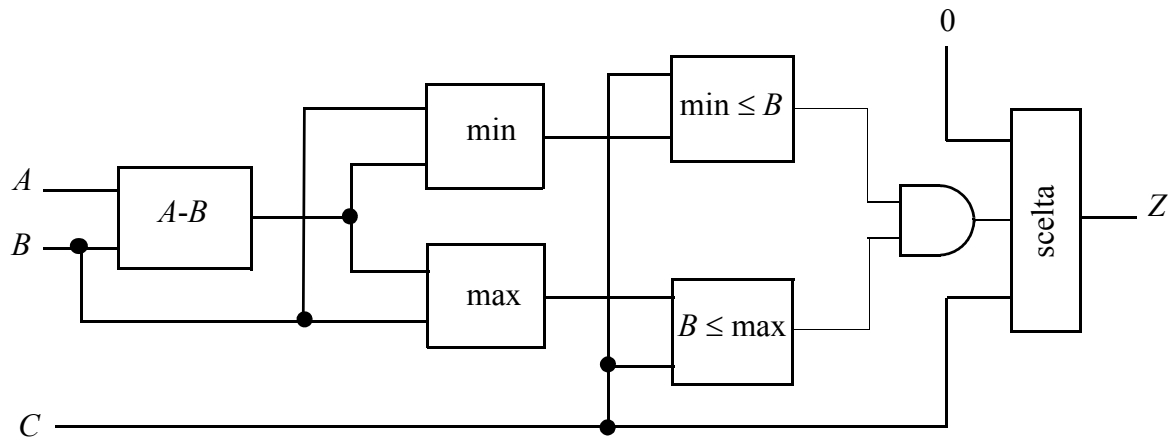


Fig. 3.5

Nella figura i vari blocchi sono identificati con la funzione che svolgono; le linee marcate indicano segnali di *n* bit, quelle sottili segnali di un solo bit. Per completare la realizzazione della rete logica rimane soltanto da sintetizzare a livello porte i vari blocchi, cercando, se possibile, di progettarli come reti a due livelli, esprimendone la funzione in forma SP o PS; così il blocco denominato "scelta" può essere realizzato sulla base della tavola di verità della sua funzione: detta x la variabile di uscita della porta AND e C_i il bit i-esimo del numero C e Z_i il corrispondente bit dell'uscita Z, si ha:

x	Z_{n-1}	...	Z_0
0	0	...	0
1	C_{n-1}	...	C_0

Vale la pena notare che se il tempo di risposta è un parametro progettuale importante, si può migliorare la soluzione eliminando la porta AND e integrando nel blocco "scelta" la funzione f), per cui dette x_0 e x_1 le uscite dei due blocchi comparatori, la tavola di verità di "scelta" si modifica nel seguente modo:

x_1	x_0	Z_{n-1}	...	Z_0
0	0	0	...	0
0	1	0	...	0
1	0	0	...	0
1	1	C_{n-1}	...	C_0

Ovviamente questo va ad aumentare la complessità strutturale del blocco "scelta" ed in particolare il suo fan-in.

Una particolare tecnica di partizionamento, frequentemente utilizzata, è la **fattorizzazione**. Essa consiste nel porre a fattore comune sottoespressioni che sono presenti in più termini prodotto o somma di espressioni booleane SP o PS , sfruttando la proprietà associativa del prodotto rispetto alla somma o della somma rispetto al prodotto. In questo modo una rete a due livelli viene spezzata in una cascata di sottoreti più semplici.

Per esempio nell'espressione $f = x_1x_2x_3\bar{x}_4 + x_1x_2\bar{x}_3x_4$ entrambi i prodotti hanno in comune il termine x_1x_2; perciò l'espressione può essere riscritta come $f = x_1x_2(x_3\bar{x}_4 + \bar{x}_3x_4)$ la quale, pur avendo valori maggiori di N_g ed N_d, consente di realizzare una rete con un fan-in inferiore.

La fattorizzazione può essere applicata iterativamente ad una stessa espressione, ottenendo ad ogni passo un aumento del numero dei livelli di logica, insieme alla riduzione della complessità del fan-in o del fan-out.

3.3 - Reti combinatorie modulari

In una rete logica di una certa complessità, accade spesso che accanto a porzioni che realizzano funzioni particolari, specifiche del problema e che prendono genericamente il nome di **logica sparsa**, ve ne siano altre per funzioni più generali, indipendenti dal problema specifico. Data la loro caratteristica, per queste funzioni non è necessario progettare reti ad hoc per ogni loro occorrenza, ma si ricorre a blocchi logici generali, identificati con un nome legato alla funzione svolta e di volta in volta è necessario solo definire alcuni parametri, come numero di ingressi e di uscite. Questi blocchi possono essere costruiti con tecnologia a media e larga scala di integrazione (MSI e LSI) sotto forma di circuiti integrati (**chip**) da montare su piastre a circuito stampato, per la realizzazione a componenti discreti di una rete logica, oppure possono essere integrati su silicio insieme all'intera rete, sotto forma di **chip custom** o di circuiti **ASIC**. In ogni caso il loro utilizzo consente di dare alla rete una notevole regolarità strutturale che facilita la costruzione e la testabilità, diminuisce i costi di progettazione e di fabbricazione, aumenta l'affidabilità.

Un'altra caratteristica interessante di questi blocchi funzionali è la **modularità**, ovvero la possibilità di realizzare la loro funzione specifica per un determinato numero di ingressi ricorrendo a blocchi della stessa funzionalità, ma per un numero di ingressi inferiore, con tecnica ricorsiva. In questo caso il parametro che ha più interesse minimizzare è il numero totale dei moduli elementari, piuttosto che i parametri N_g o N_d, in quanto questo porta a minimizzare il numero dei circuiti integrati necessari.

Nel seguito esamineremo alcune classi di funzionalità combinatorie di frequente uso nelle tecniche digitali, realizzabili mediante reti a due o più livelli di logica, seguendo l'approccio modulare descritto.

3.4 - Convertitori di codice

A questa classe di funzionalità appartengono tutte le funzioni combinatorie che operano una trasformazione da un codice di rappresentazione di ingresso ad un diverso codice di uscita; con il nome di **codificatori** e **decodificatori** sono chiamate le reti che realizzano tali funzioni.

3.4.1 - Codificatori

Un codificatore $2^n \times n$ è un circuito con 2^n ingressi, $x_{2^n-1}, x_{2^n-1-1}, \ldots, x_0$, ed n uscite, $z_{n-1}, z_{n-2}, \ldots, z_0$, nel quale le combinazioni possibili dei valori degli ingressi sono quelle che contengono un solo 1, ovvero quelle per le quali è:

$$x_i = 1 \quad 0 \le i \le 2^n\text{-}1$$
$$x_j = 0 \quad j \ne i$$

In corrispondenza a ciascuna di tali combinazioni di ingresso, in uscita si presenta la sequenza binaria:

$$z_{n\text{-}1}, z_{n\text{-}2}, \ldots, z_0 \equiv i$$

ovvero le variabili z_h, $0 \le h \le n\text{-}1$, assumono una combinazione che rappresenta in binario l'intero i, $0 \le i \le 2^n\text{-}1$. In altri termini se si considerano le linee di ingresso ordinate da 0 a $2^n\text{-}1$, il valore dell'uscita indica il numero della linea di ingresso (contando a partire da 0)sulla quale è presente il valore logico 1. Per esempio un codificatore $2^2\times2$ è definito dalla seguente tavola di verità:

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

Per tutte le rimanenti combinazioni di ingresso le uscite non sono specificate. E` facile verificare con l'ausilio delle mappe di Karnaugh che le espressioni delle uscite sono:

$$z_0 = x_1 + x_3$$
$$z_1 = x_2 + x_3$$

Espressioni di questo tipo valgono in generale, ovvero ogni variabile di uscita è la somma logica di opportuni sottoinsiemi delle variabili di ingresso. Da tali espressioni deriva direttamente la struttura del codificatore, di cui la figura seguente mostra un esempio riferito al caso esaminato, insieme alla rappresentazione di un codificatore di dimensioni generiche $2^n\times n$ come blocco funzionale:

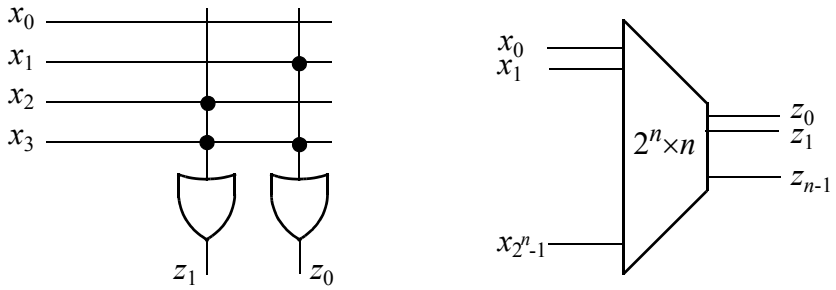


Fig. 3.6

Una applicazione dei codificatori è la seguente. In un sistema di elaborazione, affinché un

dispositivo di ingresso/uscita (periferica) possa comunicare con il processore, è necessario che ne richiami l'attenzione e si identifichi. Queste due azioni possono essere svolte insieme predisponendo per ogni periferica una linea di richiesta connessa al sistema attraverso un codificatore. In tal modo la periferica è individuata da un indirizzo, definito implicitamente dalla posizione della sua linea di richiesta all'ingresso del codificatore; l'attivazione di una di queste linee fornisce pertanto anche l'informazione necessaria per identificare la periferica (Fig. 3.7).

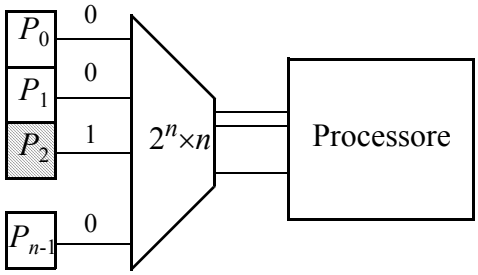


Fig. 3.7

3.4.2 - Decodificatori

I decodificatori sono reti molto importanti nelle applicazioni digitali, prime fra tutte quelle riguardanti la realizzazione di dispositivi di memoria. Limitandoci per semplicità ai **decodificatori binari**, si tratta di circuiti con n ingressi e 2^n uscite tali che per ciascuna delle 2^n combinazioni binarie degli ingressi solo una assume valore 1, mentre le altre sono 0. In altre parole un decodificatore interpreta ogni combinazione di ingresso come un numero naturale i nell'intervallo $[0, 2^n\text{-}1]$ e corrispondentemente attiva la linea di uscita che si trova nella posizione i. Poichè per ogni combinazione di ingresso solo uno dei possibili mintermini delle n variabili vale 1, le funzioni di uscita di un decodificatore sono semplicemente

$$z_i = x_0^* \cdot x_1^* \cdot \ldots \cdot x_{n-1}^*, \qquad x_h^* \in \{x_h, \bar{x}_h\}$$

e possono essere realizzate con una rete costituita da un unico livello di porte AND, almeno finchè il loro fan-in lo consente.

Per esempio un decodificatore 2 a 4 (4 out of 2) è descritto dalla seguente tavola di verità:

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

dalla quale si ricavano direttamente le espressioni delle uscite:

$$z_0 = \bar{x}_1 \bar{x}_0$$
$$z_1 = \bar{x}_1 x_0$$
$$z_2 = x_1 \bar{x}_0$$
$$z_3 = x_1 x_0$$

La Fig. 3.8 riporta la struttura del decodificatore esemplificato ed il diagramma a blocchi generale:

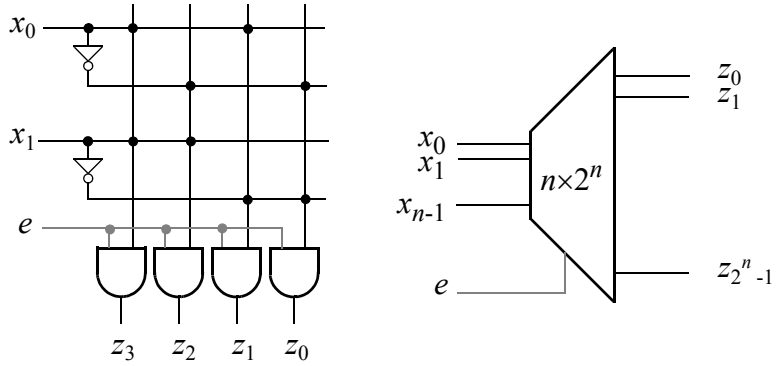


Fig. 3.8

La figura inoltre mostra che, se oltre agli ingressi x_i, $i = 0, ..., n-1$, si applica a tutte le porte AND uno stesso segnale binario e, si ottiene un decodificatore il cui funzionamento dipende proprio da e, nel senso che quando $e = 0$, le uscite sono tutte 0, indipendentemente dal valore attuale degli ingressi x_i, mentre per $e = 1$, si realizza il normale funzionamento. L'ingresso e agisce come **controllo**, e viene indicato con il termine *enable* o *select* (linea tratteggiata in Fig. 3.8); invece gli ingressi x_i costituiscono i **dati**.

Quando il numero di ingressi è grande al punto che non è possibile realizzare un circuito di decodifica con un solo livello di porte, si ricorre a varie soluzioni le quali realizzano la rete sulla base di decodificatori di dimensioni più piccole.

Per esempio un decodificatore 3×8 può essere realizzato con due decodificatori 2×4 ciascuno dei quali sia provvisto dell' ingresso di enable, che viene alimentato dal bit più significativo per un decodificatore e dal suo complemento per l'altro, secondo lo schema a blocchi di Fig. 3.9:

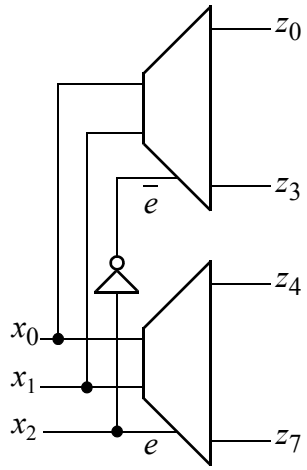


Fig. 3.9

Un metodo alternativo a quello ora illustrato consiste nel realizzare ricorsivamente un decodificatore $n\times2^n$ mediante due decodificatori $n/2\times2^{n/2}$ ed una matrice di porte AND $n\times n$; per esempio un decodificatore 4×16 può essere realizzato con due decodificatori 2×4 ed una matrice 4×4 di AND, come nella Fig. 3.10:

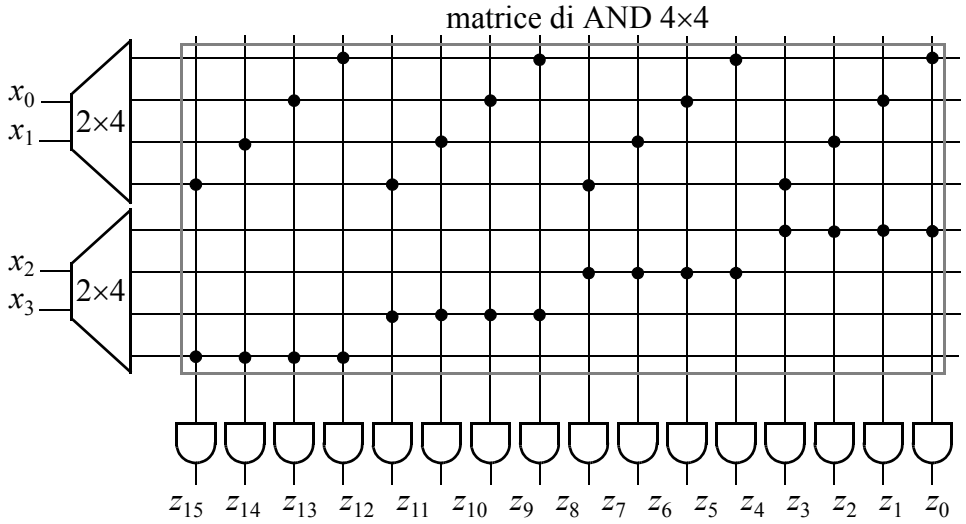


Fig. 3.10

A sua volta con due decodificatori 4×16 come quelli di Fig. 3.10 ed una matrice di porte AND 16×16 si può realizzare un decodificatore 8×256 e così via. Poichè tuttavia sono disponibili **matrici generalizzate di porte**, ovvero matrici $h\times h\times...\times h$ (k volte) aventi $k\cdot h$ ingressi riuniti in k gruppi di h ingressi ciascuno e h^k uscite e nelle quali ogni porta AND ha k ingressi, uno per ognuno dei k gruppi di ingressi della matrice, un decodificatore 8×256 può essere realizzato anche partendo da 4 decodificatori 2×4 ed una matrice 4×4×4×4, com'è illustrato nella Fig. 3.11. La scelta della tecnica realizzativa dipende come sempre da

considerazioni tecnologiche e di costo: se ad esempio il criterio è la minimizzazione di N_g, è da preferire la seconda soluzione che richiede meno porte.

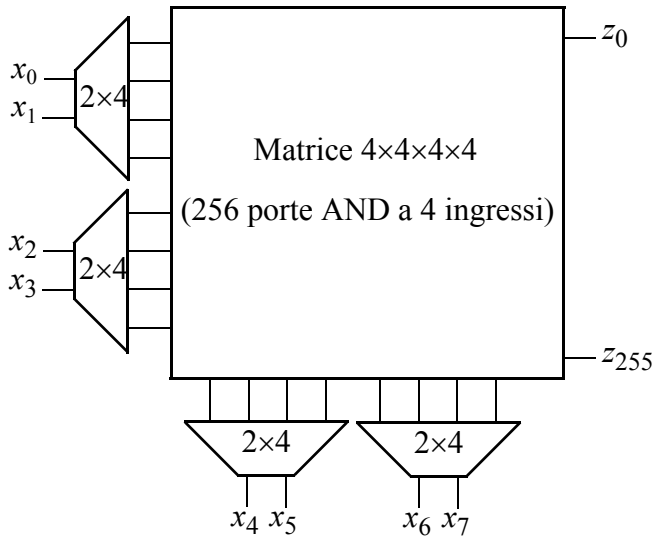


Fig. 3.11

3.5 - Selettori

Due funzionalità molto vicine alla decodifica sono quelle che effettuano la selezione di un segnale tra più, sia in ingresso (multiplexing) sia in uscita (demultiplexing). Le reti combinatorie che realizzano queste funzioni sono dette rispettivamente **multiplexer** o **selettore di ingresso** e **demultiplexer** o **selettore di uscita**.

3.5.1 - Multiplexer

Un multiplexer è un circuito che seleziona uno tra n segnali in ingresso e lo instrada su un'unica linea di uscita z, mediante l'uso di k segnali di controllo $y_0, y_1, ..., y_{k-1}$, $k = \lceil \log_2 n \rceil$ (Fig. 3.12).

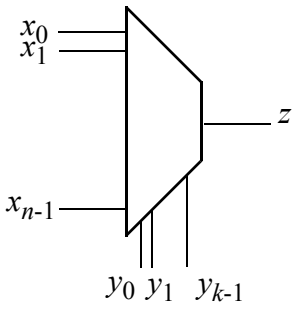


Fig. 3.12

La funzione di uscita $z(x_0, x_1, ..., x_{n-1}, y_0, y_1, ..., y_{k-1})$ si ottiene tenendo conto che per ogni combinazione dei segnali di controllo solo un segnale di ingresso deve essere selezionato; pertanto essa deve essere la somma logica dei prodotti di ciascun segnale x_i per uno dei $2^k \geq n$ mintermini delle k variabili di controllo, ossia:

$$\begin{aligned} z &= p_0 x_0 + p_1 x_1 + \ldots + p_{n-1} x_{n-1} \\ &= \bar{y}_0 \bar{y}_1 \cdots \bar{y}_{k-1} x_0 + \bar{y}_0 \bar{y}_1 \cdots y_{k-1} x_1 + \ldots + y_0 y_1 \cdots y_{k-1} x_{n-1} \end{aligned}$$

Per esempio nel caso di $n = 2$, $k = 1$, si ottiene la semplice rete di Fig. 3.13, governata dalla funzione $z = \bar{y}x_0 + yx_1$:

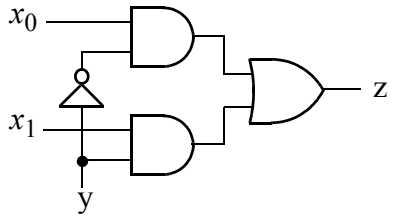


Fig. 3.13

Un multiplexer è una rete a due livelli di logica che, per quanto riguarda le varibili di controllo, è in forma canonica e come tale può essere sempre realizzata con n porte AND a k+1 ingressi ed una porta OR ad n ingressi. Tuttavia dall'espressione dell'uscita si deduce facilmente che il multiplexer è realizzabile mediante un decodificatore $k \times 2^k$ le cui uscite sono inviate ad una matrice $n \times n$ di porte AND a due ingressi insieme agli n ingressi x_i; infine le uscite della matrice sono sommate in un OR ad n ingressi (o in una rete libera da fan-in equivalente) (Fig. 3.14).

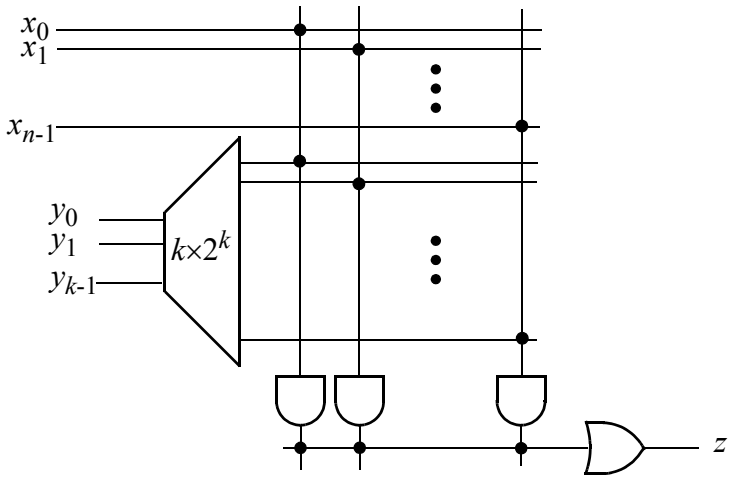


Fig. 3.14

Il multiplexer trova una interessante applicazione, oltre quella propria per la quale è

progettato, anche come **generatore universale** di funzioni combinatorie di k variabili, o meglio come dispositivo per la lettura di tavole di verità di funzioni. Infatti se confrontiamo l'espressione della sua uscita con la formulazione del teorema di espansione di una funzione in forma SP si vede che esse coincidono, se interpretiamo $y_0, y_1, ..., y_{k-1}$ come argomenti di una funzione f e le variabili $x_0, x_1, ..., x_{k-1}$ come le uscite di tale funzione corrispondenti alle n combinazioni degli argomenti, quando ogni x_i sia fissato ad un valore 0 o 1 in accordo alla tavola di verità della f.

Per esempio la funzione di tre variabili definita dalla seguente tavola di verità:

a	b	c	f
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	1
1	1	1	0

può essere realizzata con un multiplexer ad 8 ingressi-dati, fissati ai valori $x_0 = 0$, $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, $x_5 = 0$, $x_6 = 1$, $x_7 = 0$, e 3 ingressi di controllo ai quali sono applicati gli argomenti della funzione a, b, c della funzione, come in Fig. 3.15:

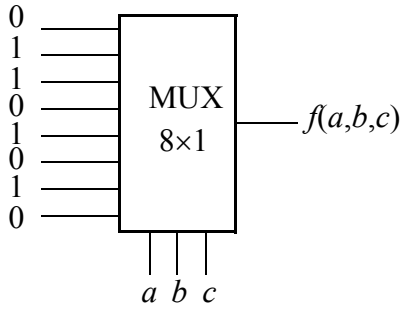


Fig. 3.15

Questa tecnica può essere estesa per generare funzioni di k variabili mediante un unico multiplexer con $i \le k$ ingressi di selezione, più logica booleana sparsa. Si divide idealmente la tavola di verità della funzione in 2^{k-i} parti di 2^i righe ciascuna e si osserva che per ogni gruppo di 2^{k-i} righe, una per ogni parte della tavola di verità, che si corrispondono distando una dall'altra di 2^i, le prime i variabili meno significative $x_0, ..., x_{i-1}$ hanno sempre valore costante, mentre le variabili da x_i a x_{k-1} assumono tutte le possibili 2^{k-i} combinazioni di valori; dunque per ognuno di quei gruppi di righe la funzione dipende solo dalle variabili che cambiano. Si

definiscono allora i funzioni g_0, ..., g_{i-1} di tali variabili, deducendone i valori dalla funzione $f(x_0, ..., x_{i-1}, x_i, ..., x_{k-1})$, come mostra la seguente tabella, e se ne costruiscono le rispettive reti; quindi le uscite delle reti sono collegate ordinatamente agli ingressi-dati del multiplexer e le variabili x_0, ..., x_{i-1} sono applicate agli ingressi di selezione (Fig. 3.15):

$x_{k-1} ... x_i$	g_0	g_1	...	g_{i-1}
0 ... 0	$f(0,0,...,0)$	$f(1,0,...,0)$		$f(i\text{-}1,0,...,0)$
...	...	...	...	...
1 ... 1	$f(0,1,...,1)$	$f(1,1,...,1)$		$f(i\text{-}1,1,...,1)$

Nota: la notazione $f(r,b,b,...,b)$, con $0 \le r \le i\text{-}1$ e $b \in \{0,1\}$, indica il valore della funzione quando le variabili x_0, ..., x_{i-1} codificano in binario l'intero r e le altre assumono una qualunque delle combinazioni da 0...0 a 1...1.

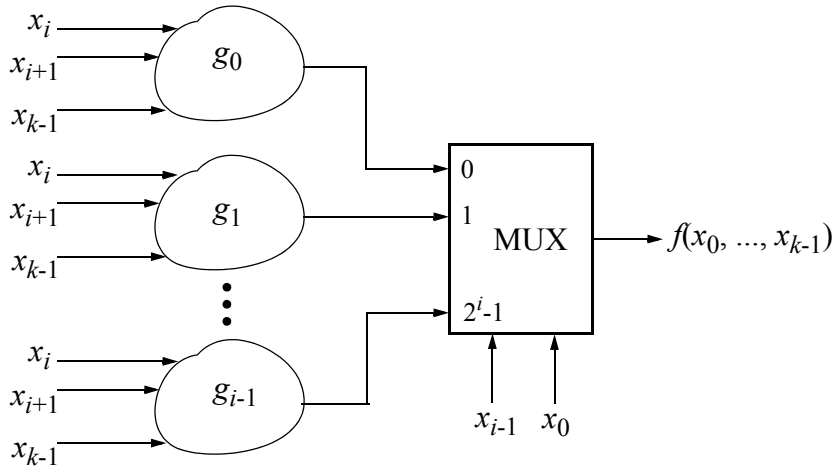


Fig. 3.16

In alternativa, si può leggere la tavola di verità della funzione mediante un albero di multiplexer identici, con i variabili di selezione, connessi nel seguente modo. Si immagina ancora la tavola di verità della funzione divisa in 2^{k-i} parti di 2^i righe ciascuna e si applicano i 2^i elementi di ciascuna parte agli ingressi di 2^{k-i} multiplexer controllati dalle variabili x_0, ..., x_{i-1}. Le uscite dei multiplexer sono applicate a gruppi di i agli ingressi di 2^{k-2i} ulteriori multiplexer controllati dalle variabili x_i, ..., x_{2i-1} e così via fino ad un unico multiplexer finale controllato dalle variabili x_{k-i}, ..., x_{k-1} (Fig. 3.17). Può accadere che non tutti gli ingressi del multiplexer finale siano utilizzati, ma soltanto 2^l, con $l < i$: in questo caso le variabili di controllo x_{k-i+l}, ..., x_{k-1} saranno fissate a 0.

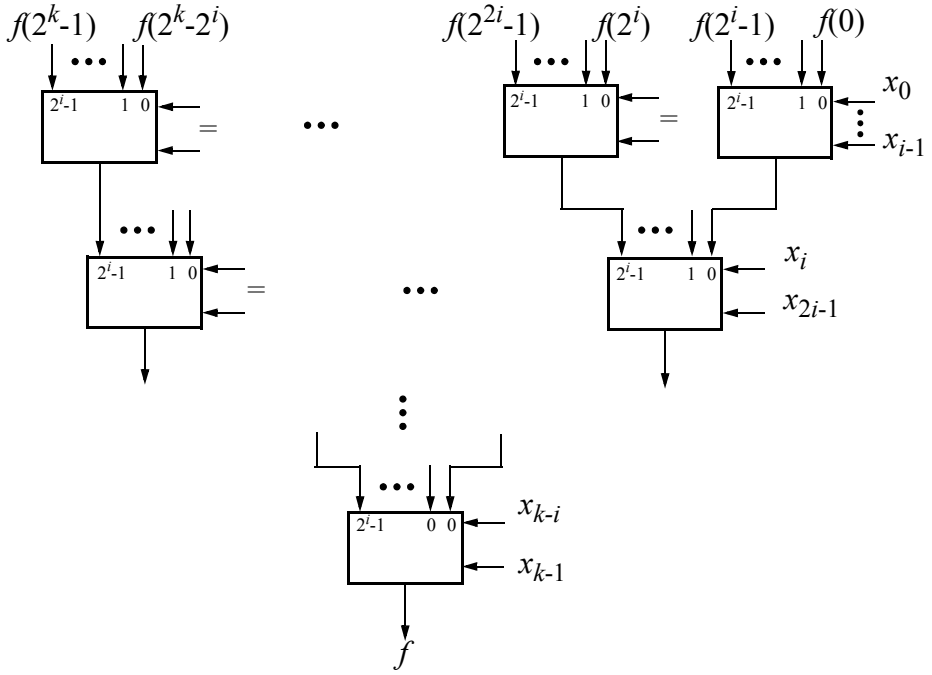


Fig. 3.17

3.5.2 - Demultiplexer

Un demultiplexer è una rete combinatoria che realizza la funzione opposta a quella del multiplexer, in quanto trasferisce un unico segnale di ingresso su una di n uscite $z_0, z_1, ..., z_{n-1}$ selezionata mediante $k = \lceil \log_2 n \rceil$ segnali di controllo. Ogni uscita sarà perciò governata da una funzione del tipo:

$$z_i = p_i \cdot x, \quad p_i = y^*_0 \cdot y^*_1 \cdot \ldots \cdot y^*_{k-1}, \quad y^*_j \in \{y_j, \bar{y}_j\}$$

nella quale è riconoscibile uno dei mintermini di k variabili moltiplicato per x; ciò conduce ad una immediata realizzazione con un decodificatore $k \times 2^k$ le cui uscite sono applicate ad una matrice $1 \times n$ di AND a due ingressi; le uscite della matrice sono le uscite del demultiplexer (Fig. 3.18).

Sia nei multiplexer che nei demultiplexer i segnali-dati in ingresso ed in uscita in generale sono di b bit, in quanto è frequente la necessità di smistare byte o parole: la struttura delle due reti non cambia, salvo che la porzione relativa alla matrice di AND deve essere replicata b volte, mentre rimane unica la parte di decodifica.

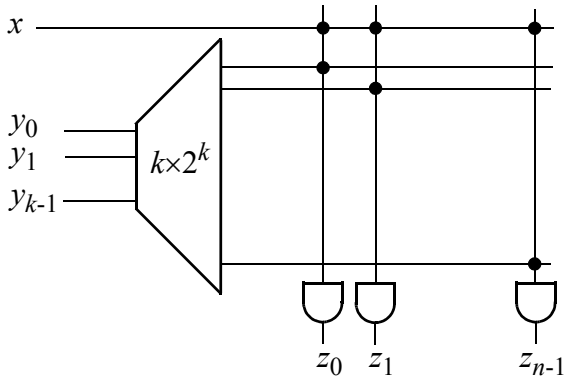


Fig. 3.18

Esistono infine reti in grado di effetture la selezione di segnali di *b* bit tra *n* in ingresso e di trasferirli su uno tra *m* segnali, pure di *b* bit, in uscita: esse sono perciò provviste di $k = \lceil \log_2 n \rceil$ segnali di controllo per la selezione di ingresso e di $h = \lceil \log_2 m \rceil$ segnali di controllo per la selezione di uscita, come illustrato in Fig. 3.19.

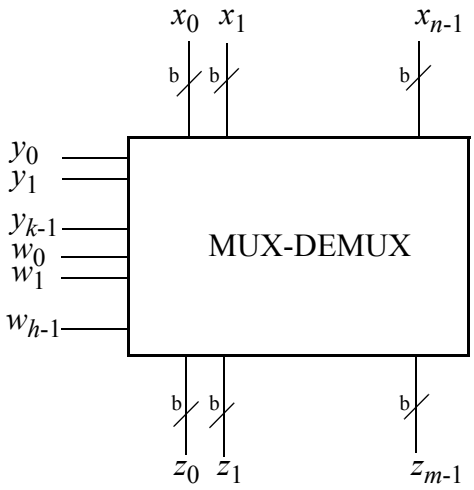


Fig. 3.19

3.6 - Generazione di funzioni combinatorie generiche

3.6.1 - ROM

Il concetto che abbiamo visto applicato nei multiplexer per realizzare funzioni combinatorie è basato in sostanza sulla generazione dei mintermini di *n* variabili, sul prodotto di ciascuno per 1 o per 0 secondo che la funzione abbia 1 o 0 in uscita per ogni data combinazione binaria e sulla somma logica di questi prodotti: la struttura che ne deriva è un caso particolare di una più generale comunemente conosciuta con il nome di **ROM** (Read Only

Memory) o **memoria di sola lettura**.

In effetti dal punto di vista concettuale il più semplice metodo di progetto di reti combinatorie ad n uscite per generare n funzioni di k variabili consiste nel produrre gli $m = 2^k$ mintermini di quelle variabili mediante un decodificatore e nel sommare funzione per funzione i mintermini corrispondenti alle righe di 1 nella tabella di verità di ciascuna funzione. La denominazione di memoria di sola lettura data ad una rete di questo tipo si giustifica con le seguenti considerazioni.

Logicamente una ROM è un dispositivo in grado di conservare indefinitamente l'informazione memorizzata al momento della fabbricazione e di renderla disponibile in uscita su richiesta. L'informazione è codificata sotto forma di m stringhe binarie di n bit ($n \geq 1$) che sono dette **parole** e sono riferite attraverso un **indirizzo** numerico variabile tra 0 ed m-1. L'indirizzo è codificato mediante $k = \lceil \log_2 m \rceil$ segnali binari da 1 bit in modo che per ogni combinazione di valori 0 ed 1 ad essi assegnati viene selezionata una delle m parole.

In realtà alle parole di una ROM non corrisponde alcuna realtà fisica, ma esse hanno solo un significato logico derivante dal fatto che viene stabilita una corrispondenza biunivoca tra ogni valore dell'indirizzo ed il valore dei bit della parola indirizzata: se interpretiamo ognuno di tali bit come l'uscita di una funzione i cui argomenti sono le variabili associate ai segnali indirizzo, una ROM può essere realizzata mediante una rete a più uscite come quella sopra descritta ed il cui schema a blocchi è indicato nella Fig. 3.20:

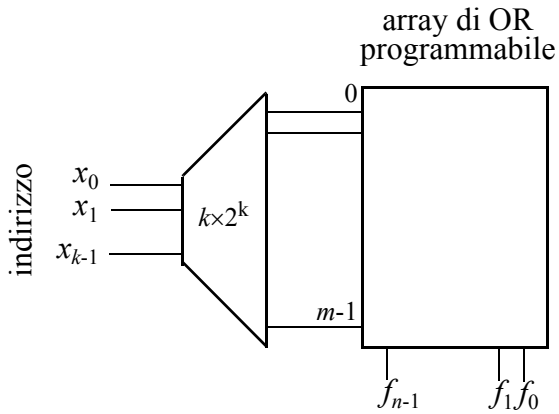


Fig. 3.20

In esso compare un array di OR detto **programmabil**e: con questo termine si intende una schiera lineare di n porte OR ad m ingressi, nella quale è possibile interrompere selettivamente determinati percorsi di ingresso, facendo bruciare con un impulso di corrente dei fusibili presenti su di essi, in modo da fare arrivare all'entrata della porta i-esima ($0 \leq i \leq n$-1) solo il sottoinsieme di mintermini richiesto per realizzare la funzione che governa il bit associato. Il processo di distruzione è permanente nelle ROM propriamente dette e viene compiuto in fabbrica secondo le indicazioni del cliente; esso rappresenta la fase di **scrittura** dell'informazione e in pratica consiste nel realizzare la rete combinatoria che genera n funzioni in forma canonica degli stessi argomenti: il contenuto di una ROM è dunque cablato in

hardware all'atto della costruzione del dispositivo.

Esistono anche ROM dette **riprogrammabili**, nelle quali la bruciatura dei collegamenti di ingresso alle porte OR non è permanente, ma essi possono essere ripristinati per mezzo di luce ultravioletta (**PROM**) o per via elettrica (**EPROM**): tale operazione determina la distruzione della configurazione circuitale (layout) precedente e quindi la perdita dell'informazione in essa cablata; riprogrammando la memoria si stabilisce una nuova configurazione di connessioni interne e quindi la memorizzazione di un nuovo contenuto informativo.

Per esempio sia richiesta una ROM di otto parole da quattro bit, configurate secondo la tabella seguente:

parola/bit	3	2	1	0
0	0	0	1	1
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	1	1	1	0
5	1	0	1	1
6	0	0	0	1
7	1	1	1	1

La struttura che si ottiene è costituita da un decodificatore 3×8 e da quattro porte OR le cui uscite sono le funzioni f_0, f_1, f_2, f_3 associate ai bit 0, 1, 2 e 3 di ogni parola; gli ingressi di ogni porta OR sono le uscite del decodificatore che nella numerazione da 0 a 7 corrispondono alla presenza di 1 nella colonna relativa della tabella (Fig. 3.21):

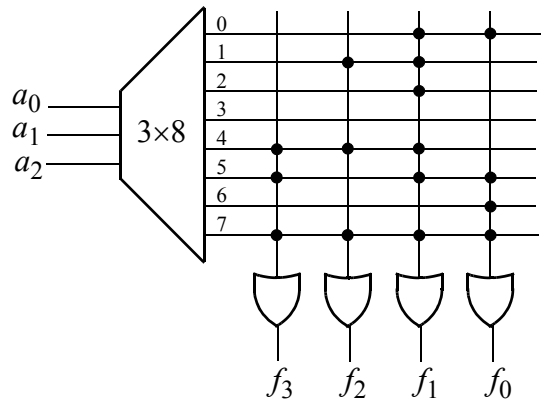


Fig. 3.21

In questo modo specificando ad esempio l'indirizzo 5, si ottiene in uscita la parola corrispondente, ovvero 1011.

Le ROM sono dispositivi molto versatili per realizzare in hardware funzioni matematiche, conversioni di codici, programmi di controllo di sistemi di elaborazione (microprogrammi),

generatori di caratteri, parti combinatorie di reti sequenziali. Tuttavia presentano alcuni inconvenienti. Prima di tutto l'ingombro, che cresce come 2^n, se n è il numero delle variabili di ingresso (indirizzo): aggiungere una variabile significa raddoppiare le dimensioni della ROM; questo dipende dal fatto che le ROM, generando funzioni in forma canonica, richiedono un decodificatore completo. Pertanto, come orientamento generale, per funzioni semplici è sempre molto più conveniente progettare reti minime specifiche. Anche la velocità non gioca a favore delle ROM, rispetto a reti progettate avendo come scopo l'ottimizzazione di questo parametro, pur essendo dispositivi più veloci di altri tipi di memorie, quali le RAM: valori tipici del tempo di risposta sono dell'ordine di qualche decina di ns (1 ns = 10^{-9} s).

3.6.2 - PLA

Con questa sigla (**P**rogrammable **L**ogic **A**rrays) si indicano componenti funzionalmente simili alle ROM, spesso accomunati a queste sotto l'unica sigla **PLD** (**P**rogrammable **L**ogic **D**evices). Tuttavia le funzioni generate dalle PLA non sono in forma canonica ma in forma minima, ossia per ciascuna funzione vengono prodotti solo i termini che costituiscono un insieme non ridondante di implicanti (o di implicati) primi.

Le PLA sono dispositivi molto utilizzati per la realizzazione di reti combinatorie qualsiasiin modo regolare e semplice, in particolare per realizzare la logica sparsa.

Si tratta di strutture a due livelli di logica, divise in due sezioni: una matrice o **piano** di AND, dove vengono generati i prodotti necessari per le funzioni, secondo i criteri di sintesi ottima delle reti a più uscite; una matrice o **piano** di OR per la generazione delle funzioni di uscita. Le uscite della matrice di AND prendono il nome di linee prodotto o **linee delle parole**, mentre gli ingressi sono dette **linee dei bit**. Gli ingressi delle porte OR sono linee delle parole e le uscite sono dette **linee di uscita**. Gli incroci delle linee delle parole con le linee dei bit o con le linee di uscita prendono il nome di **crosspoint** e in corrispondenza di ciascuno di essi esiste un ingresso ad una porta logica (Fig. 3.22). In pratica, dal punto di vista tecnologico è più conveniente realizzare i due piani con porte di un unico tipo e ciò è reso possibile grazie all'esistenza degli operatori logici universali e delle corrispondenti porte NAND e NOR.

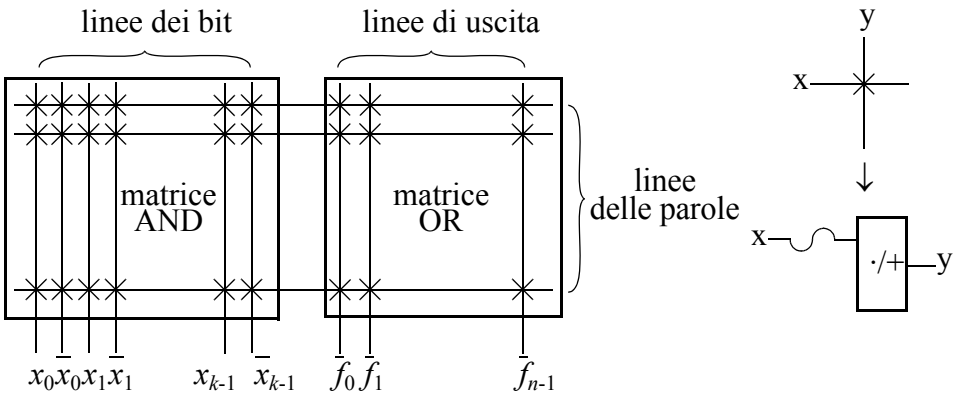


Fig. 3.22

Le variabili di ingresso sono presentate al piano AND insieme ai loro complementi attraverso circuti detti **phase splitter**, che costituiscono la forma più semplice di decodificatore e che sono provvisti di buffer per il pilotaggio delle porte AND (Fig. 3.23).

,

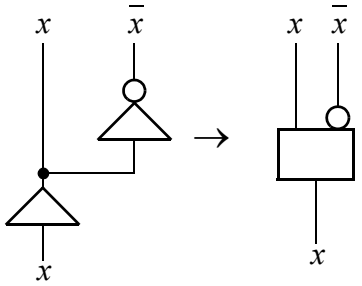


Fig. 3.23

Ovviamente non tutti i crosspoint devono essere attivi, nel senso che non deve esistere un ingresso ad una porta AND o OR per ogni coppia di linee (bit, parola) o (parola, uscita) che si incrociano, ma nel piano AND sono attivi solo quelli che servono a collegare elettricamente le linee delle parole con quelle dei bit richieste per formare i vari prodotti e nel piano OR quelli che devono collegare alle linee di uscita gli insiemi opportuni di linee delle parole. In corrispondenza dei crosspoint inutilizzati gli ingressi alle porte sono distrutti, con un sistema analogo a quello delle ROM, durante la fase di programmazione della PLA.

Esistono anche varianti più semplici delle PLA propriamente dette, nelle quali si cerca di ridurre l'ingombro ed il costo: si tratta delle **PAL** o **P**rogrammable **A**rray **L**ogic, nelle quali non esiste la possibilità di programmare il piano OR, in quanto i crosspoint presenti in esso sono collegamenti fissi e la programmabilità è limitata al solo piano AND (Fig. 3.24). I tempi di propagazione tipici delle PAL variano da circa 15 ns a circa 25 ns. Una versione ancora più semplice è costituita dalle **HAL** (**H**ard **A**rray **L**ogic) nelle quali non esistono crosspoint fusibili, ma le connessioni sono realizzate in fabbrica con un processo di mascheratura per deposizione metallica, secondo le specifiche fornite dal cliente.

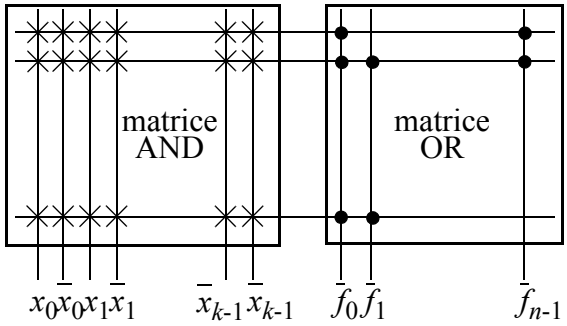


Fig. 3.24

Un'altra famiglia di componenti modulari studiati per la realizzazione di circuiti logici è costituita dai Gate Arrays o più in generale dai **P**rogrammable **G**ate **A**rrays (**PGA**); si tratta di

dispositivi formati da un gran numero di celle elementari identiche che possono essere variamente interconnesse a formare una rete più complessa. Le interconnessioni sono realizzate dal costruttore sulla base di maschere fornite dall'utente deponendo nella fase finale di integrazione uno strato di metallo (**antifusing**) fra le celle per costruire porte logiche standard, tra le porte per formare circuiti più o meno complessi e tra i circuiti ed i piedini di ingresso/uscita per le connessioni con l'esterno.

Vediamo ora un esempio di progetto logico di una PLA e per questo consideriamo le stesse funzioni definite dalle tabelle relative alla ROM di Fig. 3.21, ossia $f_0 = \sum_3(0,5,6,7)$, $f_1 = \sum_3(0,1,2,4,5,7)$, $f_2 = \sum_3(1,4,7)$, $f_3 = \sum_3(4,5,7)$. Le rispettive forme minime si possono ricavare con il metodo delle mappe di Karnaugh e sono le seguenti:

$$f_0 = \bar{a}\bar{b}\bar{c} + ac + ab$$

$$f_1 = \bar{b} + \bar{a}\bar{c} + ac$$

$$f_2 = a\bar{b}\bar{c} + \bar{a}\bar{b}c + abc$$

$$f_3 = a\bar{b} + ac$$

Pertanto il piano AND della PLA deve generare gli implicanti primi $\bar{a}\bar{b}\bar{c}$, $a\bar{b}\bar{c}$, $\bar{a}\bar{b}c$, abc, ab, $a\bar{b}$, ac, $\bar{a}\bar{c}$, $\bar{b}$, mentre il piano OR è costituito da quattro porte OR:

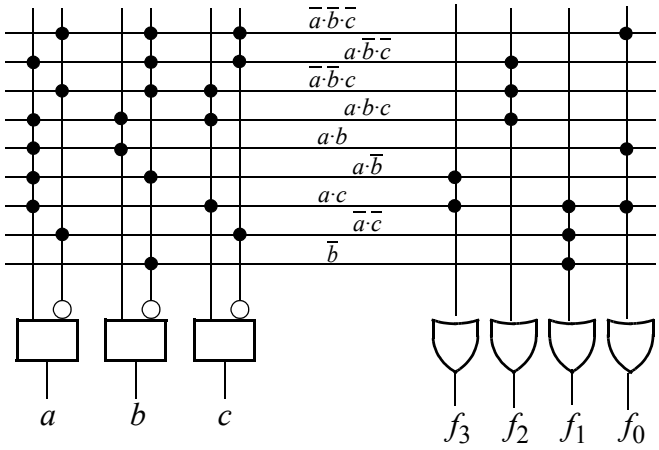


Fig. 3.25

La minimizzazione logica di una PLA, sia al fine di ridurre l'area di silicio necessaria per integrare una data struttura o di distribuirla sul minor numero possibile di chip, è sicuramente un'operazione vantaggiosa, ma presenta una complessità molto elevata (si tratta infatti di un problema NP-completo) a causa del grande numero di ingressi e di uscite di solito coinvolti e comporterebbe tempi di elaborazione proibitivi per l'esecuzione dei programmi di sintesi ottima basata sui metodi a suo tempo esaminati. Per questi motivi, in pratica si accettano soluzioni sub-ottime, di tipo euristico, nelle quali viene tollerata una certa ridondanza interna alle matrici AND e OR.

3.7 - Circuiti aritmetici

3.7.1 - Addizionatori

3.7.1.1 - Richiami di aritmetica in campo finito

Sia dato un campo numerico di capacità β^n, dove β è la base del sistema di rappresentazione ed n il numero di cifre; tutti i risultati di operazioni aritmetiche per essere significativi devono poter essere rappresentati all'interno di quel campo. D'altra parte per qualunque numero naturale X, nel campo è rappresentabile, come stringa di n cifre, il numero $|X|_{\beta^n}$ (si legga X *modulo* β^n), resto della divisione intera di X per β^n, ossia $|X|_{\beta^n} = X - \lfloor X/\beta^n \rfloor \times \beta^n$. Pertanto se è $X < \beta^n$, la sequenza di cifre contenuta nel campo è effettivamente la rappresentazione di X, altrimenti è la rappresentazione di un numero minore di X; in questo caso il numero X non può essere rappresentato in maniera corretta e se è il risultato di un'operazione aritmetica, si dice che essa ha determinato un **traboccamento** (overflow).

Siano $X \equiv x_{n-1}x_{n-2}...x_0$ e $Y \equiv y_{n-1}y_{n-2}...y_0$ le rappresentazioni in base β di due numeri naturali; il valore della cifra i-esima della somma e quello del riporto dalla posizione i-esima alla (i+1)-esima sono date dalle seguenti relazioni, valide per ogni i, $0 \le i \le n-1$:

$$s_i = \left| x_i + y_i + c_i \right|_\beta$$

$$c_{i+1} = \left\lfloor \frac{x_i + y_i + c_i}{\beta} \right\rfloor$$

Poichè valgono le disuguaglianze

$$0 \le X, Y \le \beta^n - 1$$

la somma $S = X+Y$ soddisfa le relazioni

$$0 \le S \le 2(\beta^n - 1)$$

e pertanto la sua rappresentazione può richiedere n+1 cifre; infatti il massimo numero rappresentabile con n cifre è β^n-1, quello rappresentabile con n+1 cifre è β^{n+1}-1 e valgono le disuguaglianze $\beta^n - 1 < 2(\beta^n - 1) < \beta^{n+1} - 1$. Se questa è la situazione e non è possibile espandere il campo, nell'addizione si verifica un traboccamento ed il numero S non può essere correttamente rappresentato: il campo contiene il numero $|S|_{\beta^n}$, mentre il riporto c_n è uguale ad 1.

Consideriamo ora il problema dell'addizione di numeri relativi, limitandoci al caso della base 2. Delle due rappresentazioni più utilizzate per i numeri relativi, quella in modulo e

segno comporta artificiose complicazioni in quanto l'addizione algebrica si trasforma in un'addizione aritmetica delle grandezze dei numeri se essi sono concordi, in una sottrazione se sono discordi; per questo facciamo riferimento alla rappresentazione in complemento a 2 in un campo di lunghezza pari a quella degli addendi e dell'addizionatore, una metà del quale è utilizzato per rappresentare i numeri positivi e lo 0, l'altra metà per quelli negativi. La notazione in complemento consente di ricondurre ogni operazione di tipo additivo tra numeri relativi ad un'addizione tra i numeri naturali che ne costituiscono le rappresentazioni, dal momento che la sottrazione tra due numeri equivale a sommare all'uno il complemento dell'altro. Per questa ragione non considereremo la sottrazione come operazione a sè. Tuttavia, mentre nel caso dei numeri naturali tutto il campo è disponibile per rappresentare il valore dei numeri, che pertanto coprono l'intervallo da 0 a β^n-1 estremi compresi, in questo caso il campo a disposizione per il valore è metà.

Siano X ed Y due numeri relativi, rappresentati in complemento a 2 su n bit; affinchè l'addizione possa essere effettuata correttamente, è necessario che il campo abbia ampiezza sufficiente per rappresentare il risultato; in caso contrario si ha traboccamento dal semicampo dei numeri positivi a quello dei numeri negativi o viceversa. La rilevazione del traboccamento implica il confronto fra i bit più significativi degli addendi x_{n-1} e y_{n-1} e quello della somma s_{n-1}: se gli addendi sono dello stesso segno ($x_{n-1}=y_{n-1}$), concordanza di valore di s_{n-1} con x_{n-1} (o con y_{n-1}) significa risultato corretto, discordanza significa che traboccamento; se gli addendi sono invece di segno discorde, non c'è bisogno di ulteriori controlli, perchè il risultato è sicuramente sempre corretto in valore e segno (non c'è mai traboccamento). Una possibile rete per rilevare il traboccamento è la seguente:

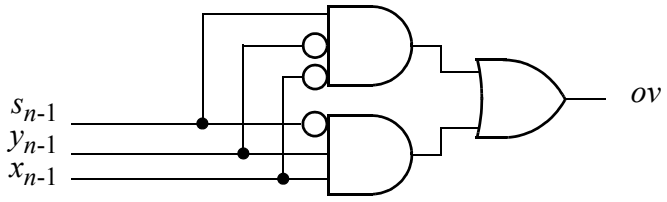


Fig. 3.26

Il riconoscimento del traboccamento però può essere effettuato anche in base alle considerazioni seguenti.

I - Se X ed Y sono entrambi non negativi, la somma $S=X+Y$ è pure un numero non negativo e poiché $x_{n-1}=y_{n-1}=0$, anche s_{n-1} deve essere 0; d'altra parte le seguenti relazioni:

$$s_{n-1} = \left|x_{n-1}+y_{n-1}+c_{n-1}\right|_2 = \left|c_{n-1}\right|_2 =$$
$$c_n = \left\lfloor \frac{x_{n-1}+y_{n-1}+c_{n-1}}{2} \right\rfloor = \left\lfloor \frac{c_{n-1}}{2} \right\rfloor ;$$

mostrano che se c_{n-1} è 1, il bit più significativo della somma è pure 1 e c'è traboccamento; se

c_{n-1} è 0, la somma è correttamente rappresentata. Il riporto c_n è in ogni caso uguale a 0.

II - Se X ed Y sono negativi, anche la loro somma $S = X+Y$ è negativa; dal momento che i bit più significativi delle rappresentazioni di X e di Y sono 1, ovvero $x'_{n-1} = y'_{n-1} = 1$, per avere il risultato corretto deve essere 0 anche il bit più significativo della rappresentazione della somma, s'_{n-1}; d'altra parte le uguagliamze:

$$s'_{n-1} = \left| x'_{n-1} + y'_{n-1} + c_{n-1} \right|_2 = \left| 2 + c_{n-1} \right|_2 = c_{n-1}$$
$$c_n = \left\lfloor \frac{x'_{n-1} + y'_{n-1} + c_{n-1}}{2} \right\rfloor = \left\lfloor \frac{2 + c_{n-1}}{2} \right\rfloor = 1$$

mostrano che se c_{n-1} è uguale a 0, s'_{n-1} è pure 0 e la somma S è rappresentata erroneamente come numero positivo e c'è traboccamento; viceversa se c_{n-1} è 1, anche s'_{n-1} è 1 ed S è rappresentata da S' correttamente. Il riporto c_n è in ogni caso 1.

III - I due numeri siano discordi con $X \geq 0$, $Y < 0$, situazione alla quale possiamo sempre riportarci senza perdere in generalità; possono darsi due casi:

a) $X \geq |Y|$; allora $0 \leq S < X$ ed è sempre rappresentata correttamente, e si ha $s'_{n-1} = 0$. Inoltre $x_{n-1} = 0$ e $y'_{n-1} = 1$ e dalla relazione:

$$s'_{n-1} = \left| x_{n-1} + y'_{n-1} + c_{n-1} \right|_2 = \left| 1 + c_{n-1} \right|_2 = 0$$

si deduce che deve essere $c_{n-1} = 1$ e quindi $c_n = \lfloor (1+c_{n-1})/2 \rfloor = 1$.

b) $X-|Y| < 0$; allora $Y < S < 0$, ovvero $s'_{n-1} = 1$. Dalla relazione

$$s'_{n-1} = \left| x_{n-1} + y'_{n-1} + c_{n-1} \right|_2 = \left| 1 + c_{n-1} \right|_2 = 1$$

si ricava $c_{n-1} = 0$ e conseguentemente $c_n = \lfloor (1+c_{n-1})/2 \rfloor = 0$.

La tabella seguente riassume le situazioni descritte:

x_{n-1}	y_{n-1}	s_{n-1}	c_{n-1}	c_n	esito
0	0	0	0	0	somma positiva (corretta)
0	0	1	1	0	traboccamento
0	1	1	0	0	somma negativa (corretta)
0	1	0	1	1	somma positiva (corretta)
1	0	1	0	0	somma negativa (corretta)
1	0	0	1	1	somma positiva (corretta)
1	1	0	0	1	traboccamento
1	1	1	1	1	somma negativa (corretta)

Essa mostra che la rilevazione del traboccamento può essere condotta sulla base del confronto tra il bit di riporto verso la posizione più significativa del risultato c_{n-1} e quello di riporto nella posizione (n+1)-esima, c_n: infatti solo in presenza di overflow tali bit sono diversi, per cui una singola porta XOR avente come ingressi tali riporti è sufficiente per generare il bit di overflow.

3.7.1.2 - Addizionatore ripple-carry

In binario, dovendo effettuare l'addizione di due numeri naturali $X = x_{n-1}x_{n-2}\ldots x_1x_0$ e $Y = y_{n-1}y_{n-2}\ldots y_1y_0$, in linea di principio si può pensare ad una funzione combinatoria a due livelli di logica con n+1 uscite la quale produca in funzione dei 2n ingressi attuali n+1 variabili binarie che rappresentano gli n+1 bit del numero S, risultato dell'addizione:

$$S = c_n s_{n-1} s_{n-2} \ldots s_1 s_0$$

Una tale funzione è in grado di produrre il risultato in due unità di tempo, una per un gruppo di porte AND che agiscono non appena si sono formati i dati di ingresso, una per un gruppo di porte OR che operano quando le porte AND hanno prodotto le loro uscite.

In pratica il numero di porte ed il loro fan-in diventano proibitivi non appena n è superiore a 2÷3, rendendo impossibile realizzare la funzione in questo modo ed è obbligatorio trovare altri metodi che usano logica a più di due livelli, anche se a prezzo di un maggiore tempo di calcolo. Infatti, mentre l'uscita s_0 della funzione S dipende solo dalla coppia (x_0, y_0), l'uscita s_1 dipende dalle due coppie (x_1, y_1) e (x_0, y_0) e così via; in generale l'uscita s_i dipende da 2(i+1) variabili, $0 \le i \le n$-1 e le uscite s_{n-1} e c_n dipendono da tutte le 2n variabili. La complessità delle sottoreti che calcolano le singole cifre della somma aumenta quindi in maniera inammissibile.

Al fine di individuare una soluzione praticabile, cominciamo a considerare la somma di due numeri binari di 1 bit, x ed y; dal momento che quando i due numeri valgono 1 il risultato dell'addizione è 10, ossia un bit s di somma pari a 0 e un bit c di riporto pari ad 1, occorre definire due funzioni logiche $s = f(x, y)$, $c = g(x, y)$, attraverso le tavole di verità:

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Come si può vedere, s vale 1 quando x ed y sono diversi e vale 0 quando sono uguali, mentre c vale sempre 0, tranne nel caso $x = y = 1$, per cui si ha:

$$s = x\bar{y} + \bar{x}y = x \oplus y$$
$$c = xy$$

La rete che realizza queste funzioni è detta **semiaddizionatore** o **half adder** (HA) ed è mostrata nella Fig. 3.27:

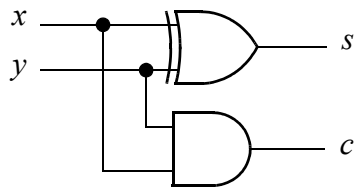


Fig. 3.27

Il termine semiaddizionatore deriva dal fatto che questa rete non è in grado di produrre correttamente la somma quando i due bit non sono una coppia isolata, ma ciascuno fa parte di una stringa di n bit rappresentante un numero binario; infatti il bit di somma e quello di riporto in posizione i-esima in tal caso dipendono non solo dalla coppia di bit addendi x_i e y_i, ma anche dal bit di riporto c_i proveniente dalla somma della coppia (x_{i-1}, y_{i-1}). Occorre allora ridefinire le tabelle di verità della somma e del riporto, facendovi figurare anche c_i:

x_i	y_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Il loro esame mostra ancora che s_i è 1 solo se il numero di 1 nella terna (x_i, y_i, c_i) è dispari, quindi è ancora espresso dall'OR esclusivo delle tre variabili, mentre il riporto verso la posizione $(i+1)$-esima è 1 solo se almeno una coppia delle tre variabili è 1, per cui si ha:

$$s_i = x_i \oplus y_i \oplus c_i = (x_i \oplus y_i) \oplus c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

D'altra parte con l'algebra si può scrivere:

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + x_i c_i \bar{y}_i + y_i c_i \bar{x}_i = (x_i \bar{y}_i + y_i \bar{x}_i) c_i + x_i y_i = (x_i \oplus y_i) c_i + x_i y_i$$

Ne risulta la rete di Fig. 3.28, nella quale sono riconoscibili due semiaddizionatori HA ed una porta OR e che è detta **addizionatore completo** o **full adder** (FA):

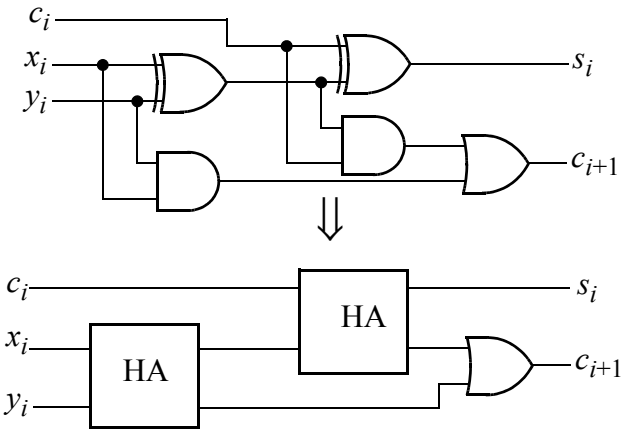


Fig. 3.28

Torniamo ora al problema iniziale di sommare due numeri binari di n bit; ciò può essere fatto mediante n-1 addizionatori completi ed un semiaddizionatore per la coppia di bit meno significativi, collegati in modo che l'uscita di riporto dello stadio i-esimo sia collegata all'ingresso di riporto dello stadio (i+1)-esimo e applicando le coppie di bit (x_i, y_i), i = 0, …, n-1, agli ingressi corrispondenti (Fig. 3.29).

In questo schema le singole cifre della somma non sono generate contemporaneamente, con lo stesso ritardo dal momento di applicazione degli operandi in ingresso, ma ognuna con un ritardo variabile che dipende strettamente dalle coppie di cifre o gruppi di coppie di cifre precedenti: prima che sia noto il valore del riporto che concorre a formare il bit di somma dello stadio i-esimo occorre, nel caso peggiore, che tutti gli stadi da 0 a i-1 abbiano prodotto il loro risultato; quando la situazione del caso peggiore si verifica per la cifra più significativa, l'addizione è effettuata con un ritardo massimo T_A pari a:

$$T_A = (n\text{-}1)\Delta_c + \max(\Delta_s, \Delta_c)$$

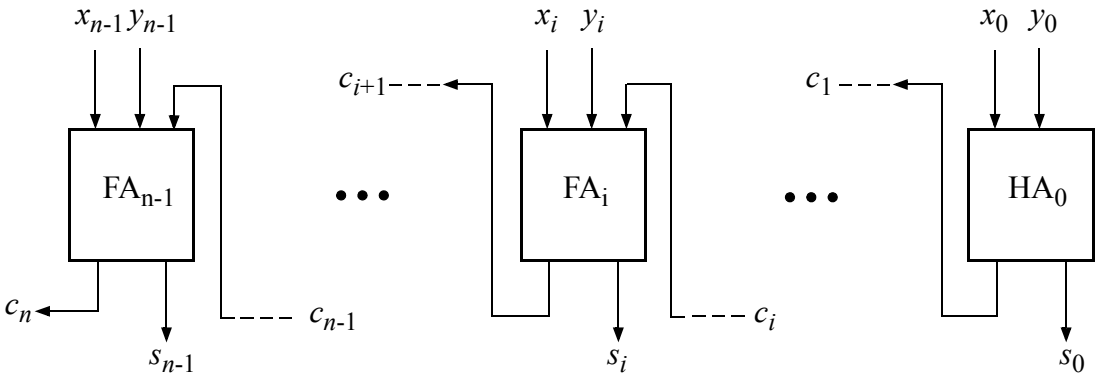


Fig. 3.29

essendo Δ_c il tempo di propagazione dall'ingresso all'uscita del riporto c in un full adder e Δ_s il tempo di propagazione dall'ingresso all'uscita della somma s. All'estremo opposto si verifica la situazione per cui i dati hanno valori tali che nessun riporto 1 viene generato e quindi il tempo di risposta dell'addizionatore ha il valore minimo Δ_s.

Questa variabilità di comportamento dell'addizionatore per quanto riguarda il tempo di risposta e la sua forte dipendenza dal valore e dalla lunghezza degli addendi è dovuta alla natura essenzialmente sequenziale del riporto, il cui fronte si propaga serialmente da uno stadio all'altro: da questa caratteristica deriva il nome di **addizionatore ripple carry** attribuito a questa rete.

Per sottrarre l'addizione all'influenza negativa del riporto e aumentarne la velocità sono state studiate varie tecniche, le più comuni delle quali sono le seguenti:

1) Si può prevedere un circuito che evita la propagazione del riporto attraverso i vari stadi, calcolandone in anticipo il valore per tutti in modo che, almeno in teoria, l'addizione possa avvenire simultaneamente su tutti i bit (tecnica di **carry look-ahead**).
2) Quando si deve eseguire un flusso continuo di addizioni, si può ripartire in uguale misura su ogni operazione il ritardo dovuto al riporto rimandandone la propagazione a dopo che è entrato nell'addizionatore l'ultimo operando (tecniche di **carry save** e di **pipelining**).

3.7.1.3 - Addizionatore carry look ahead (CLA)

Riprendiamo l'espressione del riporto generato dallo stadio i-esimo dell'addizionatore:

$$c_{i+1} = (x_i \oplus y_i)c_i + x_i y_i$$

e osserviamo che un valore 1 di esso può essere dovuto a due fatti, mutuamente esclusivi:

a) il riporto si genera localmente allo stadio i-esimo perchè i due bit x_i e y_i sono entrambi 1; questo è espresso dal termine prodotto $x_i y_i$ che viene detto **termine di generazione**; notare che quando ciò accade l'altro termine $(x_i \oplus y_i)c_i$ è nullo;
b) il riporto si genera nello stadio precedente i-1 e lo stadio i-esimo semplicemente lo propaga al successivo, essendo x_i o y_i (ma non entrambi) 1: di ciò è responsabile il termine $(x_i \oplus y_i)$ che prende perciò il nome di **termine di propagazione**; notare anche che quando il termine di propagazione vale 1, è nullo il termine di generazione, per cui essi sono mutuamente esclusivi nel determinare c_{i+1}.

Detti g e p i due termini sopra introdotti, possiamo scrivere il riporto c_{i+1} nella forma:

$$c_{i+1} = g_i + p_i c_i$$

e iterando questa equazione per $i = 0, 1, \ldots, n$-1, otteniamo:

$$c_1 = g_0$$
$$c_2 = g_1 + p_1 c_1$$

$$c_3 = g_2 + p_2 c_2$$
$$\vdots$$
$$c_{n-1} = g_{n-2} + p_{n-2} c_{n-2}$$
$$c_n = g_{n-1} + p_{n-1} c_{n-1}$$

Infine sostituendo ricorsivamente nell'equazione i-esima l'espressione di c_{i-1} data dall'equazione precedente, si arriva alle seguenti equazioni:

$$c_1 = g_0$$
$$c_2 = g_1 + p_1 g_0$$
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0$$
$$\vdots$$
$$c_i = g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3} + \ldots + p_{i-1} p_{i-2} \ldots p_2 g_1 + p_{i-1} p_{i-2} \ldots p_2 p_1 g_0$$
$$\vdots$$
$$c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} + \ldots + p_{n-1} p_{n-2} \ldots p_2 g_1 + p_{n-1} p_{n-2} \ldots p_2 p_1 g_0$$

le quali mostrano che il riporto c_i in uscita allo stadio (i-1)-esimo è dovuto al termine di generazione di quello stadio, oppure al termine di propagazione di quello stadio e al termine di generazione dello stadio (i-2)-esimo, oppure al termine di propagazione di quello stadio, al termine di propagazione dello stadio (i-2)-esimo e al termine di generazione dello stadio (i-3)-esimo e così via.

I termini di generazione e di propagazione sono facilmente ottenibili come uscite del full adder, modificandone la struttura come indicato nella Fig. 3.30:

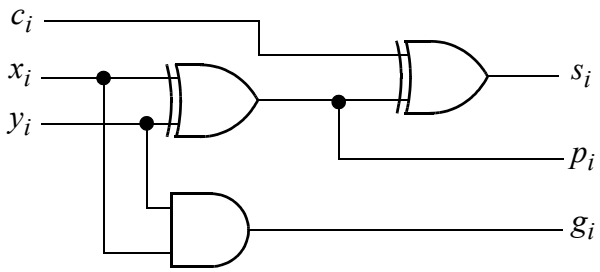


Fig. 3.30

Per ciascuna equazione possiamo allora costruire una rete combinatoria a due livelli di logica in forma SP che riceve in ingresso il termine di generazione e quello di propagazione prodotti nello stadio dell'addizionatore ad essa relativo e genera il riporto per lo stadio successivo. L'insieme di tali reti costituisce un modulo detto **carry look-ahead generator** (**CLAG**); poichè ogni riporto è generato in base alla conoscenza simultanea di tutti i termini di generazione e di propagazione degli stadi dal primo a quello corrente, e in definitiva di tutte le coppie di bit dalla meno significativa a quella corrente, i vari stadi dell'addizionatore con carry

look-ahead generator possono operare effettivamente in parallelo, producendo il risultato con un ritardo costante pari a $\Delta_s + 2\Delta_g$, essendo Δ_g il ritardo di una porta.

La Fig. 3.31 mostra un addizionatore a quattro bit con carry look-ahead generator.

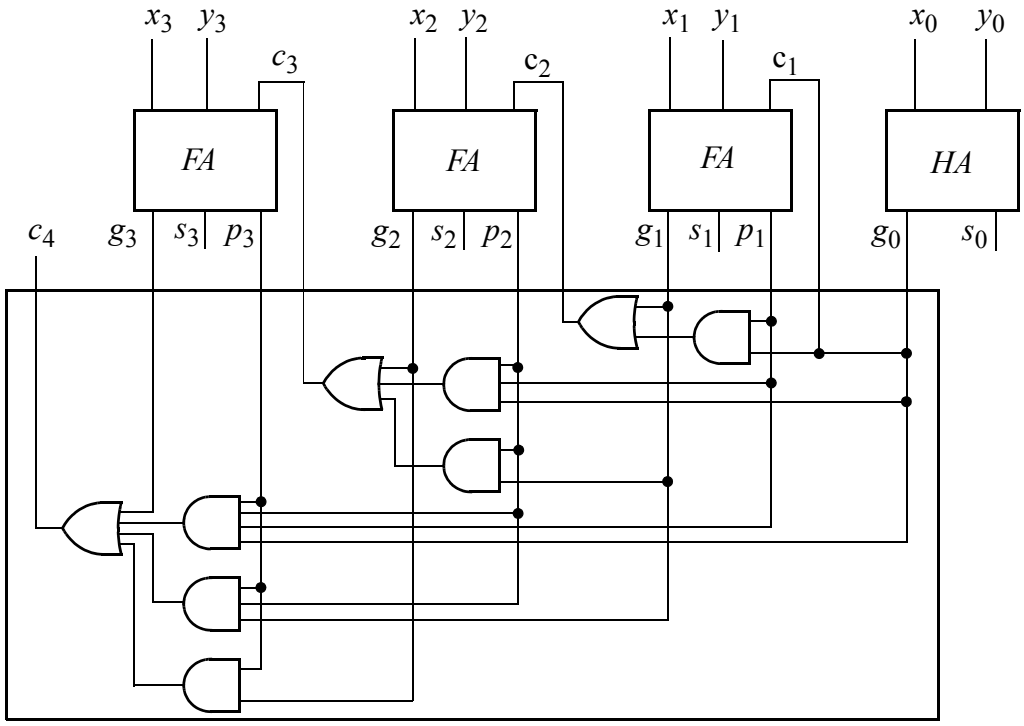


Fig. 3.31

Purtroppo i benefici dell'uso del carry look-ahead generator sono rapidamente vanificati al crescere di n perchè, come si vede già dall'esempio, il numero di porte e soprattutto il numero dei segnali in ingresso ad esse cresce a livelli inaccettabili: questa tecnica può essere applicata fino a valori di n non superiori a 5÷6.

Per addizionatori di lunghezza maggiore è necessario trovare un compromesso tra complessità realizzativa e velocità di risposta.

Una soluzione consiste nel dividere gli operandi in k gruppi di q bit, in modo che sia $n = k \times q$ e $q \leq 6$, applicando la tecnica di carry look-ahead all'interno di ogni gruppo, mentre tra un gruppo e l'altro viene applicata quella di ripple carry. In alternativa si può propagare serialmente il riporto tra gli stadi all'interno di ogni gruppo, riservando l'uso del carry look-ahead alla propagazione da un gruppo all'altro. Per questa soluzione occorre naturalmente dimensionare i gruppi in modo che sia $k \leq 6$, aumentando corrispondentemente il valore di q.

Una terza soluzione consiste nel propagare i riporti con tecnica carry look-ahead sia all'interno di ogni gruppo che tra i gruppi. In questa soluzione occorre disporre di almeno due livelli di carry look-ahead generator: al primo livello si trovano generatori dei riporti anticipati che lavorano sui singoli gruppi; al secondo livello quelli che lavorano tra i gruppi, ricevendo termini di generazione e di propagazione dai generatori del primo livello. La Fig. 3.32 mostra un addizionatore per numeri di 16 bit realizzato secondo la tecnica descritta e avendo scelto per

q un valore pari a 4.

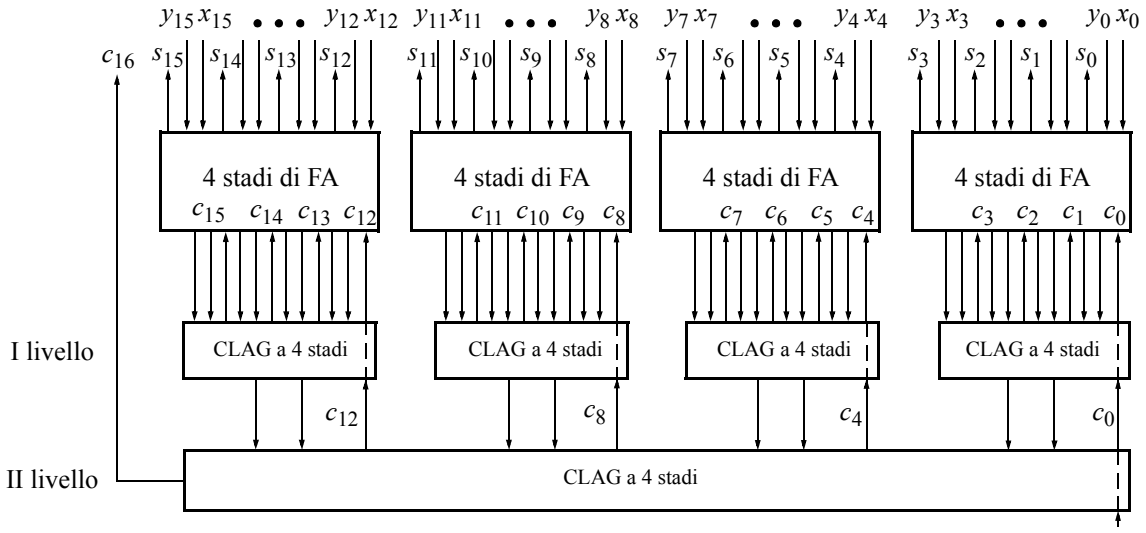


Fig. 3.32

Nel caso in cui la ripartizione dei bit degli operandi in gruppi produca un valore $k > 6$, i generatori del secondo livello devono essere ulteriormante divisi in più livelli, ottenendo complessivamente una struttura ad albero nella quale, per ragioni di modularità e di regolarità, si tende ad utilizzare in tutti i nodi generatori di carry look-ahead della stessa lunghezza q.

Con questo schema, poichè la profondità dell'albero è $\lceil \log_q n \rceil$, T_A è espresso dalla relazione:

$$T = (4 \cdot \lceil \log_q n \rceil + 2) \cdot \Delta_g$$

nella quale il fattore moltiplicativo 4 è dovuto al fatto che per ogni livello devono essere generati in parallelo (vedi Fig. 3.30) un termine di propagazione ($2\Delta_g$) ed uno di generazione (Δ_g) e quindi il riporto ($2\Delta_g$); la base del logaritmo si spiega con il fatto che l'albero è q-ario; infine il termine aggiuntivo 2 è dovuto al ritardo per la generazione del bit di somma. Da questa relazione risulta una complessità $O(\log_q n)$ che, pur essendo superiore a $O(1)$, quale avrebbe un addizionatore con carry look-ahead ad un unico livello e stadi a due livelli di logica, è decisamente inferiore alla complessità $O(n)$ del semplice addizionatore ripple carry.

Se si assume, come nell'esempio e come avviene spesso in pratica, che ogni generatore di carry look-ahead abbia quattro stadi, la relazione precedente dà per il tempo di risposta T di un CLA in funzione della lunghezza n degli addendi un andamento come riportato dalla seguente tabella:

n	4	16	64	256
T_A	$6\Delta_g$	$10\Delta_g$	$14\Delta_g$	$18\Delta_g$

3.7.1.4 - Addizionatore carry save

Un'altra tecnica per l'addizione, di pratica utilità soprattutto in presenza di addizioni ripetute, è quella detta di **carry save** (CSA) o di **acquisizione ritardata dei riporti**. L'idea di base consiste nel considerare l'addizione come una procedura ricorsiva nella quale al passo i-esimo si formano due vettori di n bit S^i e C^i, denominati **vettore delle somme parziali** e **vettore dei riporti parziali**, a partire dal vettore delle somme parziali S^{i-1} e dal vettore dei riporti parziali C^{i-1}, traslato a sinistra di una posizione rispetto al passo precedente, per tenere conto del corretto allineamento, e proseguendo fino ad avere un vettore dei riporti parziali nullo, oppure finchè un bit di valore 1 nella posizione più significativa del vettore dei riporti indica la presenza di traboccamento. Le componenti di tali vettori sono date dalle relazioni:

$$S_j^i = S_j^{i-1} \oplus C_{j-1}^{i-1}$$
$$C_j^i = S_j^{i-1} \cdot C_{j-1}^{i-1}$$

per $i = 2, 3, \ldots, j = 0, ..., n\text{-}1$ e $C_{-1}^i = 0$, mentre al primo passo si ha:

$$S_j^1 = x_j \oplus y_j$$
$$C_j^1 = x_j \cdot y_j$$

Per esempio siano $X = 0101101$ e $Y = 0110101$; avremo:

0101101	
0110101	
0011000	S^1
0100101	C^1
1001010	$2C^1$
1010010	S^2
0001000	C^2
0010000	$2C^2$
1000010	S^3
0010000	C^3
0100000	$2C^3$
1100010	S^4
0000000	$C^4 = 0$: fine della procedura

da cui il risultato $S = 1100010$.

Il procedimento potrebbe essere eseguito, in linea di principio, con una rete logica che

utilizza $n(n\text{-}1)/2$ semiaddizionatori disposti su n file, nell'i-esima delle quali vengono addizionati il vettore delle somme e quello dei riporti traslato, ottenuti come uscita della fila (i-1)-esima (Fig. 3.33). I semiaddizionatori della fila (n-1)-esima producono in uscita i bit da S_0 ad S_{n-1} ed il bit C_n del risultato.

In pratica però, in dipendenza dei valori attuali degli addendi, il vettore dei riporti parziali può annullarsi prima del passo finale, determinando la fine del calcolo, mentre i vettori parziali devono continuare ad essere trasferiti attraverso le file rimanenti dell'addizionatore, con inutile spreco di tempo; inoltre la complessità strutturale è molto più elevata rispetto a quella dell'addizionatore ripple carry e, nel caso peggiore, il tempo di risposta ha complessità O(n); per queste ragioni la rete di Fig. 3.33 ha solo valore di principio.

La complessità della struttura può essere riportata da O(n^2) a O(n) utilizzando una sola fila di n semiaddizionatori e due dispositivi (registri) capaci di memorizzare ciascuno n bit. Inizialmente essi memorizzano gli addendi X ed Y e nei vari passi del calcolo i vettori S^i e C^i; le loro n uscite sono connesse ordinatamente agli ingressi dei semiaddizionatori come mostra la Fig. 3.34.

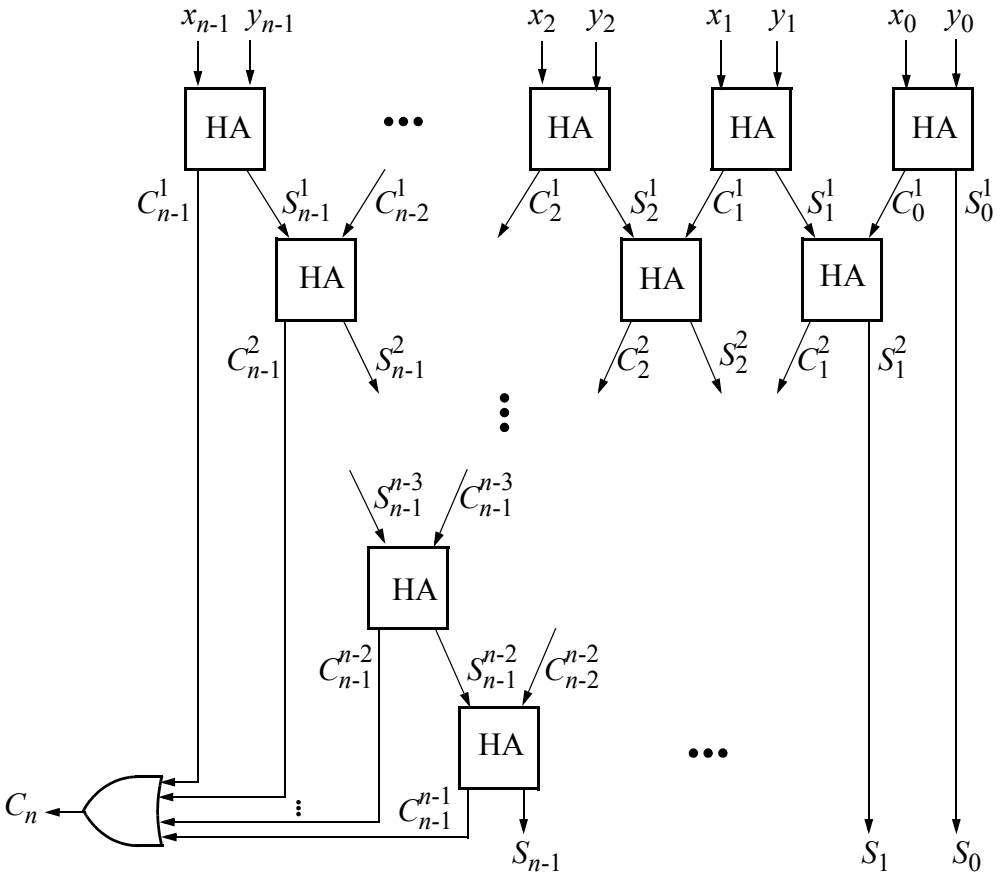


Fig. 3.33

Si noti che la traslazione verso sinistra del vettore dei riporti parziali è ottenuta semplicemente presentando agli ingressi del registro R1 la stringa di bit $c_{n-2}\ldots c_1 0$ e

trascurando il bit c_{n-1}.

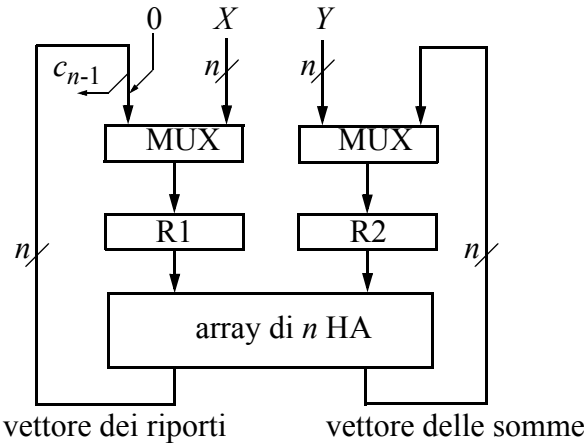


Fig. 3.34

Questo schema, pur di complessità ridotta rispetto a quello originario, presenta l'inconveniente di un tempo di risposta ancora più lungo, perchè ad un ritardo pari a quello della rete di Fig. 3.33 si aggiunge quello dei multiplexer e quello necessario per memorizzare i vettori parziali nei registri; quindi è di scarsa utilità quando si devono sommare solo due operandi. Introducendo però un terzo registro e sostituendo i semiaddizionatori con n addizionatori completi, la struttura diventa adatta per effettuare addizioni su un flusso continuo di dati (Fig. 3.35).

Ad ogni passo vengono sommati con tecnica carry save i contenuti dei tre registri che consistono in un nuovo dato di ingresso (R2), nel vettore delle somme parziali (R3) e in quello dei riporti parziali traslato (R1), questi ultimi derivanti dal passo precedente; in altri termini i nuovi contenuti dei registri R1 ed R3 al passo i-esimo sono rispettivamente:

$$R3^{(i)} = R1^{(i-1)} \oplus R2^{(i-1)} \oplus R3^{(i-1)}$$
$$R1^{(i)} = 2(R1^{(i-1)} \cdot R2^{(i-1)} + R1^{(i-1)} \cdot R3^{(i-1)} + R2^{(i-1)} \cdot R3^{(i-1)})$$

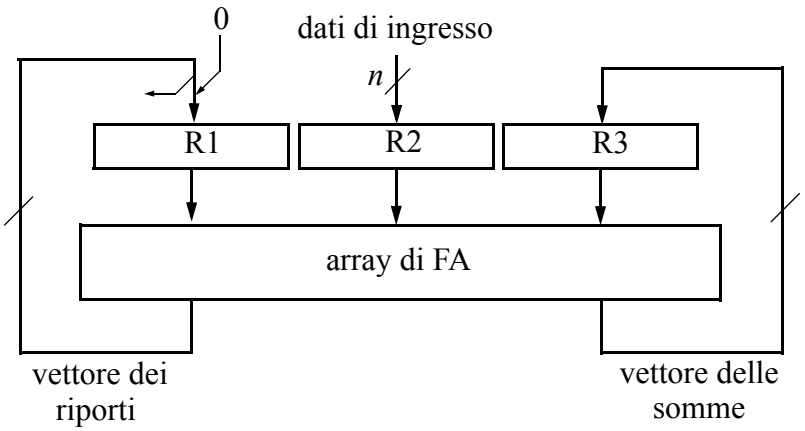


Fig. 3.35

Quando tutti gli addendi sono stati sommati, il registro R2 non partecipa più alle operazioni, mentre R1 ed R3 contengono i valori finali rispettivamente del vettore dei riporti e di quello delle somme. Per ottenere la somma corretta è necessario un ulteriore passo di addizione nel quale i due vettori sono sommati insieme con tecnica ripple carry (o carry look-ahead) ed il risultato viene memorizzato ad esempio nel registro R3: il tempo richiesto per quest'ultima addizione si ripercuote sulla durata di ciascuna somma per una frazione tanto più piccola quanto più numerosi sono i dati in ingresso. Pertanto il tempo necessario per calcolare ciascuna addizione, nel caso di N addendi, è circa uguale a $T_{FA}+T_R+T_A/(N\text{-}1)$, dove T_{FA} rappresenta il tempo di risposta di un full adder, T_R il tempo di memorizzazione in un registro e T_A il tempo dell'addizione finale. Per effettuare l'ultima somma si possono seguire due strade:

1) far lavorare l'addizionatore come quello di Fig. 3.34, mantenendo costantemente a 0 il registro R2 ed eseguendo con tecnica CS tra il contenuto di R1 e di R3 un numero di addizioni pari alla lunghezza dei vettori dei riporti e delle somme, a meno che non si arrivi prima alla situazione in cui il vettore dei riporti diventa 0;

2) riorganizzare l'array dei full adder già esistenti in modo da ottenere un addizionatore ripple-carry, come indicato in Fig. 3.36 per il bit k-esimo:

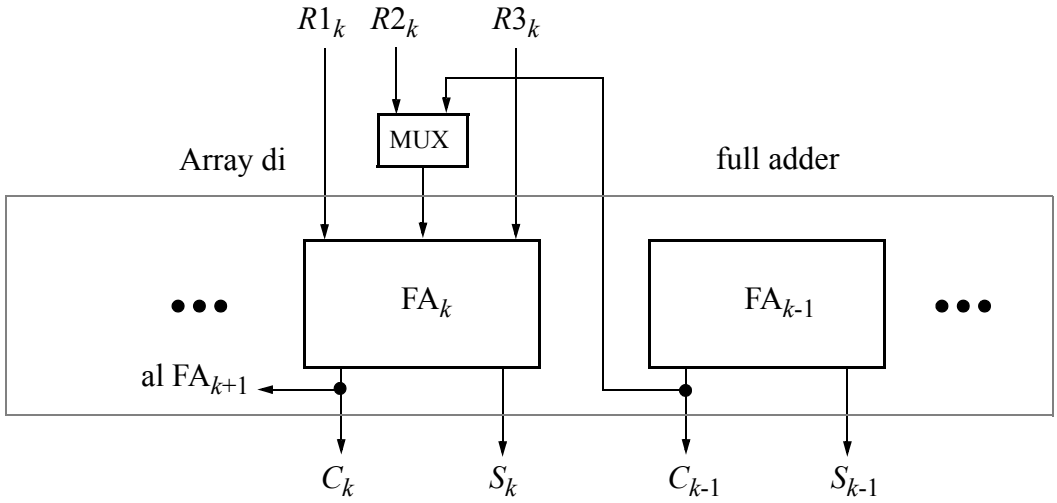


Fig. 3.36

Come ultima osservazione, è chiaro che i registri, il numero di addizionatori completi e la capacità delle linee di connessione devono essere dimensionati sulla lunghezza del risultato finale oppure, in alternativa, la loro dimensione condiziona il numero massimo di dati che possono essere sommati insieme.

Per esempio supponiamo di dover sommare quattro numeri $X_1 = 15 = 1111$, $X_2 = 12 = 1100$, $X_3 = 7 = 111$, $X_4 = 17 = 10001$; il risultato è $S = 51 = 110011$ e quindi occorre che i registri e l'addizionatore abbiano una lunghezza di almeno 6 bit. Il seguente schema riproduce i contenuti dei registri ai vari passi del calcolo. Inizialmente R1 ed R3 sono azzerati, mentre R2 contiene il primo addendo X_1; alla fine R3 contiene il risultato.

R1	R2	R3	operazione
000000	001111	000000	inizializza i registri
000000	001111	001111	calcola S^1 e C^1
000000	001100	001111	trasla C^1 e carica X_2
001100	001100	000011	calcola S^2 e C^2
011000	000111	000011	trasla C^2 e carica X_3
000011	000111	011100	calcola S^3 e C^3
000110	010001	011100	trasla C^3 e carica X_4
010100	010001	001011	calcola S^4 e C^4
101000		001011	trasla C^4
		110011	addiziona S^4 e C : risultato

3.7.1.5 - Addizionatore pipeline

Quando si tratta di calcolare una quantità di somme tra coppie di numeri, l'addizionatore CSA di Fig. 3.32 ritrova la sua utilità se tra una fila e l'altra di semiaddizionatori viene intercalata una coppia di registri con funzione di memoria temporanea per mantenere i vettori delle somme e dei riporti parziali (Fig. 3.37):

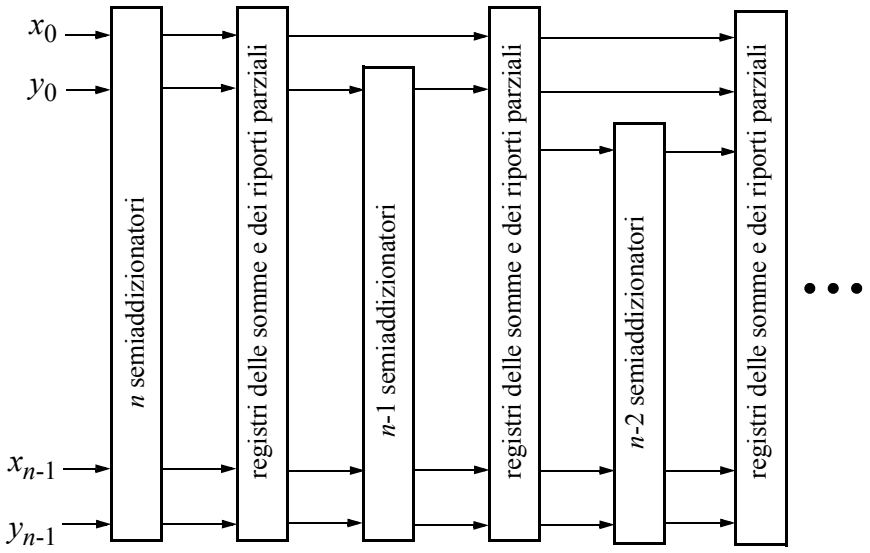


Fig. 3.37

Si ottiene un addizionatore che è detto **pipeline** e costituisce un'applicazione particolare di una tecnica di elaborazione molto diffusa, chiamata **pipelining**; si tratta di organizzare in cascata *k* elementi di calcolo ognuno dei quali fornisce come uscita l'ingresso al successivo. E' un modello simile a quello di una catena di montaggio nella quale esistono varie stazioni di servizio dove vengono effettuate le stesse operazioni su tutti gli esemplari che si muovono lungo la catena.

Nel caso dell'addizionatore, una coppia di addendi viene presentata all'ingresso del primo livello e l'uscita è memorizzata nella prima coppia di registri; nel momento in cui l'uscita di quei registri è presentata in ingresso ai semiaddizionatori del secondo livello, una seconda coppia di addendi può essere applicata all'ingresso del primo livello, e così via finchè dopo k passi tutti i livelli di addizionatori stanno lavorando, il k-esimo sulla prima coppia di dati, il (k-1)-esimo sulla seconda, il primo sulla k-esima. Da questo momento all'uscita del sistema cominciano a presentarsi i risultati che si susseguono con ritmo costante, uno ogni **passo di pipelining**, ossia uno ogni intervallo di tempo necessario perchè una stazione di servizio esplichi il proprio compito.

Si distingue così un periodo transitorio di k passi all'inizio, durante il quale la pipeline si riempie, uno di pari durata che ha inizio quando l'ultima coppia di dati viene ricevuta in ingresso e durante il quale la pipeline si svuota, ed un periodo di regime durante il quale la risorsa di calcolo è totalmente occupata. Se il numero m di coppie di addendi da sommare è considerevolmente più grande di k, l'incidenza del transitorio su ogni addizione diventa trascurabile e si può dimostrare che il tempo di risposta dell'addizionatore è costante; precisamente se k sono gli stadi di pipeline e T_e il tempo elementare di operazione, il numero di operazioni effettuate nell'unità di tempo è:

$$\nu = \frac{k}{T_e\left(1 + \frac{k-1}{m}\right)}$$

Per $m \to \infty$, ν tende al valore massimo k/Te.

3.7.2 - Moltiplicatori

La moltiplicazione è un'operazione che ricorre molto frequentemente in svariate applicazioni, spesso con vincoli stringenti di tempo di risposta, e per questo sono stati studiati molti algoritmi e molte strutture allo scopo di aumentare la velocità rispetto a quella dell'algoritmo base. Questo, noto come algoritmo di **moltiplicazione per somme e traslazioni**, deriva direttamente dall'espressione del prodotto di due numeri $X = x_{n-1}\ldots x_1 x_0$ e $Y = y_{n-1}\ldots y_1 y_0$, nella quale uno dei due fattori figura in forma polinomiale con base 2:

$$P = X \cdot Y = Y \cdot \sum_{j=0}^{n-1} x_j \cdot 2^j = \sum_{j=0}^{n-1} Y \cdot x_j \cdot 2^j$$

Poiché le cifre x_j valgono 0 o 1, l'espressione mette in evidenza che il prodotto si calcola come somma di al più n termini, ottenuti da uno dei due fattori traslato verso sinistra di 0, 1, ..., n-1 posizioni binarie; pertanto, nel caso peggiore, la complessità in tempo è $O(n^2)$ o, nella migliore delle ipotesi se le somme sono effettuate con tecnica CLA, $O(n\log n)$.

3.7.2.1 - Moltiplicatore a schiera

Per valori moderati di n, tipicamente per $n \leq 16$, è possibile effettuare la moltiplicazione usando una struttura combinatoria che deriva dalle seguenti considerazioni. Il prodotto di X ed Y può essere riscritto nella forma:

$$P = \sum_{j=0}^{n-1} Y x_j 2^j = \sum_{j=0}^{n-1} 2^j \left(\sum_{i=0}^{n-1} x_j y_i 2^i \right)$$

la quale mostra che è necessario prima di tutto generare n^2 prodotti da 1 bit del tipo $x_j y_i$. Poichè nel caso di numeri di un solo bit il prodotto aritmetico e quello logico coincidono, questi prodotti possono essere ottenuti semplicemente con una matrice di porte AND $n \times n$, come illustrato nella Fig. 3.38.

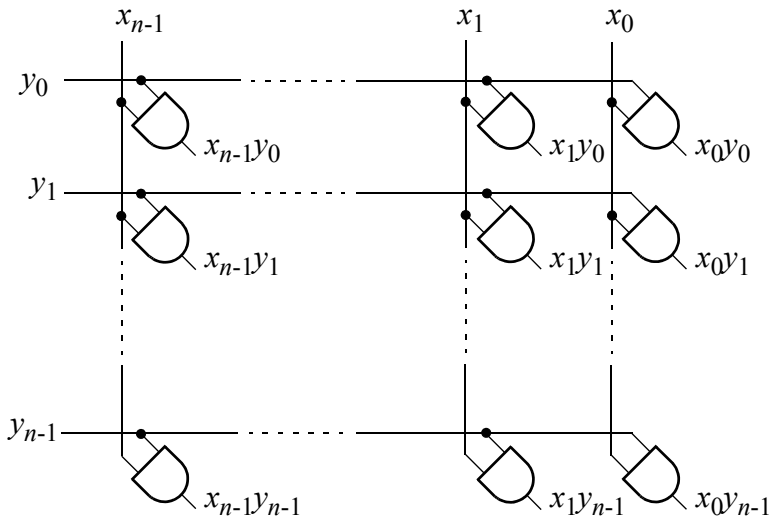


Fig. 3.38

Per calcolare la loro somma, poi, si usa una struttura formata da n semiaddizionatori ed n^2-$2n$ addizionatori organizzati secondo un array bidimensionale di n righe e $2n$-1 colonne; il bit k-esimo del prodotto, di peso 2^k, $0 \leq k \leq 2(n\text{-}1)$, viene generato dalla colonna corrispondente dell'array, secondo lo schema della Fig. 3.39, che per semplicità è disegnato per $n = 4$.

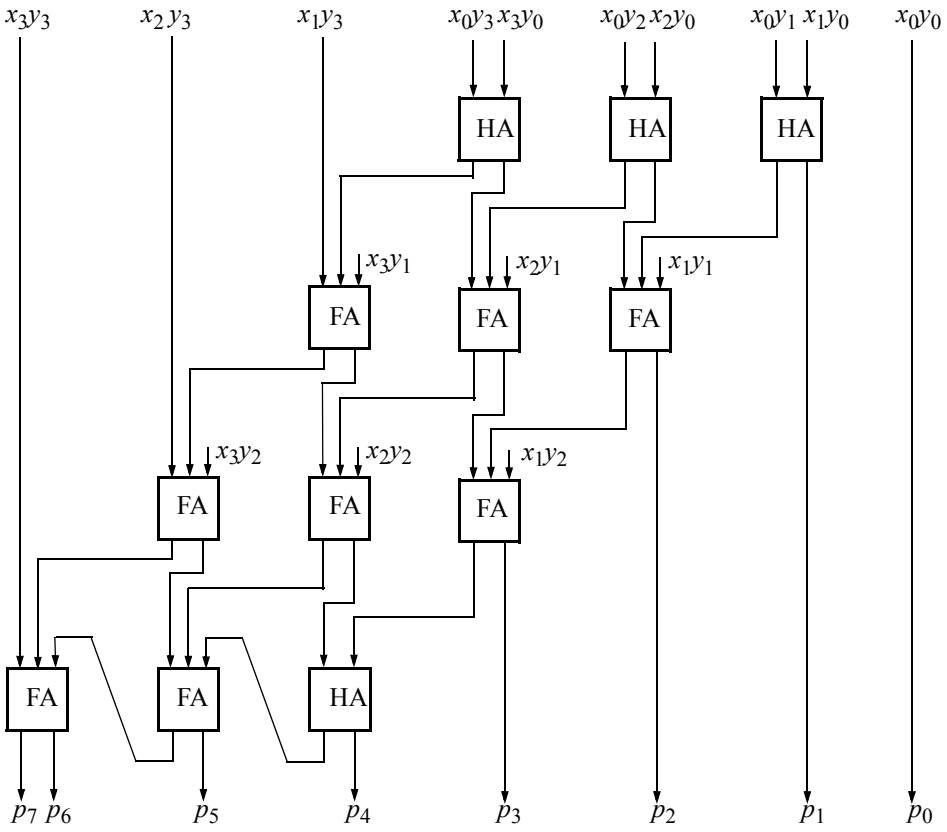


Fig. 3.39

Per spiegare la struttura di Fig. 3.39 scriviamo per esteso la doppia sommatoria della espressione del prodotto, ancora nel caso di $n = 4$:

$$\begin{aligned} P = \; & x_0y_0+x_0y_1 2+x_0y_2 2^2+x_0y_3 2^3+x_1y_0 2+x_1y_1 2^2+x_1y_2 2^3+x_1y_3 2^4+x_2y_0 2^2+x_2y_1 2^3+x_2y_2 2^4+ \\ & +x_2y_3 2^5+x_3y_0 2^3++x_3y_1 2^4+x_3y_2 2^5+x_3y_3 2^6 = \\ & =x_0y_0+(x_0y_1+x_1y_0)2+(x_0y_2+x_1y_1+x_2y_0)2^2+(x_0y_3+x_1y_2+x_2y_1+x_3y_0)2^3+ \\ & +(x_1y_3+x_2y_2+x_3y_1)2^4+(x_2y_3+x_3y_2)2^5+x_3y_3 2^6 \end{aligned}$$

Poichè sommando 4 bit si possono avere fino a 2 bit di riporto di valore 1 che vanno ad aggiungersi ai bit degli addendi di peso immediatamente superiore (in generale con k bit si possono avere $\lfloor k/2 \rfloor$ bit di riporto uguali ad 1), questa espressione mostra che dalla somma $x_0y_1+x_1y_0$ può nascere un riporto che si propaga dalla posizione di peso 2 a quella di peso 2^2; da questa possono aversi due riporti che devono essere aggiunti ai termini di peso 2^3; a loro volta possono generarsi tre riporti dalla posizione di peso 2^3 a quella di peso 2^4, ancora tre riporti di 1 dalla posizione di peso 2^4 a quella di peso 2^5 e infine da questa somma possono aversi due riporti di valore 1 da aggiungere al termine di peso 2^6.

Di qui nasce la struttura di Fig. 3.38, la quale prende il nome di **moltiplicatore a schiera**

ed è in sostanza un addizionatore ripple carry a due dimensioni; pertanto il tempo della moltiplicazione è dato dal ritardo di una porta AND della matrice più il doppio del ritardo di un addizionatore ripple carry ad *n* stadi, ovvero:

$$T_M = 2((n\text{-}1)\Delta_c + \max(\Delta_c, \Delta_s)) + \Delta_g$$

La complessità in tempo è dunque O(n), mentre quella in numero di componenti è O(n^2). Le prestazioni in tempo possono essere migliorate nel caso in cui siano da moltiplicare flussi continui di dati, facendo operare la struttura in pipeline, purchè siano intercalati tra le righe dell'array dei registri buffer.

3.7.2.2 - Moltiplicatore di Wallace

Come abbiamo visto, l'addizionatore CS ad *n* bit consiste di *n* addizionatori completi disgiunti ed ha come ingressi tre numeri e l'uscita è costituita da un vettore *S* (somma parziale) e da un vettore *C* (riporto parziale). La caratteristica importante è l'assenza di riporto all'interno dell'addizionatore, dal momento che solo dopo aver sommato tutti i dati di ingresso è richiesto un passo di acquisizione dei riporti, effettuato sommando i vettori *S* e *C* ad esempio con tecnica ripple carry sugli stessi addizionatori allo scopo opportunamente riorganizzati.

A loro volta i vettori *S* e *C* possono essere ingressi di un secondo addizionatore carry save ad *n* bit insieme ad un quarto numero *W* e così via, dando luogo ad una struttura di tipo pipeline, come illustrato in Fig. 3.40 per la somma di quattro numeri *X*, *Y*, *Z* e *W* di quattro bit:

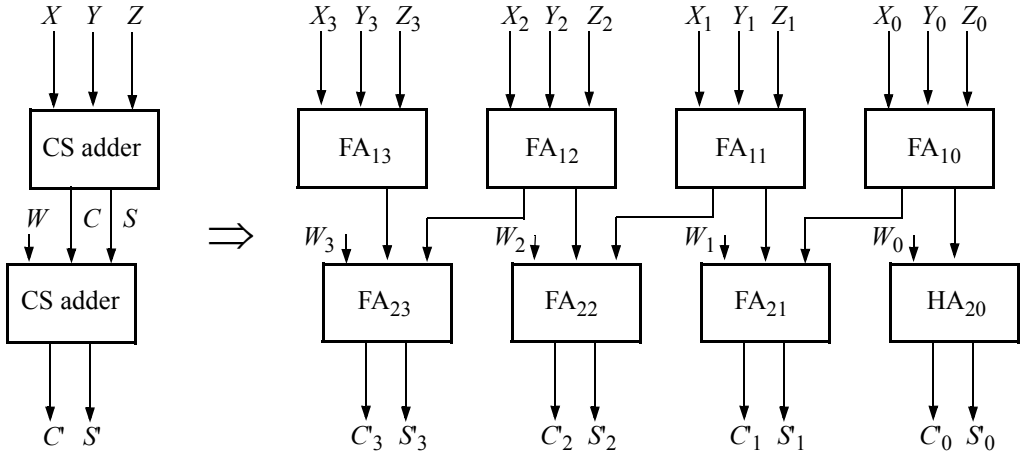


Fig. 3.40

Un addizionatore carry save a più stadi come quello sopra illustrato può essere utilizzato per eseguire la moltiplicazione secondo una struttura detta **moltiplicatore di Wallace**, esemplificato in Fig. 3.41 per numeri di 8 bit.

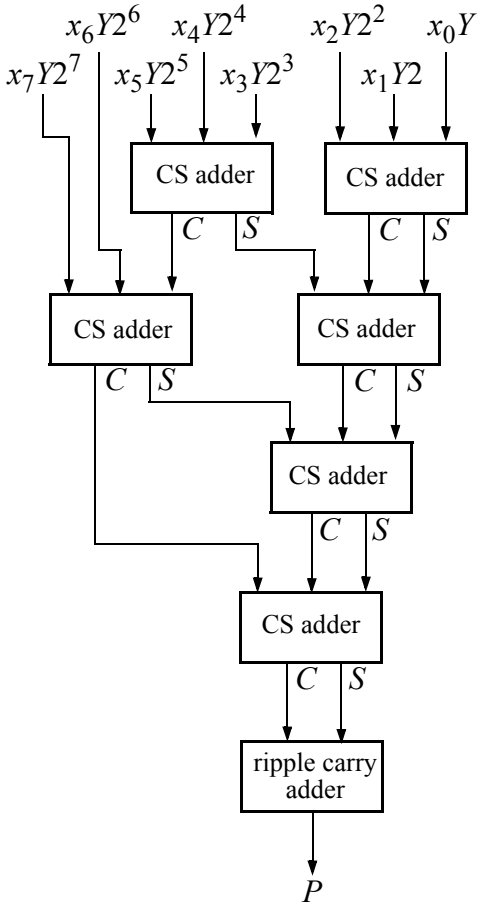


Fig. 3.41

Il suo funzionamento è definito dalle seguenti considerazioni. Il prodotto di due numeri X ed Y di n bit si può scrivere nella forma:

$$P = \sum_{i=0}^{n-1} M_i$$

nella quale vengono sommati insieme n termini di $2n$ bit ciascuno del tipo:

$$M_i = x_i Y 2^i$$

con $0 \le i \le n$-1, ottenuti moltiplicando Y per il bit i-esimo del moltiplicatore X, pesato con la potenza di 2 corrispondente.

Per effettuare la sommatoria può essere utilizzato un albero di addizionatori carry save che operano ciascuno su gruppi di tre ingressi, il quale produce due vettori C ed S di $2n$ bit; infine questi vettori sono addizionati insieme attraverso uno stadio finale di addizionatore ripple carry, come mostrato dalla Fig. 3.41.

Notare che per ottenere i termini M_i basta presentare su ciascuno degli n ingressi il valore

0 oppure il moltiplicando Y traslato verso sinistra di 2^i, a seconda del valore del bit x_i del moltiplicatore.

Il numero k di livelli dell'albero è legato alla dimensione n dei fattori dalla relazione:

$$k = \left\lceil \frac{\log_2 n - 1}{\log_2 3 - 1} \right\rceil$$

mentre il numero totale di addizionatori completi e semiaddizionatori è dell'ordine di n^2, come mostra la seguente tabella per diversi valori di n:

n	numero di addizionatori	numero di semiaddizionatori	totale	numero di livelli
6	16	12	28	3
8	38	15	53	4
12	102	29	131	5
16	192	56	248	6
24	488	87	575	7

Pertanto questa struttura è pratica solo se n è abbastanza piccolo. Per grandi valori di n il numero di addizionatori può diventare eccessivo ed in tal caso si ricorre alla tecnica di partizionare i dati in s segmenti di q bit in modo che sia $n = s \times q$ e di generare solo q prodotti M_i che vengono addizionati con un addizionatore del tipo descritto. Il procedimento viene ripetuto s volte ed ogni prodotto parziale viene accumulato insieme al risultato ottenuto dai passi precedenti.

ESERCIZI

1) Realizzare una rete combinatoria per la funzione $\Sigma_4(0,1,3,5,6,8,9,10,12,14)$ con multiplexer a 16, 8, 4 e 2 ingressi, utilizzando in tutti i casi il minor numero possibile di componenti e nell'ipotesi che si debba memorizzare l'intera tavola di verità della funzione. Si calcoli il ritardo delle varie reti costruite, esprimendolo in unità Δg.

2) Realizzare una rete per la funzione dell'esercizio 1, ammettendo di avere a disposizione solo multiplexer a 8 o a 4 ingressi e porte logiche AND e OR e rinunciando a memorizzare la tavola di verità completa.

3) Utilizzando esclusivamente multiplexer, si disegnino almeno due realizzazioni differenti della seguente funzione, escludendo la soluzione con un solo multiplexer:

$$f = \Pi_4(0, 1, 3, 4, 6, 9, 11, 12, 15)$$

4) Supponendo di disporre solo di porte XOR e di porte NOR a due ingressi, progettare una rete che accetta in ingresso numeri con segno di 6 bit e calcola sui bit 0÷4 la parità pari degli 1, se il numero è positivo; la parità pari degli 0, calcolata sempre sugli stessi 5 bit, se il numero è negativo.

5) Progettare con ROM un circuito che ha in ingresso valori di temperatura in gradi Celsius nell'intervallo da 0 °C a 20 °C di grado in grado e fornisce in uscita i corrispondenti valori in gradi Farenheit; si ricordi che tra le due scale termometriche esiste la relazione:

$$t_F = 32+1.8t_C$$

6) Ripetere l'esercizio 4 supponendo che i valori di ingresso siano specificati di due gradi in due gradi; qual è l'inconveniente ad utilizzare la stessa ROM dell'esercizio 4? Supponendo che, per qualche ragione, una ROM sia un circuito il cui costo cresce esponenzialmente con la capacità, come si potrebbe risolvere l'eventuale problema di costo derivante dal quesito precedente?

7) Progettare con ROM un circuito che accetta in ingresso valori di temperatura da 0 °C a 20 °C oppure da 32 °F a 68 °F e produce in uscita valori di temperatura nell'altra scala; realizzare la struttura in modo che ogni parola della ROM contenga sia un valore in °C, sia in °F.

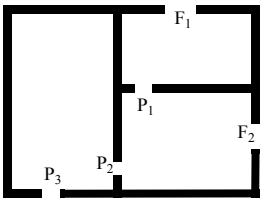
8) Progettare una ROM per tabulare con la precisione di 2^{-8} la funzione sin$x$, data di seguito per valori dell'argomento tra 0 e 90°, con passo 10°:

x	0	10°	20°	30°	40°	50°	60°	70°	80°	90°
sin$x$	0	0.1736	0.3420	0.5	0.6427	0.766	0.866	0.9397	0.9848	1

Supponendo di disporre di chip di memoria da 8 parole×4 bit, stabilire quanti chip sono necessari e come devono essere interconnessi.

9) Progettare un circuito che calcola il valore troncato all'intero della radice quadrata dei interi tra 0 e 100.

10) Un tizio molto freddoloso decide di installare nella sua casa, qui riprodotta in pianta, un sistema di allarme che faccia suonare un campanello quando porte e finestre sono aperte in modo tale da provocare correnti d'aria. Progettare il sistema di allarme.



11) Realizzare con multiplexer a quattro ingressi la seguente funzione:

		d,e							
		00	01	11	10	00	01	11	10
b,c	00	0		0	0	0	×	0	
	01		0	0		0	×	0	×
	11	0	0	×	0		0		0
	10	×	0	×	0		×		
		a = 0				a = 1			

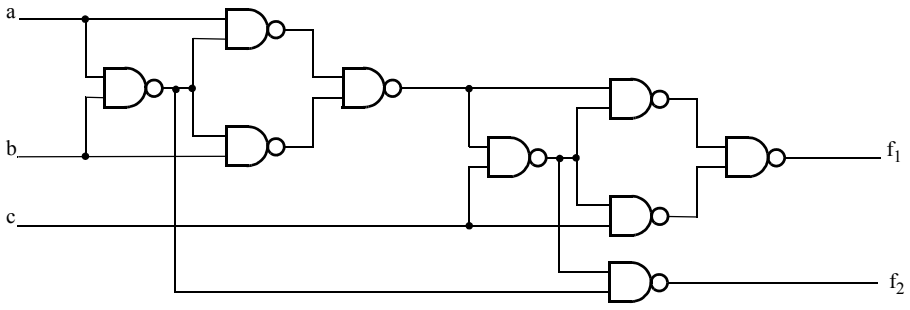
12) Disegnare lo schema a blocchi di un circuito con due ingressi *a* e *b* di *n* bit ed una uscita *z*, pure di *n* bit, il quale elabora nel seguente modo coppie di interi positivi applicati agli ingressi:

1) se *a* è dispari e *b* pari, calcola $z = \left\lceil \frac{a-b}{2} \right\rceil$;

2) se *a* è pari e *b* dispari, calcola $z = \left\lceil \frac{a+b}{2} \right\rceil$;

3) altrimenti $z = 0$.

13) Riconoscere che la rete di figura implementa un modulo FA:



14) Realizzare un circuito che calcola la somma modulo m di due numeri naturali A e B minori di m. Si osservi che nel caso peggiore la somma può valere $2m$-2 e quindi può traboccare dal campo al più una volta.

15) Progettare un circuito con due ingressi interi di 12 bit X ed Y che funziona nel seguente modo: se X-Y è maggiore o uguale a $\lceil X/2 \rceil$ il circuito dà in uscita X, altrimenti Y.

16) Progettare un circuito che riceve in ingresso su 8 bit un caratttere del codice ASCII (o Unicode) e se il carattere è una cifra (codici decimali da 48 a 57) produce in uscita il codice BCD di quella cifra; se il carattere è una lettera minuscola (codici decimali da 97 a 122) la trasforma in maiuscola (codici decimali da 65 a 90) e viceversa. Negli altri casi trasferisce in uscita il carattere entrante senza modifiche.

17) Realizzare un circuito con tre ingressi interi di 10 bit A, B, C, che produce 1 in uscita quando risulta

$$\min\{A, B\} < C < \max\{A, B\}$$

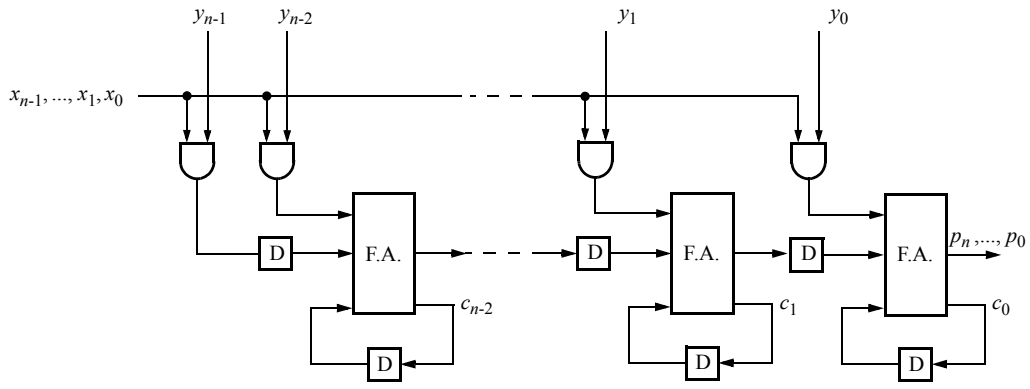
18) Realizzare un circuito con un ingresso intero di n bit X ed una uscita che assume valore $X/2$, quando X è pari, $(X+1)/2$ quando X è dispari.

19) Supponendo di disporre di circuiti per la generazione anticipata dei riporti della capacità di 5 bit, progettare un CLA per numeri di 28 bit e se ne valuti il tempo di risposta.

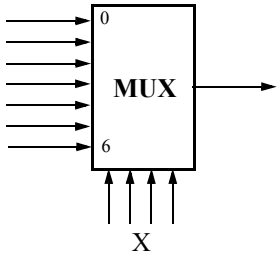
20) Calcolare la complessità, espressa come numero di HA/FA necessari, ed il tempo di risposta di un moltiplicatore di Wallace per numeri di 10 bit (o più in generale per numeri di n bit).

21) Disegnare lo schema a blocchi di un circuito sottrattore per numeri interi senza segno di n bit, definendo anche la struttura a livello porte di un modulo FS (Full Subtractor) che considera tre bit: minuendo, sottraendo e prestito.

22) Analizzare la funzione della rete logica di figura, nella quale i blocchi D sono memorie da 1 bit, i blocchi FA sono addizionatori completi x_i, y_i e p_i bit di tre numeri naturali X, Y e P.



23) E' richiesto un multiplexer a sette ingressi facendo uso di uno a 16 ingressi, ma poiché non si ha la sicurezza che gli ingressi dall'ottavo in poi non vengano mai selezionati, si decide di fare in modo che la selezione sia effettuata con la legge $Y = X \bmod 7$, essendo X l'equivalente decimale del codice di controllo applicato al multriplexer ed Y quello effettivamente operante (v. figura). Progettare il multiplexer e la logica di controllo.



24) Progettare una ROM la quale, dati due numeri a e b rispettivamente di tre e due bit, fornisce in uscita il valore $x = a^b$. Calcolare inoltre il numero totale di porte AND e OR necessarie per la ROM, supponendo che il fan-in sia 6.

25) Si disegni lo schema a blocchi di un circuito combinatorio che esegue il calcolo $x+y$, con $y = \text{abs}(2x+3)$, se $x > 0$. Per il calcolo di y si usi di una ROM.

26) Si sintetizzi mediante PLA in forma omogenea NAND la rete a quattro uscite:

$$f_1 = \Sigma_3(0,1,5,6)$$
$$f_2 = \Sigma_3(0,3,4,5,6)$$
$$f_3 = \Sigma_3(1,4,5,6)$$
$$f_4 = \Sigma_3(1,3,4,5)$$

Inoltre si sintetizzi la rete utilizzando un unico multiplexer a quattro ingressi, con 4 bit per ingresso.

27) Si disegni lo schema a blocchi di un circuito logico combinatorio che riceve in ingresso due numeri interi X ed Y di 12 bit e ne calcola rispettivamente la radice quadrata approssimata all'intero superiore ed il logaritmo in base due troncato all'intero inferiore; quindi produce come uscita un bit di valore:

1 se il numero degli 1 in $X^{1/2}$ è pari, 0 se è dispari, quando $X^{1/2} \geq \log_2 Y$;
1 se il numero degli 1 in $\log_2 Y$ è dispari, 0 se è pari, quando $X^{1/2} < \log_2 Y$.

Per il calcolo di $X^{1/2}$ e di $\log_2 Y$ si faccia riferimento a memorie di sola lettura, specificandone soltanto numero di parole e lunghezza di parola.

28) Progettare un decodificatore binario-BCD, che converte numeri interi rappresentati in binario su quattro bit nella rappresentazione 8-4-2-1 e segnala in uscita una situazione di errore per numeri maggiori di 9.