CAPITOLO II

LE RETI COMBINATORIE

2.1 INTRODUZIONE

Le reti combinatorie sono reti logiche caratterizzate dal fatto che lo stato dell'uscita all'istante t dipende solo dallo stato delle entrate allo stesso istante t (trascurando il ritardo di propagazione nelle porte).

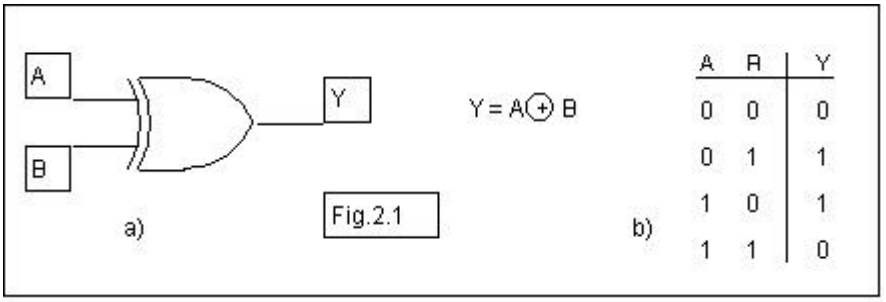
Nel capitolo successivo considereremo le reti sequenziali, nelle quali lo stato dell'uscita all'istante t dipende anche dagli stati assunti dalle entrate prima di t.

Considereremo nel seguito particolari reti combinatorie le quali implementano funzioni di uso comune, tanto da essere commercialmente disponibili sotto forma di singolo circuito integrato.

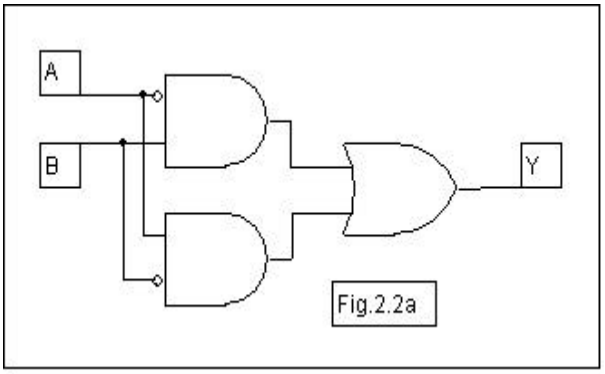
Ovviamente, la rassegna che segue è tutt'altro che esauriente, poiché vengono continuamente integrate su singolo chip funzioni sempre più complesse, man mano che il mercato le richiede. Gli esempi che seguono possono essere considerati utili esercizi di sintesi di reti logiche combinatorie.

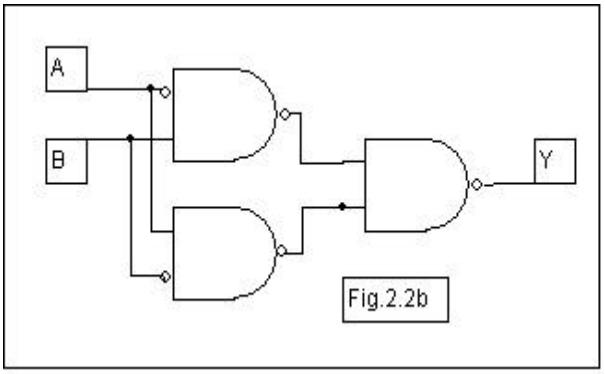
2.2 OR ESCLUSIVO

È una rete combinatoria con due entrate, A e B, ed una uscita Y, che realizza la seguente funzione: *"Y = 1 se A¹B"*.



Questa rete è particolarmente importante, ed ha un proprio simbolo circuitale, indicato in **Fig. 2.1** insieme con la tavola della verità, ed un proprio simbolo algebrico, indicato qui sotto.





La sintesi come S.d.P. dà

$$\overline{A}B + A\overline{B} = A \oplus B$$

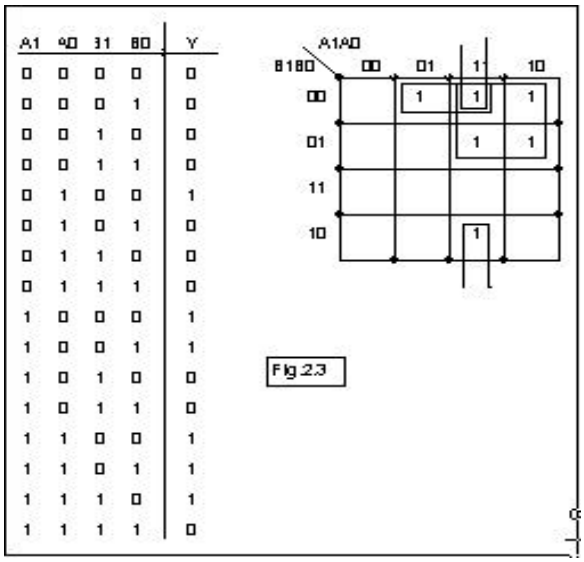
In **Fig. 2.2** è mostrata la rete sia in logica AOI che a NAND.

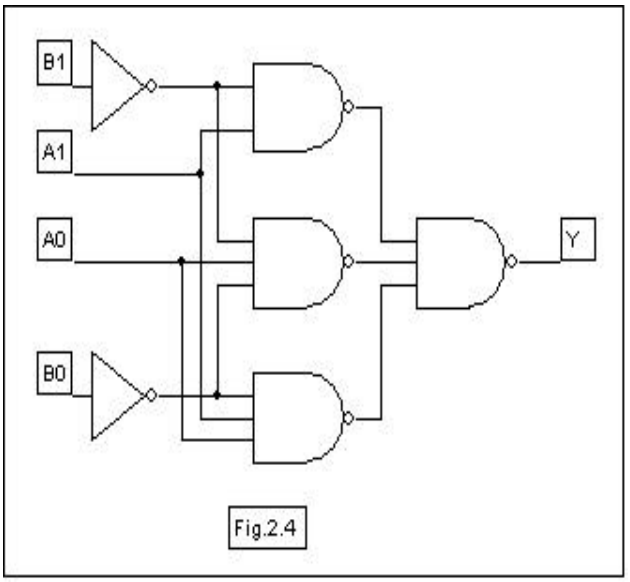
È anche disponibile il NOR esclusivo, che implementa la funzione:"*Y=0 se A¹B*"

$$Y = \overline{\overline{A}B + A\overline{B}} = AB + \overline{A}\ \overline{B} = \overline{A \oplus B}$$

2.3 COMPARATORE BINARIO

È una rete che confronta due numeri binari A e B, ciascuno di n bit (pertanto la rete ha 2n entrate),





implementando la funzione:

"Y=1 se A>B"

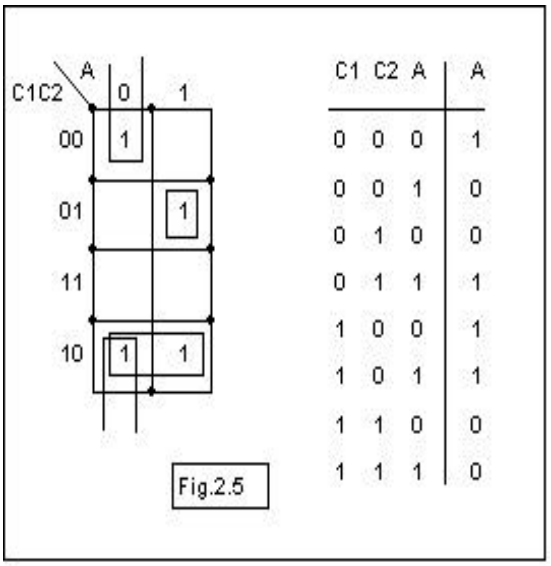
In **Fig. 2.3** è riportata la tavola della verità del comparatore, nel caso di parole a due bit, e la mappa di Karnaugh. La funzione sintetizzata è

$$A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0}$$

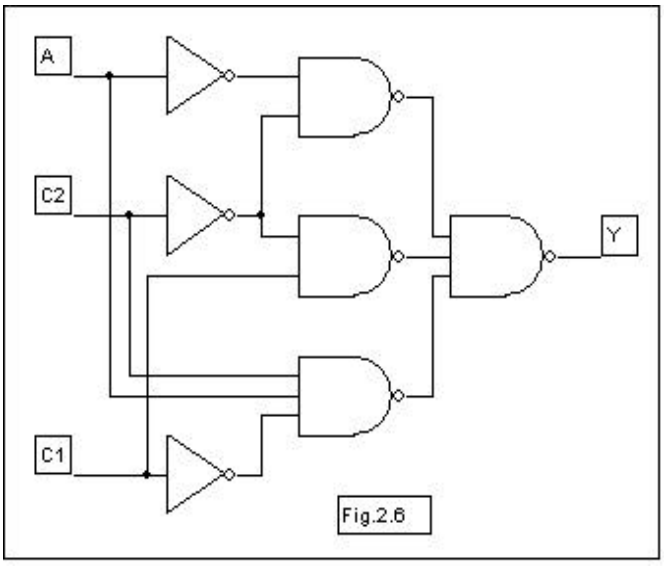
La **Fig. 2.4** mostra la rete in logica a NAND.

2.4 SELETTORE VERO/FALSO/UNO/ZERO

È una rete, avente un ingresso A, una uscita Y e due ingressi di controllo C_1C_2, la quale sotto l'azione



C1	C2	A	A
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



del controllo implementa la funzione*: "Y è uguale ad A, oppure ad* $\overline{A}$*, oppure ad 1, oppure a 0"*. La **Fig. 2.5** mostra la tavola della verità per una assegnazione (arbitraria) dei codici di controllo, e la mappa di K.

La funzione sintetizzata è

$$Y = \overline{C_2}\,\overline{A} + C_1\,\overline{C_2} + \overline{C_1}C_2A$$

La rete è in **Fig. 2.6**.

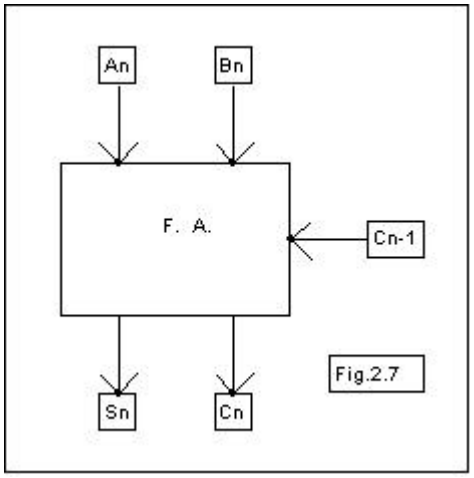
Questa rete può essere considerata come la versione più elementare di una rete combinatoria programmabile nota come Unità Logico-Aritmetica (ALU), che è una componente fondamentale di un microprocessore: questa rete agisce su due operandi, sui quali effettua sia operazioni di tipo logico (come quelle su descritte, ma molte altre), sia di tipo aritmetico (somma, sottrazione,....). E' comunque una rete troppo complessa per essere descritta in questo contesto. Si veda la letteratura.

2.5 SOMMATORE BINARIO

La somma di numeri binari si effettua, come per i decimali, sommando fra loro i bit dello stesso ordine, per esempio A_n e B_n, e il riporto C_{n-1} (Carry) proveniente dalla somma dei bit di ordine immediatamente inferiore. Il risultato sarà un bit somma S_n, ed un riporto C_n che andrà sommato ai bit di ordine n+1.

Nell'esempio che segue si sommano due numeri

```
1 1 0 0 1 1 1 1 0        riporti, C
1 1 0 0 1 1 1 1 1 +      I addendo, A
1 1 1 0 0 0 1 1 0 =      II addendo, B
_________________
1 1 0 1 1 0 0 1 0 1      somma, S
```



Si deduce che un sommatore per due numeri ad n bit è costituito da n celle uguali, ciascuna delle quali effettua la somma $A_n+B_n+C_{n-1}$, generando S_n e C_n; **Fig. 27**. La singola cella si chiama full adder.

Tavola della verità e mappe di K. per un full adder sono in **Fig. 2.8**.

An	Bn	Cn-1	Sn	Cn
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

Fig.2.8a

La sintesi dà le funzioni

$$S_n = \overline{A_n}\,\overline{B_n}C_{n-1} + \overline{A_n}B_n\overline{C_{n-1}} + A_n\overline{B_n}\,\overline{C_{n-1}} + A_nB_nC_{n-1}$$

$$C_n = B_nC_{n-1} + A_nC_{n-1} + A_nB_n$$

Conviene fare qualche elaborazione delle espressioni ottenute. Con la proprietà associativa e ricordando il paragrafo 2.2, si ha agevolmente

$$S_n = C_{n-1} \oplus (A_n \oplus B_n)$$

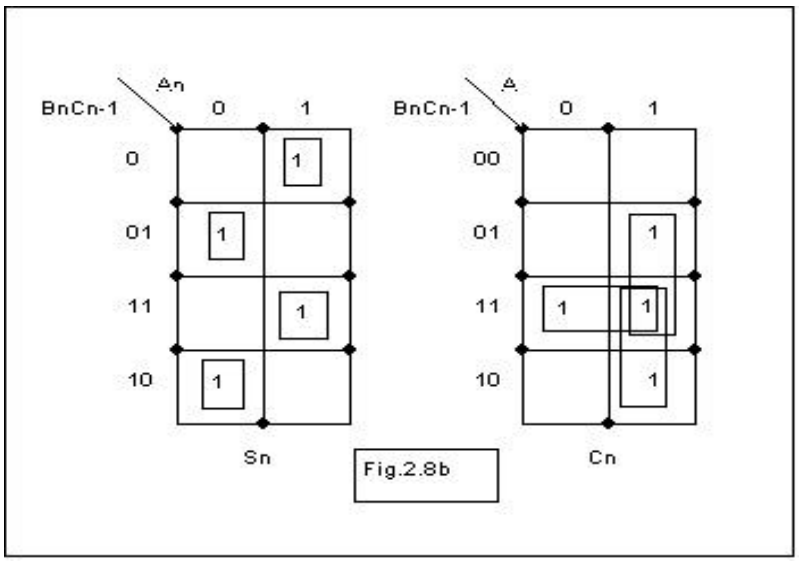
Per quel che riguarda C_n, si può scrivere

$$C_n = A_nB_n + C_{n-1}(A_n + B_n)$$

Si osservi ora che, quando $A_n=B_n=1$, C_n vale certamente 1 grazie al termine prodotto A_nB_n,

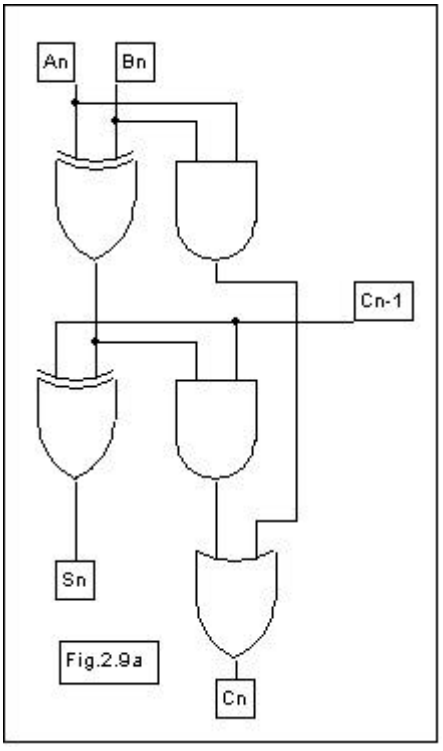
per cui si può scrivere anche

$$C_n = A_nB_n + C_{n-1}(A_n \oplus B_n)$$



Lo schema della rete è in **Fig. 2.9a**.

Il sommatore, **Fig. 2.9b**, è il principale costituente dell'unità aritmetica di un calcolatore, perché le altre tre operazioni possono essere ricondotte ad operazioni di somma. In figura, C_{-1} è ovviamente 0. Pertanto, è particolarmente importante la sua velocità operativa. Tale velocità è limitata dal tempo di propagazione del riporto. Si consideri infatti la cella i-ma in **Fig. 2.9b**: essa dà una uscita valida dopo che è giunto il riporto C_{n-1}, cioè dopo che la cella (i-1)-ma ha dato una uscita valida. Il ritardo massimo è quello relativo alla coppia dei bit più significativi. Pertanto, detto t_c e t_s, rispettivamente, il tempo che un full adder impiega per generare il bit di riporto e quello di somma, il tempo totale necessario per ottenere la somma valida di due numeri ad n bit è



$$t_d = (n-1)t_c + t_s$$

Questo ritardo è normalmente intollerabile, dato il gran numero di somme che il calcolatore deve fare ed il formato n delle parole usato nei sistemi di calcolo (32-64-128 bit). Per ridurre tale ritardo, si ricorre ad una rete combinatoria che calcola a parte i riporti, presentandoli in anticipo all'ingresso dell'opportuno full adder, che non deve più attendere la propagazione del riporto attraverso i full adder precedenti. Questa rete, nota come carry look-ahead, è disponibile su singolo integrato. La funzione logica di questa rete può essere ottenuta come segue. Da

$$C_i = A_iB_i + C_{i-1}(A_i \oplus B_i)$$

essendo

$$C_{i-1} = A_{i-1}B_{i-1} + C_{i-2}(A_{i-1} \oplus B_{i-1})$$

sostituendo nella precedente si ha

$$C_i = A_iB_i + A_{i-1}B_{i-1}(A_i \oplus B_i) + C_{i-2}(A_{i-1} \oplus B_{i-1})(A_i \oplus B_i)$$

Iterando la sostituzione, ed essendo $C_{-1} = 0$, si ottiene

$$C_i = A_iB_i + \sum_{j=1}^{i}\left[A_{i-j}B_{i-j}\prod_{k=1}^{j}\left(A_{i+1-k} \oplus B_{i+1-k}\right)\right]$$

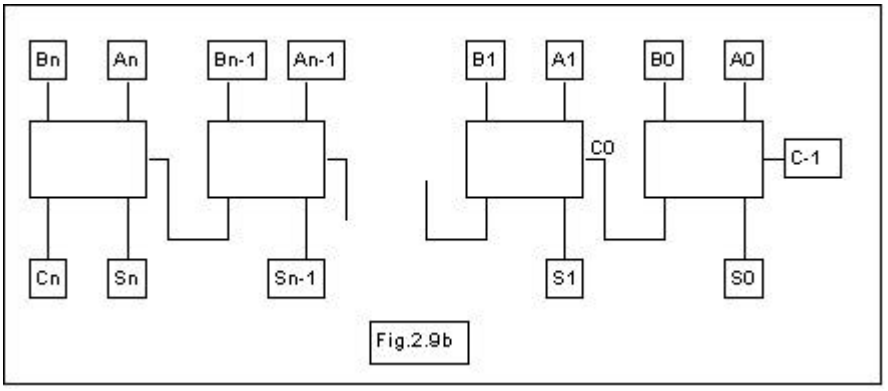


Fig.2.9b

Pertanto, ciascun riporto può essere calcolato, indipendentemente dagli altri, partendo dai bit delle due parole da sommare. Si può verificare che il numero di livelli richiesto è al più quattro.

2.6 GENERATORE DI PARITÀ

È una rete con n ingressi ed una uscita Y, che implementa la funzione: *"Y=1 se il numero di 1 nella stringa binaria di ingresso è pari"*. La tavola della verità può essere costruita come segue: supponiamo di aver contato gli 1 fino al bit (i-1)-mo della stringa di ingresso, e sia Y_{i-1} il risultato; allora, detto b_i il successivo bit della stringa, l'uscita Y_i sarà data dalla tavola della verità di **Fig. 2.10**, cioè

Yi-1	bi	Yi
0	0	0
0	1	1
1	0	1
1	1	0

Fig.2.10

$$Y_i = Y_{i-1} \oplus b_i$$

Il primo bit della stringa va confrontato con un 1 (infatti, "prima" del primo bit vi sono zero 1, e zero è un numero pari).

Il generatore di parità si implementa con una cascata di OR esclusivi; lo schema per stringhe di 4 bit è in **Fig. 2.11**.

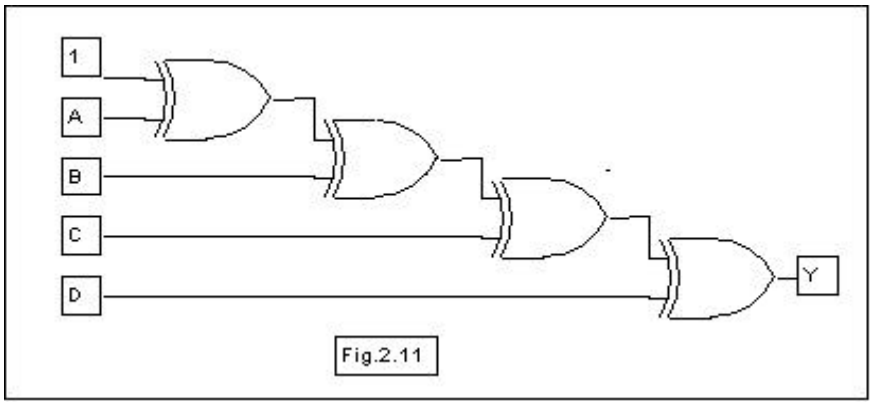


Fig.2.11

L'aggiunta di un ulteriore ingresso, E, permette di introdurre un bit di "controllo della parità": se E = 0, la parità della parola non viene cambiata; se E = 1, la parità viene ribaltata.

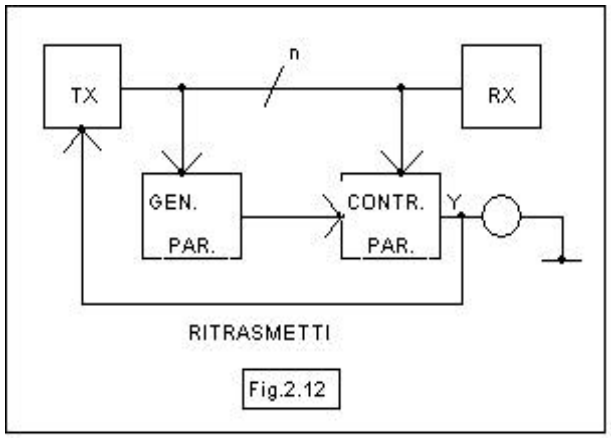


Fig.2.12

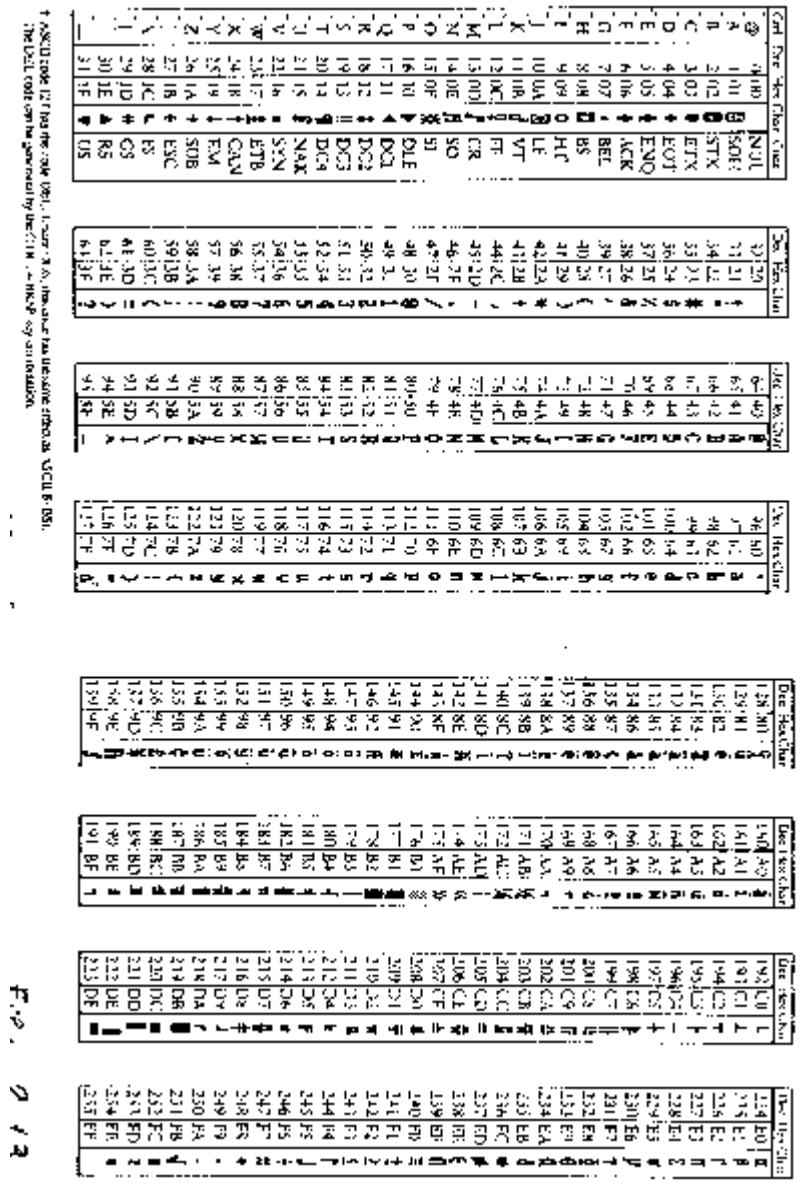
Il generatore di parità viene usato per rivelare errori nella trasmissione a distanza di parole binarie, secondo lo schema indicato in **Fig. 2.12**: la parità della parola che esce dal trasmettitore, TX, viene confrontata con quella giunta al ricevitore, RX. È facile verificare che, in assenza di errori, il controllo di parità deve

dare Y = 0.

Il sistema di **Fig. 2.12** non è ovviamente sicuro in assoluto: per esempio, fallisce se nel canale di trasmissione cambia contemporaneamente un numero pari di bit.

2.7 LA CODIFICA DELL'INFORMAZIONE

Nella comunicazione tra l'uomo e un sistema digitale di elaborazione, occorre che l'informazione sia espressa in un "linguaggio" comprensibile dagli interlocutori. Per quel che riguarda il sistema digitale, le "parole" del linguaggio sono scritte sulla base di un "alfabeto" di 2 simboli, 0 e 1; pertanto esse sono costituite da stringhe di simboli binari ordinati secondo un criterio prestabilito. Questo criterio, cioè l'algoritmo che permette di associare un messaggio per noi significativo ad una parola binaria, si chiama "codice". L'insieme delle parole del codice costituisce il "vocabolario" (codebook). La rete che effettua l'associazione si chiama codificatore (encoder).



Esistono moltissimi codici, alcuni di uso comune, altri studiati appositamente per un particolare problema (per es. la crittografia). Ci limiteremo ad accennare a qualcuno di quelli più diffusi.

Gli scopi che si vogliono ottenere con la codifica possono essere della natura più varia:

- facilitare l'input/output dell'informazione nel sistema digitale e l'esecuzione di calcoli (codici BCD);
- standardizzare lo scambio di informazione alfanumerica in forma bina-

ria (codice ASCII);

- agevolare la rivelazione di errori nella trasmissione a distanza dell'informazione (codici con bit di parità, codici con numero fisso di 1, ecc.);
- semplificare la codifica di angoli (codice Gray);
- minimizzare la ridondanza contenuta nell'informazione (codici di Huffman); -ecc.

Una semplice codifica si può fare associando, come parola del codice, a ciascun messaggio il numero d'ordine (indirizzo) del messaggio stesso in un elenco prestabilito, già noto agli interlocutori. Così, dovendo codificare N messaggi, occorrono N parole, ciascuna di n bit, essendo n il minimo intero tale che $2^n >= N$.

Di questa natura è il codice ASCII (American Standard Code for Information Interchange), che è un codice a 7 bit che codifica i 52 simboli alfabetici (maiuscole e minuscole), i 10 simboli numerici, ed altri simboli ed operazioni varie (parentesi, simboli di punteggiatura, controlli del formato di scrittura, ecc.). Questo è il codice che permette, per esempio, ad un operatore di colloquiare con un calcolatore tramite una tastiera. La **Fig. 2.13** mostra il codice ASCII nella versione estesa, che richiede 8 bit.

Per rappresentare informazione numerica decimale, risulta molto utile il codice BCD (Binary-Coded-Decimal): in questo codice, un numero decimale viene codificato in binario codificando singolarmente le sue cifre, in formato a 4 bit. L'encoder decimale -> BCD è quindi una rete con 10 ingressi e 4 uscite. Con 4 bit si possono scrivere 16 parole diverse, ma ne occorrono solo 10: si hanno quindi più codici BCD possibili. Uno di essi è il BCD naturale, nel quale cioè ogni cifra decimale è codificata con la sua espressione binaria naturale a 4 bit. È anche noto come codice 8-4-2-1, dal peso che assumono i 4 bit rispettivamente.

Consideriamo, per esempio, il decimale 521. La codifica BCD 8421 delle singole cifre è:

5 = 0101 2 = 0010 1 = 0001

Quindi

$$521_{(10)} = 010100100001_{(BCD)}$$

Il codice Gray ha la seguente caratteristica: due parole successive di questo codice differiscono per un solo bit. L'utilità di questo codice è particolarmente evidente quando si vogliono codificare angoli. Supponiamo, per esempio, di voler comandare mediante una rete digitale la rotazione di un motore passo-passo con una risoluzione di 1 grado (360 passi); usando un codice binario naturale occorrono 9 bit. Supponiamo che il motore sia posizionato sull'angolo 127 gradi (codice 001111111) e di comandare l'avanzamento a 128 gradi (codice 010000000). Se (come è probabile che avvenga a causa dei ritardi) i bit non cambiano contemporaneamente, può per esempio accadere che si transiti attraverso il codice intermedio 000000000 (i primi 7 bit diventano 0 prima che l'ottavo diventi 1), oppure 011111111 (l'ottavo bit diventa 1 prima che gli altri diventino 0). La conseguenza potrebbe essere una inutile, se non dannosa, sollecitazione per il meccanismo che esso sta eventualmente trascinando.

I codici Gray corrispondenti a 127 e 128 sono rispettivamente

001000000 e 011000000

che non presentano l'inconveniente menzionato.

La traduzione di un numero binario naturale a n bit in codice Gray a n bit viene effettuata con il seguente algoritmo: partendo dal bit meno significativo della parola binaria a n bit, il k-mo bit del codice Gray si ottiene effettuando l'OR esclusivo tra il k-mo ed il (k+1)-mo bit binario (k=0,1,...n-1). Esempio

0 0 1 1 1 1 1 1 1 codice binario

⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕

0 0 1 0 0 0 0 0 0 codice Gray

La decodifica Gray-binario naturale si ottiene con un algoritmo simile: partendo dal bit più significativo del codice Gray, il k-mo bit binario si ottiene facendo l'OR esclusivo tra il k-mo bit Gray ed il (k+1)-mo bit binario (k=n-1,n-2,...1,0). Esempio

1 1 0 1 1 1 0 0 1 codice Gray

⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕

1 0 0 1 0 1 1 1 0 codice binario

I codici per ridurre la ridondanza hanno la funzione di codificare i messaggi in modo tale che il numero medio di bit per messaggio sia quello minimo, necessario e sufficiente per la comprensione del messaggio stesso. Questi codici sono particolarmente utili, per esempio, per massimizzare la quantità di informazione trasmessa per unità di tempo (e quindi anche il numero di messaggi trasmessi per unità di tempo), su di un canale di comunicazione. È intuitivo che la velocità massima di trasmissione dell'informazione binaria (esprimibile in bit/sec) è finita, sia per i limiti dell'elettronica sia per quelli del canale che collega trasmettitore e ricevitore.

Supponiamo che il trasmettitore debba inviare n parole , ciascuna delle quali rappresenta uno di k messaggi diversi (k<<n), per es. un elenco di nomi. Se i k messaggi sono codificati in binario naturale (per es. secondo il loro ordine in un elenco), il codice sarà costituito da k parole ciascuna avente lunghezza fissa di b bit (b minimo intero tale che 2^b>=k). Se la velocità massima consentita dal canale di trasmissione usato è v bit/sec, il numero di messaggi trasmessi mediamente in un secondo è v/b.

La trasmissione può essere resa più efficiente (cioè si può aumentare il numero medio di messaggi trasmessi per unità di tempo) con considerazioni di tipo statistico. Supponiamo che i k messaggi non siano equiprobabili. Si può allora pensare di usare un codice con parole di lunghezza variabile, associando ai messaggi più probabili (cioè quelli trasmessi più di frequente) le parole di lunghezza più breve. In questo modo, la "lunghezza media" di una parola trasmessa sarà più breve di quella che si avrebbe usando parole di lunghezza fissa, con conseguente aumento del numero dei messaggi trasmessi mediamente per unità di tempo.

Per esemplificare, supponiamo che i messaggi siano 6, con probabilità di occorrenza

0.66, 0.3, 0.03, 0.005, 0.003, 0.002

rispettivamente.

Una codifica con lunghezza fissa richiede sei parole di 3 bit, per esempio le seguenti

000, 001, 010, 011, 100, 101

Ovviamente, la lunghezza media di ogni messaggio è 3 bit/messaggio, e la velocità media di

trasmissione è v/3 = 0.33v messaggi/sec.

Un risultato migliore è ottenuto usando per esempio le seguenti parole di codice per i sei messaggi, rispettivamente

1, 00, 011, 0100, 01010, 01011

Infatti, la lunghezza media di ogni parola (o messaggio) è

$$\langle l \rangle = \Sigma p_i l_i = 0.66 \cdot 1 + 0.3 \cdot 2 + 0.03 \cdot 3 + 0.005 \cdot 4 + 0.003 \cdot 5 + 0.002 \cdot 5$$

cioè 1.39 bit/messaggio. Il numero di messaggi mediamente trasmessi è ora v/1.39 = .72v messaggi/sec. È facile convincersi che le parole del nuovo codice sono "univocamente decodificabili", cioè i messaggi ad esse associati sono identificabili senza possibilità di confusione. Per es., la stringa 10000101100101010111000111010011010 10 rappresenta, nell'ordine, i messaggi n. 1, 2, 2, 6, 2, 1, 5, 1, 1, 1, 2, 3, 1, 4, 1, 1, 5.

Si può concludere dicendo che nella codifica dei sei messaggi usando parole di lunghezza fissa viene introdotta della informazione ridondante, cioè inutile ai fini della comprensibilità del messaggio. Huffman ha individuato un algoritmo di codifica statistica che minimizza la ridondanza, una volta nota la distribuzione di probabilità dei messaggi; ma su di esso non possiamo soffermarci. Il problema della riduzione della ridondanza è comune a tutti i sistemi di compressione dell'informazione.

L'esempio precedente è una compressione senza perdita di informazione (lossless). Con questo vincolo, purtroppo, difficilmente si possono ottenere fattori di compressione elevati, tranne pochi casi particolari in cui il set di messaggi da codificare è noto e limitato (ciò accade per es. in tipografia, ove si tratta di codificare i caratteri alfanumerici delle varie fonti usate per la stampa). Compressioni efficienti si possono ottenere rinunciando a informazione di secondaria importanza: per es. nella trasmissione di segnali video o audio, si può rinunciare a dettagli non percepibili dall'occhio o dall'orecchio, rispettivamente. Le tecniche di compressione oggi popolari (MP3, JPEG,...) si basano tutte su questo principio.

2.8 L'ENCODER

La rete digitale che assolve la funzione di associare a ciascun messaggio di un elenco una parola di un assegnato codice si chiama encoder. Essa avrà N ingressi, essendo N il numero di messaggi, e k uscite, essendo k il numero di bit necessario per scrivere N parole del codice. Per esempio, nel codice BCD è N = 10, per cui k = 4. Sintetizziamo ora un encoder BCD 8421.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Fig.2.14

La tavola della verità dovrebbe avere 1024 righe; tuttavia, associando 1 alla istruzione da codificare e 0 alle altre, le righe che interessano sono 10. Le restanti righe, che contengono nessun 1 o più di un 1 per riga, daranno uscite non-importa poiché siamo interessati a codificare una e una sola istruzione per volta. Consideriamo tutte = 0 le uscite non-importa e, per comodità, omettiamole dalla tavola della verità, che così appare come in **Fig. 2.14**.

Facciamo la sintesi di Y_2, come somma di prodotti.

$$Y_2 = \overline{A_9}\,\overline{A_8}\,\overline{A_7}...A_4...\overline{A_0} + \overline{A_9}\,\overline{A_8}\,\overline{A_7}..A_5..\overline{A_0} + \overline{A_9}\,\overline{A_8}\,\overline{A_7}A_6.....\overline{A_0} + \overline{A_9}\,\overline{A_8}A_7.....\overline{A_0}$$

Per la proprietà distributiva, indicando il fattore comune con K

$$= K(\overline{A_7}\,\overline{A_6}\,\overline{A_5}A_4 + \overline{A_7}\,\overline{A_6}A_5\,\overline{A_4} + \overline{A_7}A_6\,\overline{A_5}\,\overline{A_4} + A_7\,\overline{A_6}\,\overline{A_5}\,\overline{A_4})$$

Ora, deve essere certamente K = 1, altrimenti risulterebbe Y_2=0, mentre abbiamo considerato solo le righe dove Y_3=1. Allora, considerando i primi due termini prodotto, si ha

$$\overline{A_7}\,\overline{A_6}(A_5 \oplus A_4)$$

Poiché A_5 e A_4 possono essere solo singolarmente =1, come appare dalla tavola, si può anche scrivere

$$= \overline{A_7}\,\overline{A_6}(A_5 + A_4)$$

Sommando a questo risultato il terzo termine prodotto, si ha

$$\overline{A_7}\,\overline{A_6}(A_5 + A_4) + \overline{A_7}A_6\,\overline{A_5}\,\overline{A_4} = \overline{A_7}\,\overline{A_6}(A_5 + A_4) + \overline{A_7}A_6\,\overline{A_5 + A_4} = \overline{A_7}(A_6 + A_5 + A_4)$$

Sommando il quarto termine prodotto, si ha infine

$$Y_2 = A_4 + A_5 + A_6 + A_7$$

Analogamente, si trova

$Y_3 = A_8 + A_9$

$Y_1 = A_2 + A_3 + A_6 + A_7$

$Y_0 = A_1 + A_3 + A_5 + A_7 + A_9$

La rete è mostrata in **Fig. 2.15**.

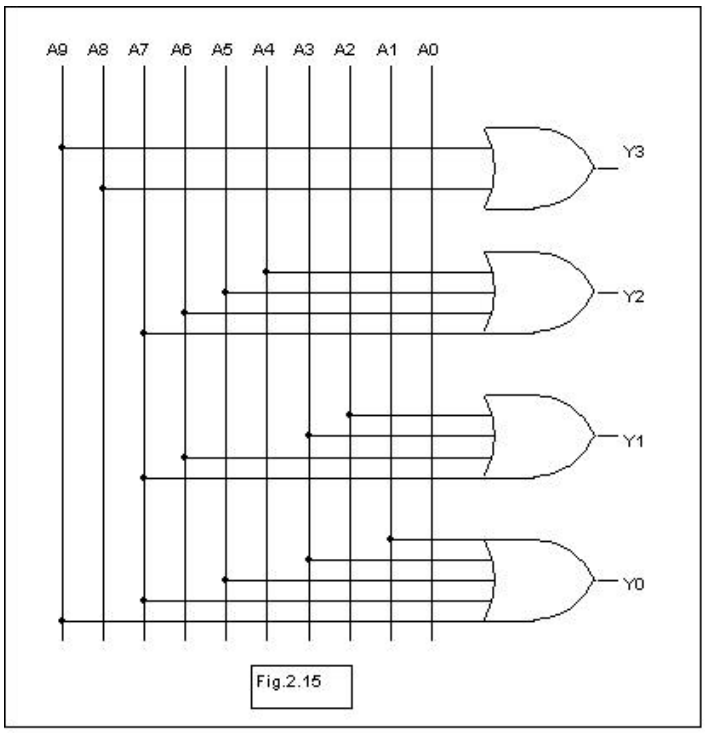


Fig.2.15

Potrebbe accadere, per es. per errore, che più di un ingresso al codificatore venga attivato contemporaneamente. L'indecisione viene risolta dal *codificatore con priorità* (priority encoder) il quale codifica una sola entrata con un criterio di priorità: tipicamente viene codificato l'ingresso che ha il valore binario più alto, e gli altri vengono ignorati. Una applicazione è nel caso in cui si abbiano eventi fisici indipendenti dei quali uno ha per noi più importanza degli altri.

La sintesi del priority encoder è lasciata come esercizio.

2.9 RETI DI DECODIFICA

Il decodificatore effettua l'operazione inversa del codificatore, cioè traduce il significato di una parola

A	B	C	D	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0													0	1	0
0	0	1	0	0													1	0	0
0	0	1	1	0												1			
0	1	0	0	0											1				
0	1	0	1	0										1					
0	1	1	0	0									1						
0	1	1	1									1							
1	0	0	0								1								
1	0	0	1							1									
1	0	1	0						1										
1	0	1	1					1											
1	1	0	0				1												
1	1	0	1			1													
1	1	1	0		1														
1	1	1	1	1															

Fig.2.16

del codice binario nel relativo messaggio.

Come esempio, sintetizziamo un decodificatore binario naturale -> decimale. La rete relativa avrà 4 ingressi e 16 uscite. In **Fig. 2.16** è la tavola della verità. È del tutto ovvio che la sintesi dà luogo alle seguenti equazioni

$$Y_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

$$Y_1 = \overline{A}\,\overline{B}\,\overline{C}D$$

.....

$$Y_{15} = ABCD$$

La rete è mostrata in **Fig. 2.17**.

Con un ulteriore ingresso su ogni AND (indicato tratteggiato in figura), si ha una linea di "strobe", S: l'uscita è considerata valida solo se S = 1. Il segnale di strobe è utile nei sistemi sincronizzati, in cui i

segnali devono essere letti solo al momento della loro utilizzazione.

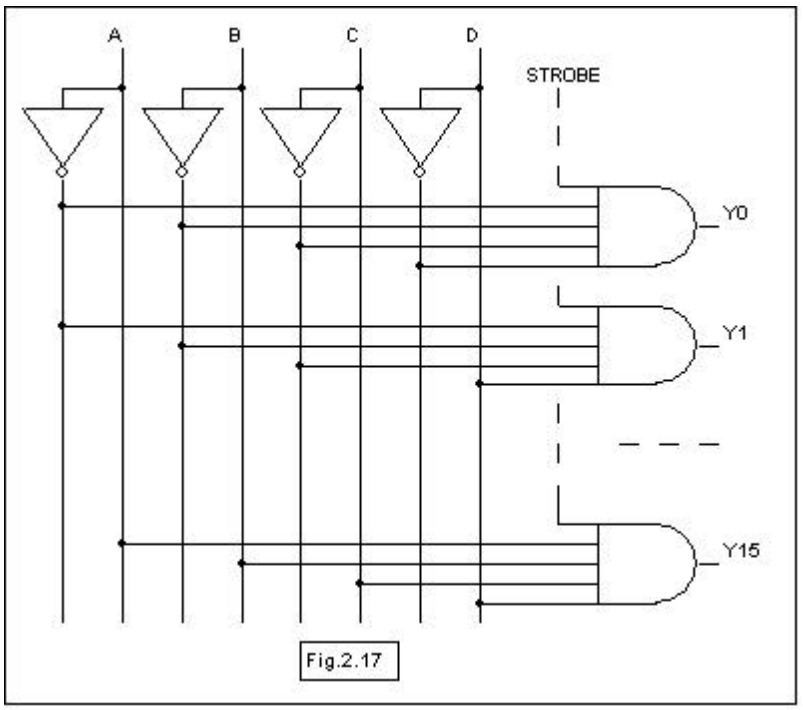


Fig.2.17

Un interessante utilizzo della linea di strobe è il seguente. Supponiamo, per esempio, che all'ingresso del decodificatore sia presente il codice 1000 (8, decimale). Allora, tutte le uscite, tranne quella dell'ottavo AND, saranno nello stato 0, indipendentemente da S, mentre l'uscita dell'ottavo AND sarà 1 oppure 0 a seconda che S sia 1 oppure 0, rispettivamente. Ne segue che, se inviamo una informazione sulla linea S, in forma binaria seriale, essa uscirà dall'uscita n. 8. Usato in questo modo, il decoder si chiama *demultiplexer*: S ha la funzione di ingresso dati, mentre il codice ABCD ha la funzione di indirizzo dell'uscita su cui si vuole instradare il dato presente su S.

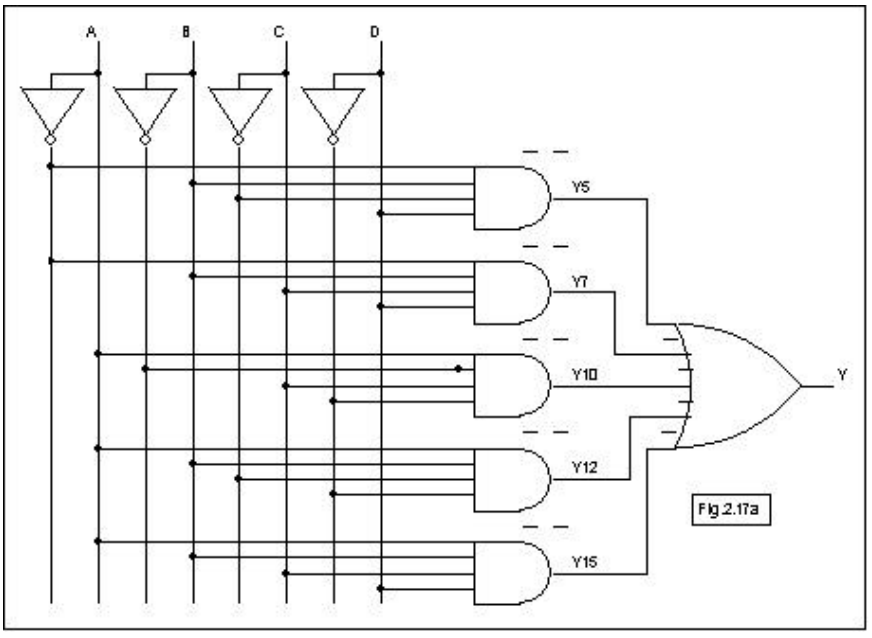


Fig.2.17a

È interessante infine notare che un decoder a n entrate con l'aggiunta di un OR può essere usato per implementare *una qualunque funzione di n variabili*. Infatti, con n variabili si possono generare 2^n termini canonici (quante sono le righe della tavola della verità). Allora,

una qualunque funzione di n variabili che contenga k (<=2^n) termini canonici può essere costruita con un decoder a n entrate e 2^n uscite più un OR a 2^n entrate, effettuando le opportune connessioni fra uscite del decoder e OR, come mostrato nell'esempio seguente. La funzione di 4 variabili

$$Y = \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}C\overline{D} + ABCD + AB\overline{C}\,\overline{D}$$

può essere implementata con un decoder a 4 entrate e 16 uscite ed un OR a 16 entrate, secondo lo schema mostrato in **Fig. 2.17a**, ove sono state connesse all'OR solo le uscite che decodificano i 5 termini canonici. Al cambiare della funzione di 4 variabili, si usa la stessa rete 'riprogrammando' opportunamente le connessioni. Si è costruito quindi una *rete logica universale*. Questo principio è alla base degli array logici programmabili (PAL, PLA, GAL,....) di cui si parla più avanti.

2.10 MULTIPLEXER

X0	X1	X2	X3	A	B	Y
0	-	-	-	0	0	0
1	-	-	-	0	0	1
-	0	-	-	0	1	0
-	1	-	-	0	1	1
-	-	0	-	1	0	0
-	-	1	-	1	0	1
-	-	-	0	1	1	0
-	-	-	1	1	1	1

Fig.2.18

Con questa rete logica, n linee dati possono essere collegate, una alla volta, ad una sola uscita.

La rete ha, quindi, n ingressi, 1 uscita e k linee di indirizzo (k essendo il minimo intero tale che $k>=\log_2 n$) che servono a selezionare uno degli ingressi. La funzione logica implementata dal multiplexer è: *"Y = 1 se l'ingresso della linea indirizzata è 1"*. Un esempio di tavola della verità per 4 linee dati è in **Fig. 2.18**. La funzione sintetizzata è

$$Y = X_0 \overline{A}\,\overline{B} + X_1 \overline{A}B + X_2 A\overline{B} + X_3 AB$$

La rete è in **Fig. 2.19**.

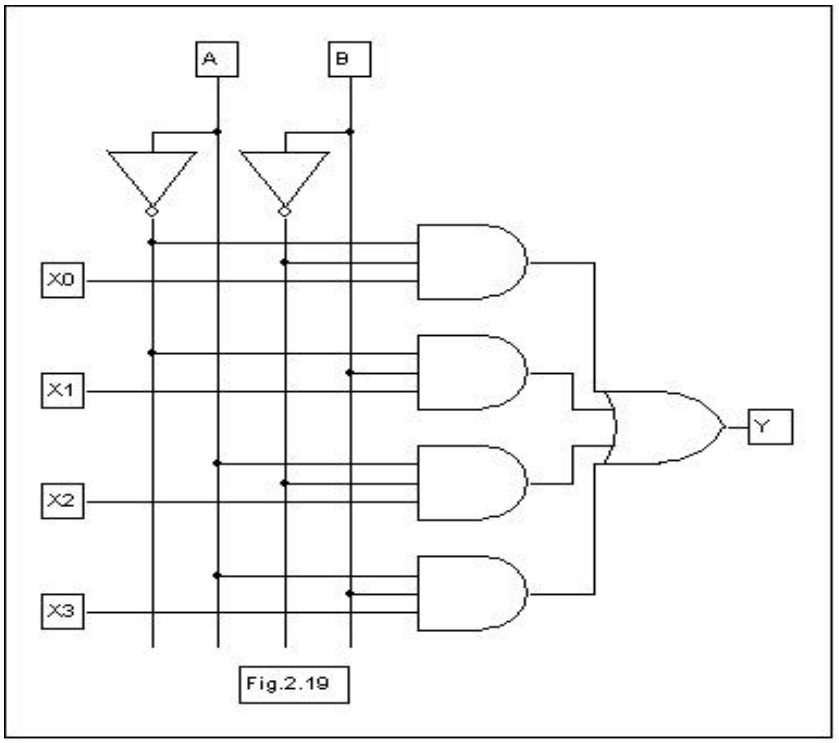
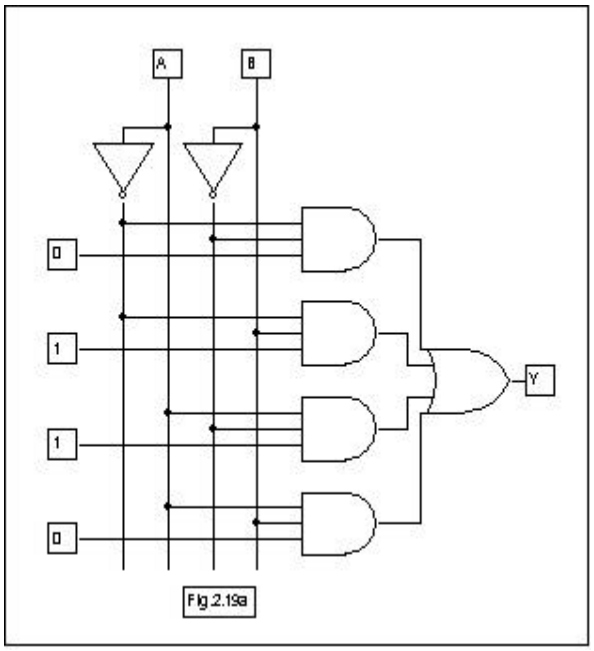


Fig.2.19

Il multiplexer a n entrate può essere usato per serializzare informazioni costituite da parole di n bit, presentate in parallelo sulle entrate che vengono poi scandite serialmente incrementando l'indirizzo (convertitore parallelo-seriale).

Dal confronto con la Fig. 2.17a, appare che un multiplexer con 2^n entrate è una rete logica universale, cioè può essere usato per implementare *qualunque funzione di n variabili*. Ciò è realizzabile usando le n variabili come indirizzo e ponendo a 1 le entrate il cui indirizzo è uno dei termini canonici della funzione e ponendo a 0 le altre entrate. Per es., la funzione $Y = \overline{A}B + A\overline{B}$ è realizzata con un multiplexer programmato come in **Fig. 2.19a**. Un'altra funzione di due variabili viene realizzata con la

stessa rete riprogrammando le connessioni di entrata.



2.11 RETI LOGICHE UNIVERSALI

Consideriamo il problema più generale di tradurre un codice numerico C1 (non necessariamente decimale) in un altro codice C2 (anch'esso non necessariamente decimale).

A ben vedere, qualunque rete logica a N entrate e M uscite effettua proprio questa operazione: infatti, essa converte il "codice di entrata" a N bit (le parole di questo codice sono gli stati delle variabili di entrata, cioè le righe della tavola della verità) in un "codice di uscita" a M bit (stati delle uscite), elencati nella stessa tavola.

Per reti complesse, una realizzazione a logica sparsa (random logic, cioè utilizzando singoli circuiti integrati commerciali) è in genere troppo onerosa (occorrono molti circuiti integrati e molto spazio). La tecnologia corrente permette di semplificare di gran lunga il problema, mettendo a disposizione circuiti integrati "programmabili", basati sul concetto di rete logica universale.

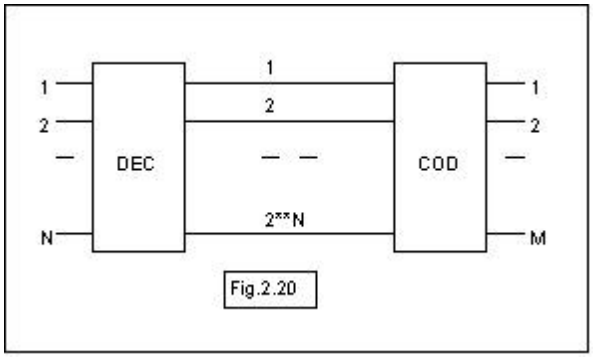


Fig.2.20

Se guardiamo alla disponibilità commerciale, esistono due modi di approccio al problema, così come due sono i modi di costruire una rete logica universale, uno basato sul decodificatore e l'altro sul multiplexer. Il primo modo consiste nella generalizzazione dell'esempio di fig. 2.17a che si riferisce al caso N=4, M=1. Il funzionamento di questa rete può essere descritto come segue: essa decodifica in decimale il codice di entrata a 4 bit (uscita degli AND) e ricodifica questo codice decimale nel codice di uscita a 1 bit (uscita dell'OR). La generalizzazione di questa architettura di rete logica universale è ovvia: il codice di ingresso, C1, a N bit, viene prima decodificato in 2^N parole decimali; il codice decimale viene quindi codificato nel codice di uscita, C2, a M bit, **Fig. 2.20**. La rete, in generale, sarà costituita da 2^N porte AND, ciascuna a N entrate (decodificatore), e da M porte OR, ciascuna a 2^N entrate (codificatore).

La **Fig. 2.21** mostra lo schema di una semplice matrice con N=2 e M=4. In questo caso, il decodificatore è già programmato; le connessioni nel codificatore non sono indicate, e saranno programmate a seconda dell'applicazione. La tecnica di programmazione è descritta più avanti.

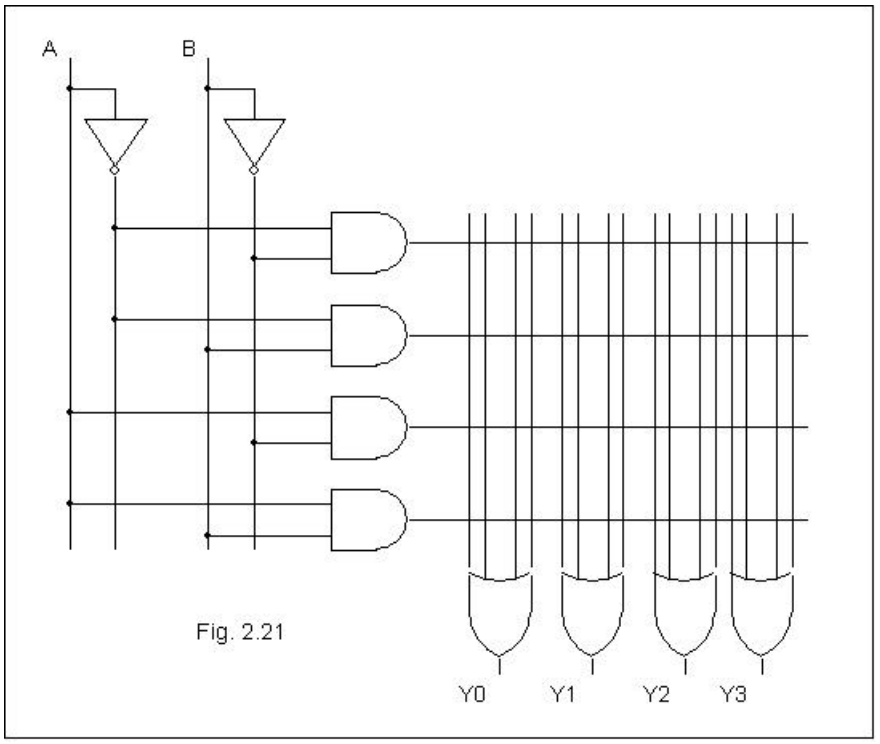


Fig. 2.21

Con un linguaggio immaginifico, possiamo dire che una rete siffatta si comporta come una memoria in cui è scritta la traduzione da C1 a C2: la parola di entrata alla rete può essere considerata come indirizzo di una "locazione di memoria" ove è "scritta" la parola corrispondente del nuovo codice, la quale viene presentata in uscita. La “scrittura” della memoria avviene quando si programma la rete; durante l’uso, ci si limita a “leggere” il contenuto della memoria: si parla pertanto di memoria read-only, ROM.

La disponibilità commerciale di reti universali programmabili del tipo sopra descritto è variegata. Nella memoria a sola lettura (ROM), codifica e decodifica sono già realizzate dal costruttore (ad hardware, cioè mediante collegamenti tra le porte logiche, realizzati una volta per tutte come in una rete cablata): pertanto, l'informazione immagazzinata è di tipo "non volatile" (cioè, permane anche con alimentazione spenta), e può essere solo letta quando lo si desidera. In una ROM si possono far risiedere stabilmente procedure ricorrenti, tavole di frequente consultazione, ecc.: ciò risulta particolarmente utile per es. nei calcolatori tascabili, perché così si evita di distruggere le tavole e le procedure ogni volta che si spegne il calcolatore stesso.

È ovvio che ordinare ad un costruttore delle ROM per un particolare codice può aver senso solo per grandi quantitativi; per applicazioni limitate e per la definizione di prototipi, sono disponibili reti programmabili dall'utente. Queste reti, chiamate genericamente PROM (Programmable ROM), hanno nomi specifici diversi a seconda della tecnica con cui sono realizzati i collegamenti e a seconda del livello di programmabilità. La programmazione avviene mediante semplici strumenti, disponibili commercialmente, da collegare ad un personal computer.

Si possono individuare due diverse tecnologie con cui vengono realizzati i collegamenti all'interno di una PROM. In un tipo di dispositivi, tutti i collegamenti potenzialmente utili vengono già realizzati dal costruttore tramite una lega fusibile; la programmazione del componente consiste nel "bruciare" (cioè interrompere) i collegamenti che non servono. In questo caso, i collegamenti bruciati non sono più recuperabili, ed è facile immaginare le conseguenze in caso di errore. In un altro tipo di dispositivi, che è anche il più diffuso (EPROM, Electrically ...), non esistono connessioni iniziali; mediante impulsi elettrici è possibile attivare nella matrice i collegamenti utili per realizzare la rete desiderata. È anche possibile "cancellare" il contenuto della memoria (tutti i collegamenti vengono interrotti), e successivamente riprogrammare il dispositivo.

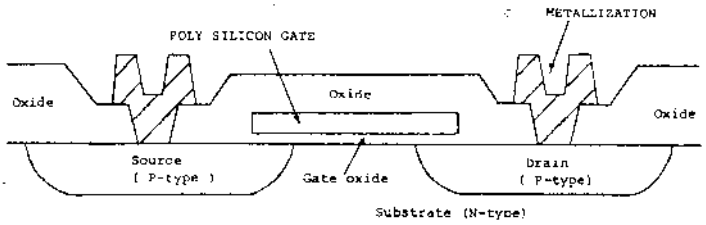


Figure 1 Physical Construction of Floating Gate

a)

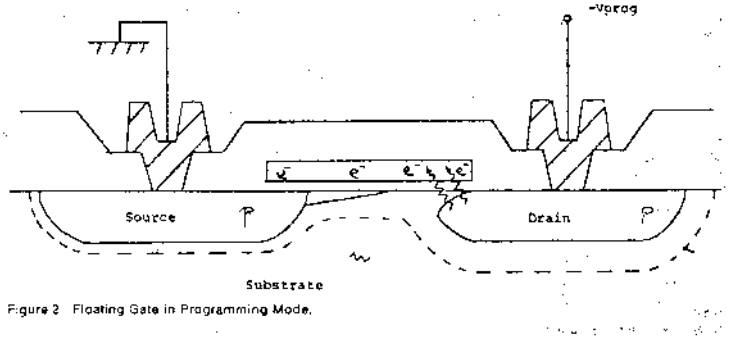


Figure 2 Floating Gate in Programming Mode.

b)

Fig. 2.22

Vediamo qualche dettaglio su come vengono effettuati i collegamenti in una struttura riprogrammabile. Tutti gli incroci nell'array sono collegati ai terminali Drain e Source di un MOS a gate flottante, **Fig. 2.22,** cioè non accessibile mediante un terminale esterno. Prima della programmazione, tutti i MOS sono aperti (resistenza di canale fra Source e Drain infinita). La programmazione (cioè l'attivazione dei collegamenti che interessano) avviene come segue: scelto il nodo ove si intende attivare un collegamento (si osservi che ogni nodo è indirizzabile in maniera unica, per righe e per colonne), si pone il Source a massa e si applica al Drain un impulso negativo di ampiezza tale da provocare una piccola scarica a valanga nella regione in cui Gate e Drain sono più vicini; un certo numero di elettroni viene così iniettato nel Gate, rimanendo ivi imprigionati. La carica iniettata nel Gate è quella suffi-

ciente a provocare l'inversione del canale (la carica positiva ivi indotta soverchia la carica negativa presente), e quindi la conduzione fra Source e Drain.

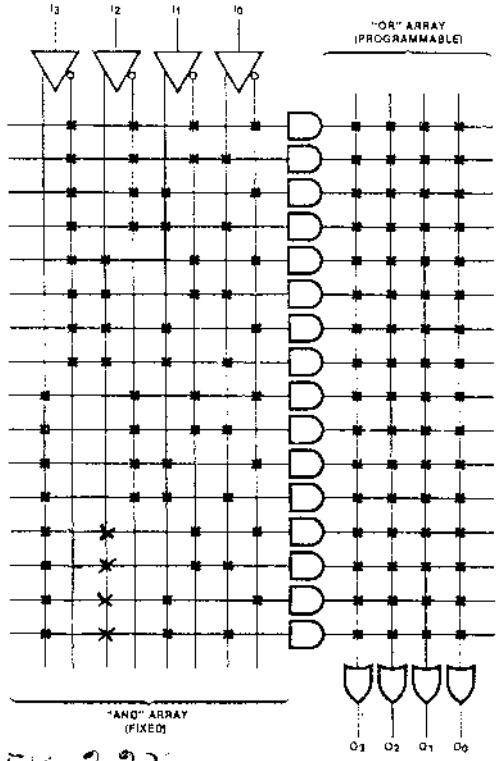


Fig. 2.23 PROM Having 16 Words x 4 Bits.

La cancellazione dell'EPROM avviene illuminando il chip mediante luce ultravioletta, la quale giunge sui Gate di tutti i MOS attraverso una finestra di quarzo, sotto la quale c'è il chip di Silicio. La luce viene assorbita dagli elettroni nel Gate, i quali acquistano così sufficiente energia per tornare indietro nel Drain. L'illuminazione deve avere intensità e durata indicate dal costruttore, per garantire il ripristino della neutralità elettrica nei Gate e quindi l'interruzione dei collegamenti.

Nelle EEPROM (Electrically Erasable ....) anche la cancellazione avviene mediante impulsi elettrici.

Le PROM**, Fig. 2.23**, sono in genere destinate a quelle applicazioni in cui il codice di entrata è un codice binario naturale; in questo caso il decoder è pre-programmato dal costruttore, e l'utente si limita a programmare l'encoder (come in fig. 2.21).

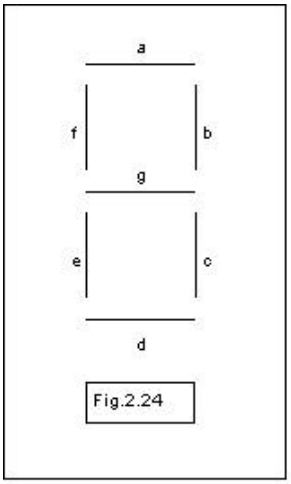


Fig.2.24

Nelle PAL (Programmable Array Logic), è programmabile il decoder, mentre nelle FPLA (Field Programmable Logic Array), sono generalmente programmabili sia il decoder che l'encoder. La loro struttura interna è del tutto simile a quella di Fig. 2.23.

Vediamo come esempio una rete che permette di passare dal codice BCD al codice a 7 segmenti, usato nei ben noti display, **Fig. 2.24**. In accordo con quanto detto più sopra, si effettua prima la decodifica BCD -> decimale, quindi la codifica decimale -> 7 segmenti. La tavola della verità è in **Fig. 2.25**.

Si lascia come esercizio il verificare che la decodifica dà luogo alle seguenti equazioni (si noti che

nella tavola della verità ci sono sei stati non-importa):

$Y_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ $Y_1 = \overline{A}\,\overline{B}\,\overline{C}D$ $Y_2 = \overline{B}C\overline{D}$ $Y_3 = \overline{B}CD$

$Y_4 = B\overline{C}\,\overline{D}$ $Y_5 = B\overline{C}D$ $Y_6 = BC\overline{D}$ $Y_7 = BCD$

$Y_8 = A\overline{D}$ $Y_9 = AD$

A	B	C	D	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1
0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0	1
0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1	1
0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	1
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	1	1	1
0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1

Fig.2.25

Analogamente a quanto fatto nel paragrafo 2.8, è facile verificare che le equazioni di uscita sono le seguenti:

$a = Y_0+Y_2+Y_3+Y_5+Y_6+Y_7+Y_8+Y_9$ $b = Y_0+Y_1+Y_2+Y_3+Y_4+Y_7+Y_8+Y_9$

$c = Y_0+Y_1+Y_3+Y_4+Y_5+Y_6+Y_7+Y_8+Y_9$ $d = Y_0+Y_2+Y_3+Y_5+Y_6+Y_8+Y_9$

$e = Y_0+Y_2+Y_6+Y_8$ $f = Y_0+Y_4+Y_5+Y_6+Y_8+Y_9$

$g = Y_2+Y_3+Y_4+Y_5+Y_6+Y_8+Y_9$

La rete corrispondente può essere realizzata programmando una prom del tipo di fig. 2.23, con N=4 e M=7.

E' anche possibile una soluzione alternativa che fa uso di porte open-collector. Con il teorema di De Morgan, le equazioni dell'encoder si riscrivono:

$$a = \overline{\overline{Y_0}\ \overline{Y_2}\ \overline{Y_3}\ \overline{Y_5}\ \overline{Y_6}\ \overline{Y_7}\ \overline{Y_8}\ \overline{Y_9}}$$

$$b = \overline{\overline{Y_0}\ \overline{Y_1}\ \overline{Y_2}\ \overline{Y_3}\ \overline{Y_4}\ \overline{Y_7}\ \overline{Y_8}\ \overline{Y_9}}$$

$$c = \overline{\overline{Y_0}\ \overline{Y_1}\ \overline{Y_3}\ \overline{Y_4}\ \overline{Y_5}\ \overline{Y_6}\ \overline{Y_7}\ \overline{Y_8}\ \overline{Y_9}}$$

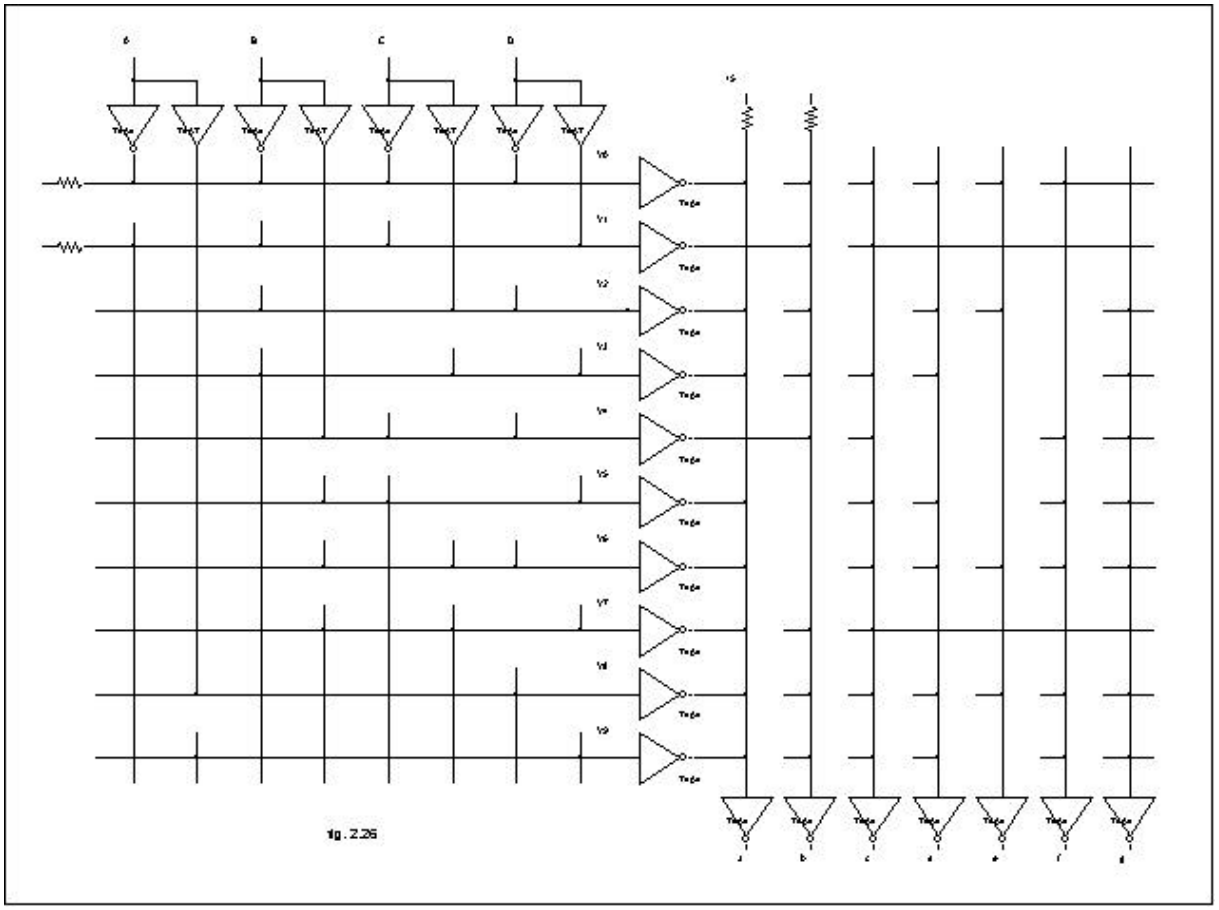
$$d = \overline{\overline{Y_0}\ \overline{Y_2}\ \overline{Y_3}\ \overline{Y_5}\ \overline{Y_6}\ \overline{Y_8}\ \overline{Y_9}}$$

$$e = \overline{\overline{Y_0}\ \overline{Y_2}\ \overline{Y_6}\ \overline{Y_8}}$$

$$f = \overline{\overline{Y_0}\ \overline{Y_4}\ \overline{Y_5}\ \overline{Y_6}\ \overline{Y_8}\ \overline{Y_9}}$$

$$g = \overline{\overline{Y_2}\ \overline{Y_3}\ \overline{Y_4}\ \overline{Y_5}\ \overline{Y_6}\ \overline{Y_8}\ \overline{Y_9}}$$

Tutti gli AND, sia nel decoder che nell'encoder, possono essere realizzati in logica cablata, usando porte a collettore aperto del tipo descritto nel paragrafo 1.16. Lo schema di principio della rete è in **Fig. 2.26**. I nodi marcati rappresentano i collegamenti fisici nell'array, che realizzano appunto l'AND cablato (il segnale a ogni nodo arriva da una gate separata). Sono anche indicate alcune delle resistenze di pull-up richieste dalle porte open-collector.



Una classe più evoluta di reti logiche universali si basa sul multiplexer, ed i componenti sono noti come FPGA (Field Programmable Gate Array). In questi dispositivi, è presente una matrice di celle ognuna delle quali è un multiplexer (tipicamente a 4 entrate). La programmazione consiste nel colle-

gare tra loro le singole celle fino a realizzare la rete desiderata. Normalmente sono presenti sia celle di tipo combinatorio (MUX a 4 entrate) che sequenziale (flip flop, vedi capitolo 3), per soluzioni del tutto generali. I costruttori forniscono le librerie in cui sono descritte le caratteristiche elettriche delle celle, un apposito software di programmazione nonché il programmatore vero e proprio (che è uno strumento dotato di uno zoccolo sul quale si inserisce il circuito integrato da programmare), da collegare ad un PC. Questi software di programmazione sono oggi altamente sofisticati, perchè consentono di descrivere il circuito da realizzare non mediante equazioni booleane o uno schema, bensì mediante istruzioni di un linguaggio astratto in cui si esprime a parole cosa deve fare il circuito. Il linguaggio Verilog HDL (Hardware Description Language) è uno dei più versatili. Un sintetizzatore ricrea lo schema dalla nostra descrizione verbale. Un CAD elettronico, per es. Pspice, utilizza le librerie per fare una simulazione realistica del circuito che si vuole realizzare; quando il progetto è ritenuto soddisfacente, il software provvede a programmare l'FPGA, cioè a riprodurre nel suo interno il circuito. La tecnologia corrente (2001) consolidata fornisce matrici che contengono più di 120 mila gates equivalenti (per gate equivalente si intende, per convenzione, un NAND a 2 entrate), che possono operare a frequenze di clock superiori a 100 MHz.