UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

POLO DIDATTICO E DI RICERCA DI CREMA

APPUNTI DI

RETI LOGICHE

Nello SCARABOTTOLO

Versione 23 ottobre 2003

INDICE

1 INTRODUZIONE

La prima parte dell'insegnamento di Architettura degli Elaboratori I affronta le metodologie di base utilizzabili per la progettazione dei circuiti elettronici digitali, ovvero dei componenti che consentono la realizzazione della parte fondamentale di un qualsiasi elaboratore o calcolatore elettronico.

Scopo di questa introduzione è quello di inquadrare gli argomenti trattati nei capitoli successivi, fornendo una chiave di lettura che consenta sempre di collegare i singoli argomenti all'aspetto realizzativo dei calcolatori cui si vuole fare riferimento.

1.1 PERCHÉ GLI ELABORATORI ELETTRONICI

Per elaboratore o calcolatore intendiamo in questo contesto una macchina progettata per **elaborare informazioni in modo automatico**, cioè un sistema capace di ricevere informazioni dall'ambiente circostante (o *mondo esterno*), di operare *trasformazioni* su queste informazioni, tese a dedurre o costruire nuove informazioni come *risultato* delle suddette trasformazioni, di comunicare al mondo esterno l'esito delle proprie elaborazioni.

Va subito detto che le informazioni da elaborare possono essere estremamente varie (grandezze fisiche come temperatura, pressione, peso, valori numerici, parole o frasi di testo, ecc.) e che una loro elaborazione automatica richiede di ricorrere a una opportuna **codifica**, cioè di:

- scegliere una grandezza fisica che funga da *rappresentante* dell'informazione da rappresentare;
- associare, a ogni possibile "aspetto" dell'informazione da rappresentare, un valore della grandezza fisica rappresentante che consenta di risalire all'informazione stessa.

La scelta del tipo di grandezza fisica rappresentante è guidata da considerazioni di natura prettamente tecnologica: dipende infatti dal livello di maturità industriale di una certa tecnologia realizzativa la possibilità di produrre, a costi ragionevoli, macchine utili. Un esempio per chiarire questo concetto è dato dalle calcolatrici da tavolo meccaniche diffuse diversi anni fa, nelle quali la grandezza fisica rappresentante le informazioni numeriche da elaborare era costituita dalla posizione relativa di vari ingranaggi metallici.

Negli ultimi anni, la scelta della grandezza fisica è stata sempre più indirizzata verso fenomeni di tipo elettrico (tensione e/o corrente elettrica) dal progressivo affermarsi della tecnologia di realizzazione dei circuiti elettronici.

Un dispositivo capace di comportarsi da interruttore comandato (cioè capace di aprire o chiudere un collegamento elettrico sulla base di uno stimolo elettrico) è oggi realizzabile — grazie alla tecnologia di integrazione su silicio, che realizza tali dispositivi trattando opportunamente superfici piane di silicio — in un'area avente dimensioni nell'ordine delle frazioni di *micron*, quindi con la possibilità di inserire strutture composte da milioni

di tali dispositivi in "pezzetti" (*chip*) di silicio di pochi millimetri quadrati. Inoltre, i fenomeni elettrici hanno comportamenti incomparabilmente più veloci dei fenomeni meccanici, grazie alla bassissima "inerzia" dei flussi di elettroni nei conduttori rispetto al movimento di parti meccaniche: se con organi meccanici si possono a fatica ottenere tempi di risposta dell'ordine del decimo di secondo (10^{-1} secondi), con dispositivi elettronici è possibile ottenere tempi di risposta dell'ordine del nanosecondo (10^{-9} secondi) quindi con un fattore di miglioramento pari a 10^8 (100 milioni di volte più velocemente).

Se si considera poi che il costo delle materie prime (il silicio e le altre sostanze necessarie a trattarne la superficie) è irrisorio, e che un impianto di realizzazione di circuiti integrati è in grado di produrre milioni di pezzi identici a costi singolarmente molto contenuti, si può facilmente intuire come tale tecnologia abbia preso il sopravvento su altre possibilità. Va comunque sottolineato come tale situazione non sia immutabile: non sono oggi rari i casi di ricorso a un fenomeno fisico — la luce — che consente mediante *macchine ottiche* di ottenere in particolari situazioni comportamenti ancora più vantaggiosi di quelli elettronici.

1.2 RAPPRESENTAZIONE ANALOGICA E RAPPRESENTAZIONE DIGITALE

Dato per acquisito il ricorso a rappresentazione elettronica dell'informazione, si tratta ora di vedere quale forma fare assumere alla funzione che associa a ogni "aspetto" dell'informazione da rappresentare un valore della grandezza elettrica usata come rappresentante.

Una prima forma è quella schematizzata dal diagramma di Figura 1.1, dove si adotta una corrispondenza biunivoca e lineare fra ogni aspetto dell'informazione da rappresentare (in questo caso, il valore di temperatura all'interno di un certo intervallo predefinito) e il corrispondente valore di tensione elettrica compreso fra 0 e V_{MAX}.

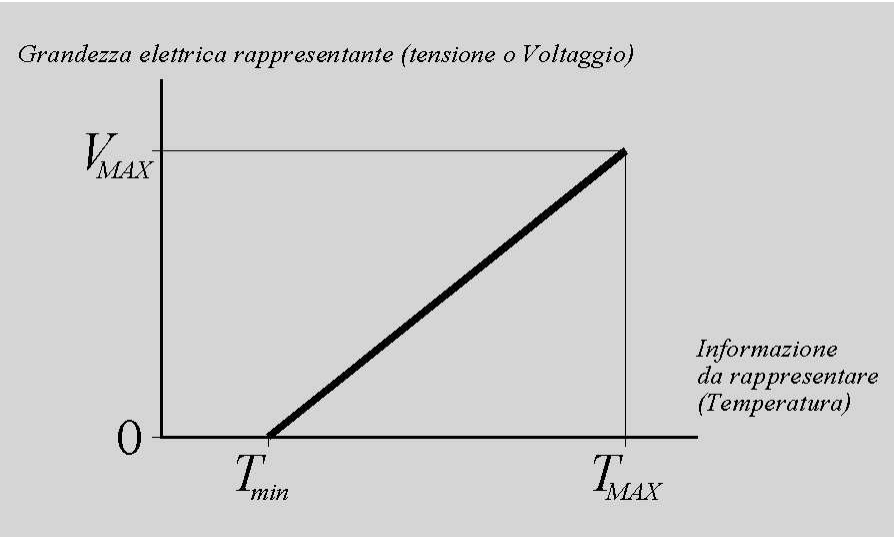


Figura 1.1 — Rappresentazione analogica dell'informazione

Questa rappresentazione è detta **analogica** poiché esiste una chiara analogia fra il comportamento dell'informazione e della sua rappresentazione: ogni possibile valore di temperatura è associato a un diverso valore di tensione, e ogni variazione di temperatura comporta una corrispondente, *analoga* variazione di tensione.

Numerosi sono i pregi di tale rappresentazione:

- ☺ è una rappresentazione fedele, nel senso che anche piccole variazioni dell'informazione da rappresentare vengono avvertite;
- ☺ è una rappresentazione intuitiva, poiché facilita la comprensione immediata della situazione legata all'informazione rappresentata; basti pensare ad esempio a un orologio a sfere, dove la posizione delle lancette consente un rapido riconoscimento dell'ora, o a un tachimetro a lancetta, dove la posizione viene immediatamente tradotta in indicazione della velocità che si sta tenendo.

Tali pregi, molto utili soprattutto nell'interazione fra uomo e macchina, sono però associati a numerosi difetti, che rendono tale rappresentazione poco adatta all'uso all'interno della macchina:

- ☹ l'elemento elettronico che gestisce la rappresentazione analogica dell'informazione ha — come sempre accade nel mondo reale — un comportamento "non ideale", dovuto a sia pur minimi difetti di realizzazione, a invecchiamento, a presenza di disturbi di natura elettrica, in generale a fenomeni che introducono un ***errore di rappresentazione***, cioè una differenza fra il valore assunto dalla grandezza elettrica e il valore che "avrebbe dovuto assumere" per rappresentare correttamente l'informazione;
- ☹ ogni elaborazione di rappresentazioni affette da errore comporta un progressivo "aumento dell'errore": basti pensare al fatto che, dovendo sommare due valori affetti da errore, l'errore risultante è la somma degli errori di partenza; ecco quindi che una rappresentazione poco affidabile mal si presta a elaborazioni complesse (cioè composte da numerose trasformazioni delle informazioni di partenza).

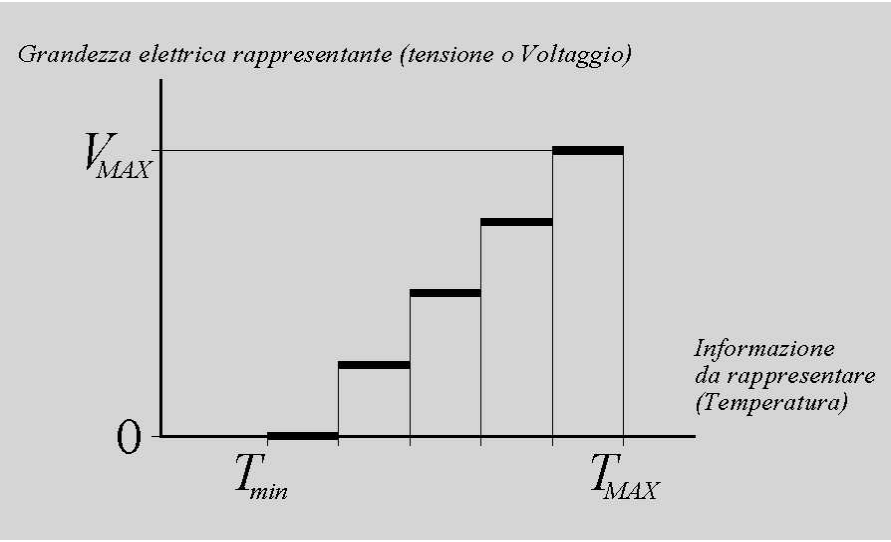


Figura 1.2 — Rappresentazione digitale dell'informazione

Per introdurre un comportamento più robusto e affidabile, si può ricorrere a una rappresentazione come quella schematizzata dal diagramma di Figura 1.2, dove la grandezza rappresentante può assumere solo un numero discreto di valori, ciascuno dei quali associato a un sottoinsieme dei possibili aspetti che l'informazione da rappresentare può assumere.

Questa rappresentazione è detta **digitale** perché definisce un numero discreto di "valori" o "cifre" (in inglese: *digit*) che la grandezza rappresentante può assumere. Pregi e difetti di tale rappresentazione sono naturalmente opposti rispetto alla rappresentazione analogica:

- ☹ la rappresentazione è poco fedele: variazioni anche ampie dell'informazione all'interno di un sottoinsieme associato a una stessa "cifra" non vengono avvertite, mentre piccole variazioni a cavallo del confine fra due sottoinsiemi comportano continui cambiamenti della "cifra" rappresentante;
- ☹ la rappresentazione non è altrettanto immediata, perché si tratta di interpretare il significato della cifra per risalire all'informazione codificata (ad esempio, un orologio digitale richiede di interpretare due numeri di due cifre ciascuno come ora e minuti per risalire all'informazione temporale associata);
- ☺ la rappresentazione è robusta, perché è decisamente più difficile che un difetto o un malfunzionamento ci facciano confondere due "cifre" diverse, data la loro significativa differenza;
- ☺ la rappresentazione si presta a elaborazioni anche complesse, poiché la probabilità decisamente inferiore di errori rende affidabile il risultato finale.

I due ultimi vantaggi sono ulteriormente enfatizzati dalla rappresentazione schematizzata in Figura 1.3, detta **rappresentazione (digitale) binaria** poiché usa solo due possibili valori della grandezza rappresentante, corrispondenti agli estremi 0 e V_{MAX} dell'intervallo utilizzato. Inoltre, il comportamento fisico dei dispositivi elettronici è tale da minimizzare il dispendio di energia quando i dispositivi stessi si trovano agli estremi dell'intervallo di possibili valori di funzionamento, a ulteriore vantaggio di un'impostazione binaria.

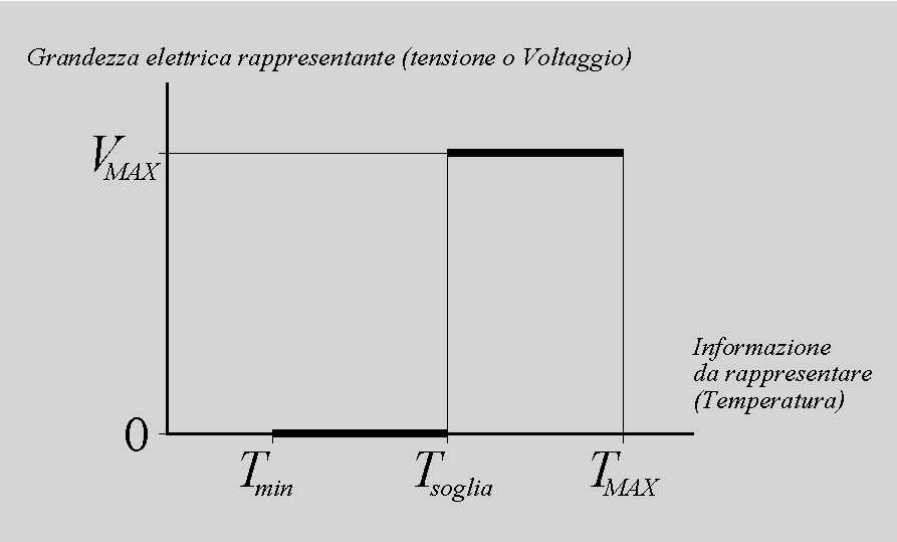


Figura 1.3 — Rappresentazione binaria dell'informazione

Con questa scelta, paghiamo l'estrema robustezza e affidabilità della rappresentazione con un potere rappresentativo troppo ridotto (nell'esempio della temperatura, possiamo solo sapere se siamo sopra o sotto il valore discriminante T_{soglia}). Per superare tale limite inaccettabile, l'unica via possibile è quella di associare — a ogni "aspetto" dell'informazione da rappresentare — ***un insieme di dispositivi elettronici rappresentanti***, in modo tale che l'informazione rappresentata dipenda non dallo stato di un solo rappresentante elettronico, ma dallo stato di un gruppo di rappresentanti, ognuno dei quali fornisce una delle "cifre" binarie (***bit: bi****nary digi****t***) della sequenza o *stringa* che rappresenta l'informazione suddetta.

1.3 ROAD MAP ALLA LETTURA DEL TESTO

A conseguenza di quanto sommariamente discusso sopra, questo testo è organizzato nel modo seguente.

In primo luogo, avendo stabilito che il modo elettronicamente più conveniente di rappresentare informazioni è quello di associare a tali informazioni opportune *stringhe di bit*, si tratta ora di definire i criteri di scelta di tali stringhe, dal momento che è completamente arbitrario e convenzionale il passaggio da un'informazione alla stringa che la rappresenta. Il **capitolo 2** si occupa proprio di esaminare i più diffusi metodi di codifica binaria di informazioni, e di introdurre le più comuni tecniche di elaborazione di queste informazioni.

Si tratta a questo punto di individuare metodologie di progetto che, partendo dalle codifiche binarie sopra citate e dalle necessità di elaborazione che si vuole realizzare, permettano di ottenere la macchina elettronica desiderata.

A tale scopo, si discute nel **capitolo 3** un formalismo algebrico (logica booleana o algebra di commutazione) che permette di trattare in modo simbolico grandezze binarie, utilizzando a tale scopo pochi semplici operatori di facile realizzabilità da un punto di vista elettronico.

Avendo a disposizione metodi di codifica di informazioni qualsiasi e metodi di descrizione simbolica delle operazioni, si tratta infine di individuare tecniche di progetto della macchina elettronica (o ***rete logica***, cioè rete elettrica composta da elementi circuitali che si comportano secondo le regole della logica booleana) che svolga il compito richiesto; tali tecniche sono oggetto dei **capitoli 4 e 5** dedicati rispettivamente alle *reti combinatorie* (macchine il cui comportamento dipende da ciò che istante per istante si presenta ai loro ingressi) e alle *reti sequenziali* (macchine il cui comportamento dipende dalle sequenze temporali di ingressi, quindi dalla loro "storia").

2 CODIFICA DELL'INFORMAZIONE

Lo scopo di questa parte è un sommario dei concetti di base della codifica binaria delle informazioni, importante per la natura fisica dei dispositivi elettronici integrati, che presentano vantaggi di immunità al rumore, di minore consumo di energia elettrica e di migliore impaccabilità se utilizzati con valori quantizzati.

Valgono le seguenti relazioni numeriche fondamentali:

1. con n bit, si rappresentano $N = 2^n$ diverse configurazioni (stringhe) di bit;
2. per rappresentare M diversi "aspetti" di un'informazione, servono $n = \log_2\lceil M \rceil$, dove $\lceil M \rceil$ rappresenta M arrotondato all'intero superiore.

2.1 NUMERAZIONE POSIZIONALE

Un numero N in una generica base b viene rappresentato come:

$$(N)_b = p_n b^n + p_{n-1} b^{n-1} + \ldots + p_1 b^1 + p_0 b^0 + p_{-1} b^{-1} + \ldots + p_{-m} b^{-m} = \sum_{i=-m}^{n} p_i b^i$$

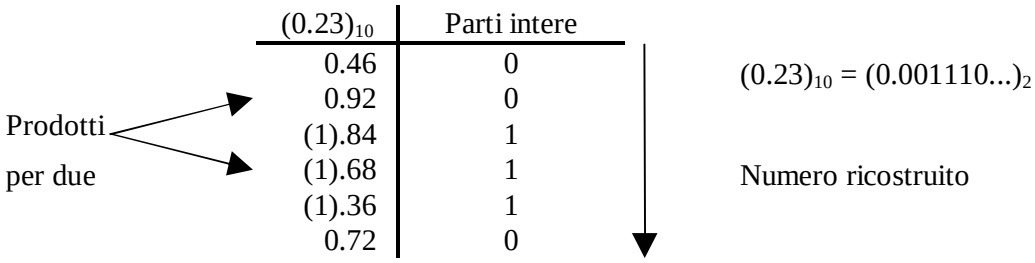
Per effettuare la conversione da base b a base 10 si procede in modo diretto, con il **metodo polinomiale**:

$$(10010.1)_2 = 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0 + 1\times2^{-1} = 16 + 2 + 0.5 = 18.5$$

Per effettuare la conversione da base 10 a base b, si utilizza un **metodo iterativo**:

A. per la parte intera, si divide $(N)_{10}$ per b, il resto costituisce il bit meno significativo di $(N)_b$ Il metodo viene iterato finché si ottiene quoziente = 0

B. per la parte frazionaria, si moltiplica $(N)_{10}$ per b, e si verifica se il risultato è 1. Se sì, il bit più significativo di $(N)_b$ vale uno, altrimenti è la parte intera (1 nel caso b=2) del prodotto $(N)_{10}b$. Si elimina tale parte intera e si itera il procedimento, *che può non terminare.*



2.1.1 Operazioni Aritmetiche

Somma

Si procede secondo le regole dell'aritmetica decimale.

Sottrazione

Si procede secondo le regole dell'aritmetica decimale.

Complemento

È un metodo molto utile per uniformare da un punto di vista *algoritmico* (cioè in termini di operazioni da svolgere) l'esecuzione di somme e sottrazioni.

Viene applicato come esempio a *parti frazionarie* (cioè numeri < 1). In tal caso, il complemento a 2 di una parte frazionaria B è definito da:

$$^2B = (2 - B)_{10} = (10 - B)_2 \qquad \Rightarrow \qquad ^2(0.1101) = 10.0000 - 0.1101 = 1.0011$$

Esiste un metodo più semplice per effettuare la complementazione, se si tiene conto che:

$$(10)_2 = 10.0000 = 1.1111 + 0.0001$$

da cui deriva che:

$$10.0000 - 0.1101 = 1.1111 - 0.1101 + 0.0001$$

l'operazione 1.1111 - 0.1101 costituisce il *complemento a 1* del numero B, e si effettua semplicemente sostituendo ogni 0 di B con 1 e viceversa. Il complemento a 2 si ottiene dunque invertendo tutti i bit di B e sommando poi un 1 al meno significativo.

L'utilità del complemento a 2 sta nel fatto che il risultato della sottrazione A-B può essere ottenuto effettuando la somma $A + {}^2B = (10 + A\text{-}B)_2$. Si distinguono due casi:

$A\text{-}B \geq 0$ in questo caso, è 10 più la parte frazionaria positiva A-B, ricavabile direttamente eliminando l'1 più significativo della rappresentazione di $A + {}^2B$.

$A\text{-}B < 0$ in questo caso, $A + {}^2B = (10 - | A\text{-}B |)_2$ che corrisponde a $^2(A\text{-}B)$. Nell'ipotesi di assumere che una parte frazionaria negativa venga rappresentata dal complemento a 2 del suo valore assoluto, anche questo caso viene risolto dall'adozione del complemento a 2.

Si noti che l'ultima ipotesi sulla codifica di parti frazionarie in complemento a 2 fa sì che sia anche molto facile discriminare fra parti frazionarie positive e negative: queste ultime risultano infatti > 1, cioè hanno il bit più significativo (a sinistra della virgola) = 1.

L'adozione del complemento a 2 richiede naturalmente di gestire opportunamente le operazioni di somma. In particolare, la somma di parti frazionarie con uguale segno porta a codifiche prive di significato qualora il valore assoluto di tale somma sia ≥ 1. Si deve quindi verificare che la somma di due parti frazionarie positive non dia luogo a una codifica negativa (generazione di *overflow*) e allo stesso modo che la somma di due parti frazionarie negative non dia luogo a una codifica positiva (generazione di *underflow*).

Una buona rappresentazione grafica della codifica in complemento a 2 può essere realizzata con una struttura circolare, come quella riportata in Figura 2.1.

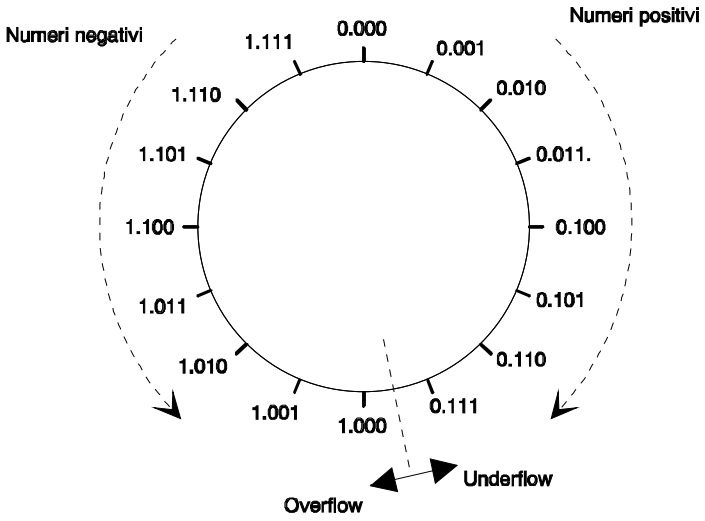


Figura 2.1 — Rappresentazione circolare dei numeri in complemento a 2

Il discorso si può facilmente estendere a numeri (interi o frazionari) rappresentabili con un predefinito numero di bit.

Shift

una traslazione verso sinistra (destra) di k bit corrisponde a una moltiplicazione (divisione) per b^k.

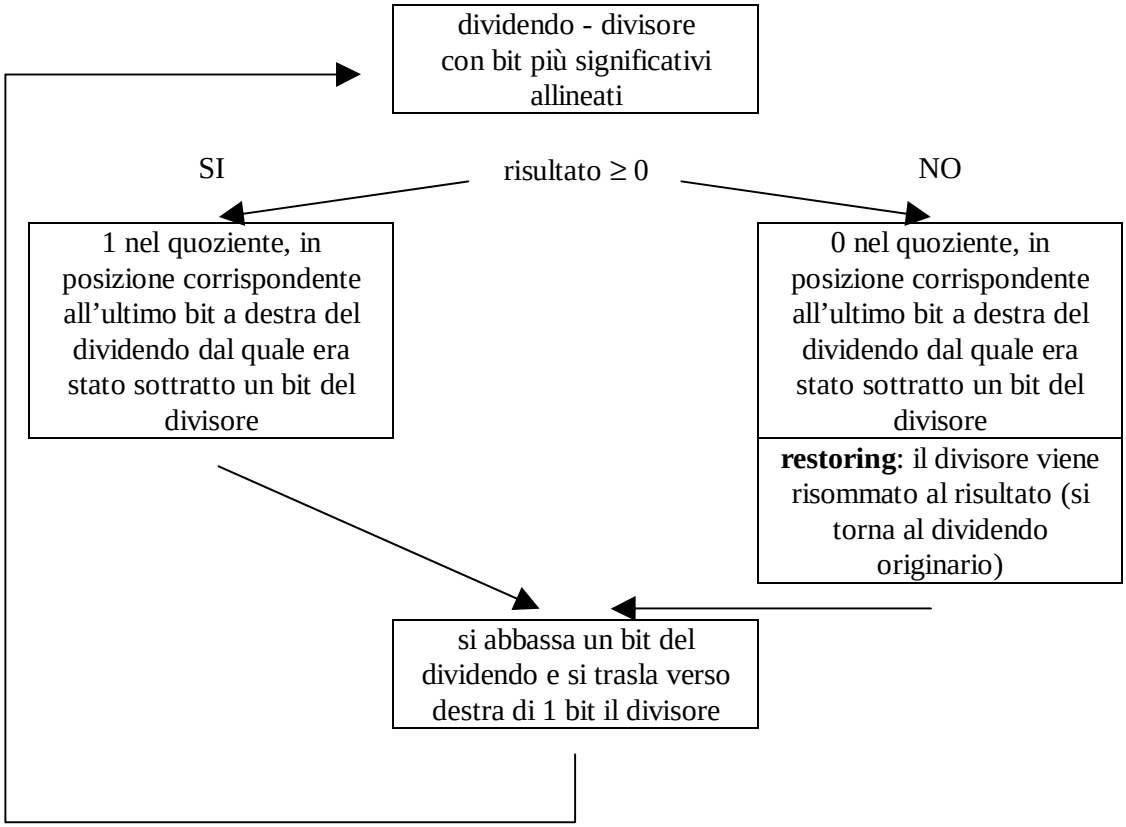
Moltiplicazione

Si procede secondo le regole dell'aritmetica decimale. Tecniche più efficaci verranno analizzate in seguito.

Divisione

Le regole dell'aritmetica decimale si basano su un meccanismo di *trial and error*, che non è particolarmente facile da realizzare in modo automatico. Si ricorre a due metodi iterativi.

Restoring division



Esempio: 1100 / 1111

		1	1	0	0				
sottrazione		1	1	1	1				
negativo (bit A)	-	0	0	1	1				ottenuto come -(1111-1100)
restore		1	1	0	0	0			abbassa una cifra
sottrazione			1	1	1	1			
positivo (bit B)			1	0	0	1	0		abbassa una cifra
sottrazione				1	1	1	1		
positivo (bit C)				0	0	1	1	0	abbassa una cifra
sottrazione					1	1	1	1	
negativo (bit D)				-	1	0	0	1	0 abbassa una cifra
restore					0	1	1	0	
RISULTATO		0.	1	1	0	…			
		A	B	C	D				

Non restoring division

Invece di risommare il divisore *Y* per poi sottrarlo traslato al dividendo *X*, si somma direttamente il divisore traslato, dal momento che uno shift equivale a dividere per 2, quindi:

$$X + Y - \tfrac{1}{2} Y = X + \tfrac{1}{2} Y$$

2.2 CODIFICA BINARIA DI NUMERI DECIMALI

Valgono le seguenti considerazioni:

- servono almeno 4 bit, dal momento che devo rappresentare 10 cifre decimali, quindi servono almeno 10 configurazioni diverse;
- con 4 bit ho a disposizione $2^4 = 16$ configurazioni, quindi 6 sono inutili;
- la scelta delle 10 configurazioni utili è una qualsiasi delle disposizioni $D_{16,10}$ cioè uno qualsiasi dei modi possibili di definire una sequenza di 10 simboli diversi presi da un insieme di 16 simboli. Il numero di possibili codici è dunque:

$$D_{16,10} = C_{16,10} \times P_{10} = \binom{16}{10} \times 10! = \frac{16!}{10! \times (16\text{-}10)!} \times 10! = \frac{16!}{6!} \cong 2.9 \times 10^{10}$$

Di tutti questi codici, citiamo i più conosciuti (riassunti in Tabella 2.1):

8421 prende in ordine i primi 10 codici binari. È un *codice pesato* perché ogni bit ha un peso associato alla sua posizione. È noto anche come **codice BCD** (da Binary Coded Decimals);

84-2-1 è un *codice pesato*, *autocomplementante* nel senso che il complemento a 1 del codice (tutti i bit invertiti) corrisponde al complemento a 9 della cifra decimale associata;

eccesso-3 è un codice non pesato, *autocomplementante*, ottenuto sommando 3 al codice 8421. Per le sue proprietà, si utilizzava nei primi calcolatori.

Cifra decimale	8421	84-2-1	eccesso-3
0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 1 1	0 1 0 0
2	0 0 1 0	0 1 1 0	0 1 0 1
3	0 0 1 1	0 1 0 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 0 1 0	1 0 0 1
7	0 1 1 1	1 0 0 1	1 0 1 0
8	1 0 0 0	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 0 0

Tabella 2.1 — Codici per la codifica binaria di numeri decimali

2.3 RAPPRESENTAZIONE DI INSIEMI NUMERABILI

Date le precedenti relazioni, è immediato calcolare il numero di bit necessari per rappresentare una certa informazione in base ai diversi valori che tale informazione può assumere. Tuttavia, la corrispondenza fra ciascun valore dell'informazione e la

corrispondente codifica binaria è assolutamente arbitraria, e la sua scelta — che deve peraltro essere poi mantenuta congruente da tutti gli utilizzatori di tale informazione — può privilegiare altri parametri.

Ad esempio, volendo codificare con numero binari i 7 colori dello spettro luminoso più la mancanza di colori (ROSSO, ARANCIO, GIALLO, VERDE, AZZURRO, INDACO, VIOLETTO, NERO) servono $\log_2 8 = 3$ bit. Tuttavia, i modi in cui posso associare le configurazioni di 3 bit ai colori sono tutte le possibili permutazioni di 8 valori, cioè $8! = 40320$. Esiste però un modo, riportato in Tabella 2.2, nel quale a valori crescenti della configurazione binaria corrispondono frequenze crescenti della radiazione elettromagnetica associata al colore, ovvero indicazione della posizione del colore nello spettro. Questa scelta può essere considerata ottimale perché mantiene traccia di ulteriori proprietà dell'informazione codificata.

Informazione	**Codifica**
NERO	000
ROSSO	001
ARANCIO	010
GIALLO	011
VERDE	100
AZZURRO	101
INDACO	110
VIOLETTO	111

Tabella 2.2 — Codifica ottimale dei colori dello spettro

In modo analogo, esistono vari modi di codificare i caratteri alfanumerici (lettere dell'alfabeto, cifre decimali, simboli di interpunzione, ecc.). Fra questi, il più noto e diffuso è lo standard ***ASCII*** (***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*) basato su stringhe di 7 bit scelte in modo tale da mantenere la corrispondenza fra valore binario della codifica e posizione alfabetica della lettera.

2.4 CODICI A DISTANZA DI HAMMING UNITARIA

Per ***distanza di Hamming*** fra due configurazioni (stringhe) binarie di uguale lunghezza, si intende il numero di bit che assumono valore diverso nelle due configurazioni. I codici a distanza di Hamming unitaria sono basati su codifiche a distanza unitaria per simboli consecutivi nell'alfabeto rappresentato, quindi su codifiche dei simboli consecutivi che si ottengono l'una dall'altra cambiando il valore di un solo bit.

Sono utili ad esempio per operazioni di conversione Analogico-Digitale, onde evitare errori di allineamento (come accade ad esempio negli *encoder* assoluti, realizzati con batterie di LED e fototransistori separati da un disco trasparente opacizzato in modo selettivo).

Tipico esempio il **codice Gray**, ottenuto per "riflessioni" successive con premessa di 0 all'originale e di 1 al riflesso:

simbolo	codice			
0	0	0	0	
1	0	0	1	riflessione
2	0	1	1	
3	0	1	0	riflessione
4	1	1	0	
5	1	1	1	
6	1	0	1	
7	1	0	0	

2.5 CODICI RILEVATORI E CORRETTORI DI ERRORI

Il problema è rappresentabile come in Figura 2.2: $\exists$ probabilità $\neq 0$ che $C_R \neq C_T$

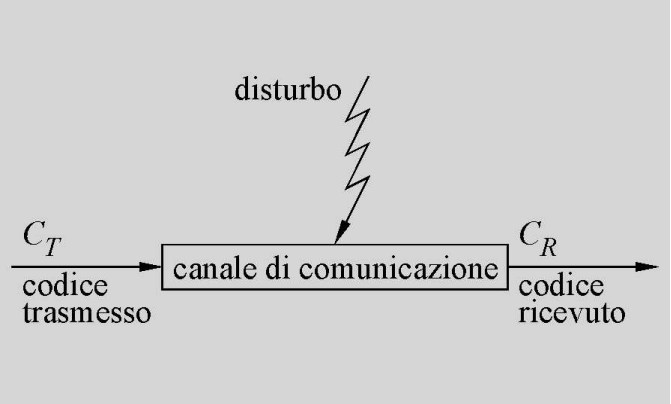


Figura 2.2 — Cause di errore nella trasmissione di configurazioni binarie

Se l'alfabeto dei C_T è **completo** (ogni codice è associato a un simbolo significativo, quindi è trasmissibile) non c'è possibilità di individuare errori di comunicazione a livello di singolo simbolo (si può ricorrere ad analisi contestuali, ma ad altro livello logico e di complessità).

Se l'alfabeto dei C_T è **incompleto** (codifiche ridondanti, che introducono naturalmente maggiore complessità, maggiori costi e maggiore vulnerabilità nei dispositivi) come indicato ad esempio in Figura 2.3, si verifica quanto segue.

Il trasmettitore — per definizione "onesto" — trasmette solo codici $\in C_T$

Se il ricevitore riceve un codice $\in C_R$ ma $\notin C_T$ può affermare con certezza che si è verificato un errore di trasmissione.

Se il ricevitore riceve un codice $\in C_T$ può solo "sperare" che coincida con quello originariamente trasmesso.

La situazione può essere rappresentata come in Figura 2.4:

Problemi:

1. determinare il grado di ridondanza necessario per ottenere un codice che gestisca gli errori attesi (ad esempio, gli errori di campionamento in una forma d'onda a livelli in presenza di *rumore bianco*);

2. scegliere C_T in modo da minimizzare la probabilità che si verifichi l'evento ③, sempre nell'ipotesi che ① >> ② + ③

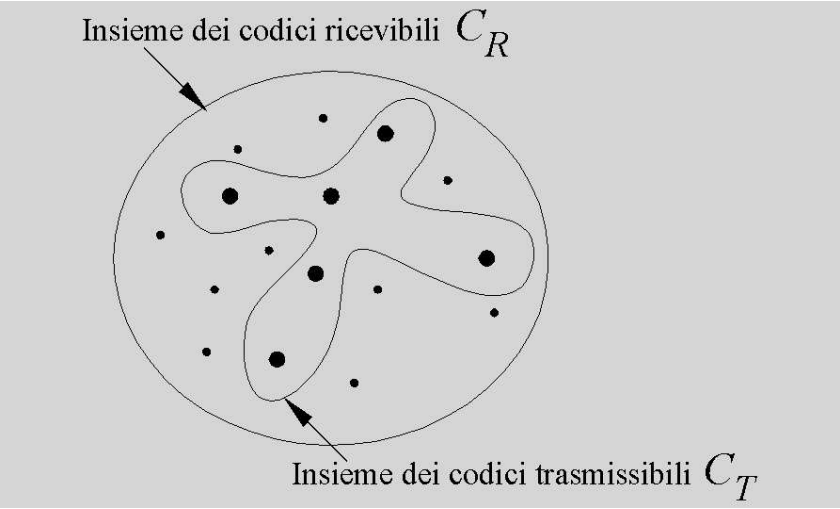


Figura 2.3 — Relazione fra codici trasmissibili e codici ricevibili

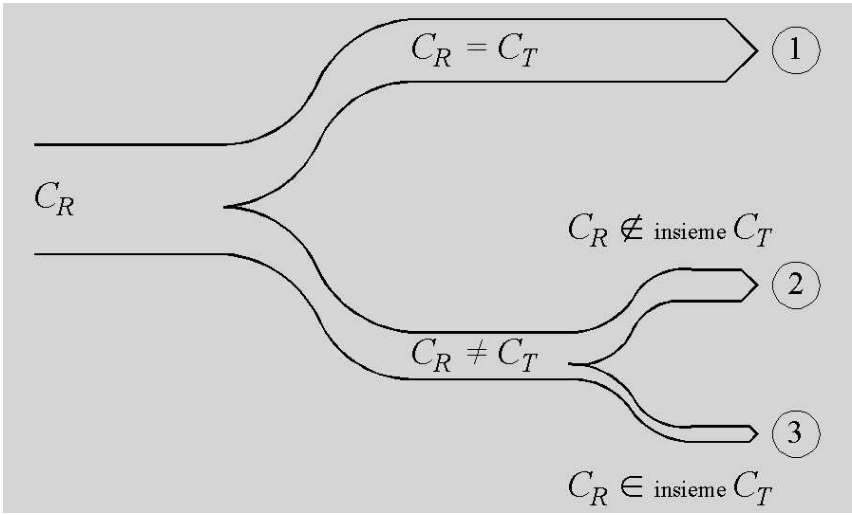


Figura 2.4 — Possibile comportamento del ricevitore

2.5.1 Esempi di codici basati sulla distanza di Hamming

Bit di parità

Si basa su un bit di ridondanza, scelto in modo da rendere pari (parità pari) o dispari (parità dispari) il numero di 1 presenti nei codici trasmessi C_T

In questo modo, i C_T hanno distanza di Hamming minima pari a 2, e consentono pertanto di rilevare *errori dispari* (quindi di garantire la **rilevazione di errori singoli**). Il codice ha dunque particolare efficacia se l'andamento della probabilità di errore in funzione del numero di bit errati è del tipo riportato in Figura 2.5.

Codice di Hamming

È un codice che garantisce una distanza di Hamming minima pari a 3 fra due qualsiasi codici C_T e consente pertanto una **correzione** dell'errore basata sull'ipotesi che gli errori più probabili siano errori singoli, e che pertanto sia ragionevole ricondurre al codice trasmissibile più vicino un eventuale codice ricevuto $\notin C_T$

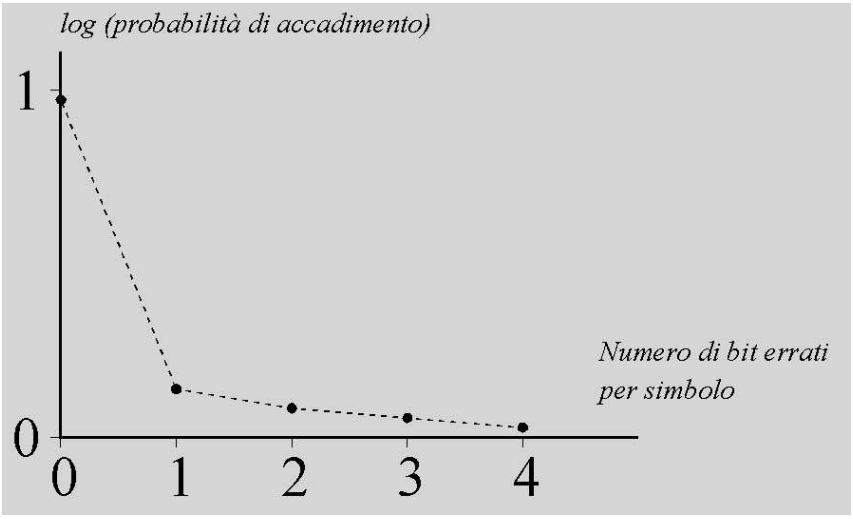


Figura 2.5 — Andamento tipico della probabilità di errore

La valutazione del ***livello di ridondanza*** (cioè del numero di bit aggiuntivi necessari per consentire la realizzazione del codice di Hamming) si basa sulle seguenti considerazioni, dove sia n il numero di bit nominali (necessari a codificare i simboli dell'alfabeto utilizzato), r il numero di bit ridondanti e $c = n + r$ il numero di bit totali del codice:

- ogni "nuvola" attorno a un elemento $\in C_T$ deve essere costituita da tutti gli elementi $\in C_R - C_T$ che si trovano a distanza di Hamming unitaria dall'elemento considerato, quindi ogni nuvola contiene $c + 1$ elementi (l'elemento $\in C_T$ più i c elementi $\in C_R - C_T$ ottenuti cambiando a turno uno dei bit dell'elemento di partenza);
- le "nuvole" devono essere disgiunte, altrimenti avremmo due elementi $\in C_R$ che distano fra loro meno di 3 bit;
- serve una "nuvola" per ogni elemento $\in C_T$ quindi servono 2^n nuvole;
- si hanno a disposizione 2^c codici.

Per soddisfare le condizioni sopra elencate, deve pertanto valere la disequazione:

$$2^n (c + 1) \leq 2^c$$

ovvero:

$$2^n (n + r + 1) \leq 2^n \times 2^r$$

da cui:

$$n + r + 1 \leq 2^r$$

La ***costruzione del codice*** si basa invece sul seguente algoritmo:

- si numerano i c bit di C_T da 1 a c, con $c_1 = MSB$
- si inseriscono i bit di controllo (bit ridondanti) in posizione $i = 2^k$ dove $k = 0..r\text{-}1$
- si vuole ottenere dal codice una *parola di posizione*, costituita da una configurazione binaria che dica la posizione dell'eventuale bit errato nel codice. Facendo l'esempio del codice di Hamming applicato al codice BCD (servono 4 bit nominali più 3 bit ridondanti) tale risposta è definita in base alla Tabella 2.3:

p_3	p_2	p_1	**significato**
0	0	0	nessun errore
0	0	1	errore in $c_1 \equiv r_1$
0	1	0	errore in $c_2 \equiv r_2$
0	1	1	errore in $c_3 \equiv n_1$
1	0	0	errore in $c_4 \equiv r_3$
1	0	1	errore in $c_5 \equiv n_2$
1	1	0	errore in $c_6 \equiv n_3$
1	1	1	errore in $c_7 \equiv n_4$

Tabella 2.3 — Sindrome di errore del codice di Hamming

Nell'ipotesi di associare i bit della parola di posizione ai corrispondenti bit ridondanti nel codice, si possono notare le seguenti relazioni:

- p_1 vale 1 se ci sono errori in (c_1, c_3, c_5, c_7) quindi r_1 può essere calcolato come il bit di parità del gruppo (c_1, c_3, c_5, c_7): un errore di parità in tale gruppo viene segnalato da r_1 e forza a 1 il valore di p_1
- p_2 vale 1 se ci sono errori in (c_2, c_3, c_6, c_7) quindi r_2 può essere calcolato come il bit di parità del gruppo (c_2, c_3, c_6, c_7): un errore di parità in tale gruppo viene segnalato da r_2 e forza a 1 il valore di p_2
- p_3 vale 1 se ci sono errori in (c_4, c_5, c_6, c_7) quindi r_3 può essere calcolato come il bit di parità del gruppo (c_4, c_5, c_6, c_7): un errore di parità in tale gruppo viene segnalato da r_3 e forza a 1 il valore di p_3

Un esempio di costruzione di codice di Hamming è riportato nella Tabella 2.4:

$c_1 \equiv r_1$	$c_2 \equiv r_2$	$c_3 \equiv n_1$	$c_4 \equiv r_3$	$c_5 \equiv n_2$	$c_6 \equiv n_3$	$c_7 \equiv n_4$
		0		1	0	0
1						
	0					
			1			
1	0	0	1	1	0	0

Tabella 2.4 — Esempio di codifica di Hamming

È facile dimostrare che la posizione di un eventuale errore viene individuata analizzando quali bit di parità r_i risultano sbagliati, e assegnando il valore 1 al corrispondente bit della parola di posizione.

Codice a Ridondanza Ciclica (CRC)

È basato su elaborazione aritmetica invece di distanza di Hamming per minimizzare la probabiità che un errore durante la comunicazione dia luogo in ricezione a un codice accettabile.

È un codice *rivelatore* di errori, adatto per risolvere situazioni di errore anche diverse da quelle affrontate dal codice Hamming (errori a burst, tratabili ma con difficoltà mediante trasmissione in colonna).

Si basa sulla definizione dei seguenti polinomi:

$M(x)$ polinomio *messaggio*, in cui i coefficienti della variabile x sono i valori dei bit da trasmettere;

$G(x)$ polinomio di grado r, *generatore* del codice, noto sia al trasmettitore che al ricevitore;

Si effettua — in campo di Galois di 2, cioè con operazioni modulo 2 senza riporti — la seguente operazione:

$$\frac{M(x) \times x^r}{G(x)}$$

L'operazione porta al calcolo di un polimonio *quoziente* $Q(x)$ e di un polinomio *resto* $R(x)$. Si effettua la trasmissione del polinomio *trasmesso*:

$$T(x) = M(x) \times x^r - R(x)$$

Si noti che questo polinomio è divisibile esattamente per $G(x)$ e che ha anche la proprietà di consentire l'immediata ricostruzione di $M(x)$, dal momento che $R(x)$ ha grado r-1, quindi va ad occupare i bit meno significativi di $M(x) \times x^r$ senza alterare i bit di $M(x)$.

Qualora durante la comunicazione avvengano errori, il polinomio *ricevuto* può essere espresso come:

$$T'(x) = T(x) + E(x)$$

Dove $E(x)$ è il polinomio *errore*, i cui coefficienti sono 1 in corrispondenza delle posizioni di $T(x)$ in cui si sono verificati errori. Dal momento che:

$$\frac{T'(x)}{G(x)} = \frac{T(x) + E(x)}{G(x)} = \frac{T(x)}{G(x)} + \frac{E(x)}{G(x)}$$

e che la divisione

$$\frac{T(x)}{G(x)}$$

dà per definizione resto zero, la costruzione del codice si riconduce a trovare un $G(x)$ tale da massimizzare la probabilità che il resto della divisione:

$$\frac{E(x)}{G(x)}$$

sia **diverso** da zero per i polinomi di errore interessanti, cioè per i casi di errore ritenuti più probabili.

Si possono dimostrare le seguenti proprietà:

errori singoli $E(x) = x^j$

rilevabili se $G(x)$ ha il termine noto (cioè il bit meno significativo ≠ 0);

errori doppi $E(x) = x^{k+j} + x^j = x^j \times (x^k + 1)$

non esistono regole generali: si devono costruire per tentativi dei $G(x)$ opportuni (ad esempio, $G(x) = x^{15} + x^{14} + 1$ rileva tutti gli errori doppi fino a $k = 32768$);

errori dispari $E(x) = x^j \times (x^i + x^k + 1)$

rilevabili se $G(x)$ contiene fra i suoi fattori primi il binomio $(x + 1)$;

errori a burst

rilevabili se hanno grado $< r$ (cioè se la loro lunghezza è inferiore a quella di $G(x)$).

Esistono alcuni polinomi predefiniti, utilizzati in varie applicazioni pratiche. Sono degni di nota i seguenti:

CRC-16 = $x^{16} + x^{15} + x^2 + 1$

CRC-CCITT = $x^{16} + x^{12} + x^5 + 1$

Il seguente esempio mostra la costruzione del codice CRC.

$M(x) = 101101110$

$G(x) = x^3 + x + 1 = 1011$

$r = 3$

$M(x) \times x^r = 101101110\ 000$

1	0	1	1	0	1	1	1	0	0	0	0	1	0	1	1
1	0	1	1									1			
	0	0	0	0								0			
		0	0	0	1							0			
			0	0	1	1						0			
				0	1	1	1					0			
					1	1	1	0				1			
						1	0	1	0			1			
							0	0	1	0		0			
								0	1	0	0	0			
									1	0	0				

$R(x) = 100$

$T(x) = 101101110100$

Verifica di corretta ricezione

1	0	1	1	0	1	1	1	0	1	0	0	1	0	1	1
1	0	1	1									1			
	0	0	0	0								0			
		0	0	0	1							0			
			0	0	1	1						0			
				0	1	1	1					0			
					1	1	1	0				1			
						1	0	1	1			1			
							0	0	0	0		0			
								0	0	0	0	0			
									0	0	0				

Errore singolo

1	0	1	1	0	1	**0**	1	0	1	0	0	1	0	1	1
1	0	1	1									1			
	0	0	0	0								0			
		0	0	0	1							0			
			0	0	1	0						0			
				0	1	0	1					0			
					1	0	1	0				1			
						0	0	1	1			0			
							0	1	1	0		0			
								1	1	0	0	1			
									1	1	1				

Errore a burst (3 bit)

1	0	1	1	0	**0**	**0**	**0**	0	1	0	0	1	0	1	1
1	0	1	1									1			
	0	0	0	0								0			
		0	0	0	0							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			
								0	1	0	0	0			
									1	0	0				

Un metodo ulteriore è la cosiddetta **checksum**, costituita da una somma su *n* bit (somma modulo 2^n) di tutte le parole di *n* bit nelle quali si può scomporre il messaggio.

3 ALGEBRA DI COMMUTAZIONE

Lo scopo di questa algebra — nota anche come *logica booleana* dal matematico inglese Boole — è fornire strumenti per la modellizzazione e la trattazione di dispositivi funzionanti con **segnali a due valori**, cioè dispositivi basati su logica binaria. Mediante l'algebra di commutazione, è possibile effettuare una *progettazione logica* dei dispositivi, trascurando il modo in cui verranno fisicamente realizzati, quindi anche il tipo (tensione, corrente, ma anche posizione meccanica di un commutatore, ecc.) e il valore quantitativo di grandezza fisica che sarà utilizzata per rappresentare tali segnali.

Nel seguito del capitolo, assumiamo *logica positiva*, cioè utilizziamo i simboli 0 e 1 per rappresentare — rispettivamente — il valore inferiore e il valore superiore dei segnali in gioco.

3.1 POSTULATI DELL'ALGEBRA DI COMMUTAZIONE

Definiamo come X una qualsiasi variabile binaria, cioè la rappresentazione algebrica di uno dei segnali che trattiamo nelle reti logiche. Il primo postulato afferma semplicemente che trattiamo logica binaria.

(P1) $X = 0$ iff $X \neq 1$ (P1') $X = 1$ iff $X \neq 0$

Definiamo ora come $\overline{X}$ il segnale avente valore opposto di quello assunto dalla variabile X. Si noti che X e $\overline{X}$ non sono due variabili diverse: ci si riferisce a questi due simboli come a due **letterali**, definendo dunque come letterale una variabile logica associata o meno al simbolo di inversione di valore. X e $\overline{X}$ sono dunque due diversi letterali di una stessa variabile. Il legame fra i letterali di una stessa variabile è definito dal secondo postulato:

(P2) if $X = 0$ then $\overline{X} = 1$ (P2') if $X = 1$ then $\overline{X} = 0$

Sulle variabili logiche così introdotte, si possono definire le operazioni di **inversione** (o negazione, o **NOT**, indicata dal tratto sopra il nome della variabile), di **prodotto logico** di due variabili (o operazione **AND**, indicata dal simbolo ·, a volte omesso come nell'algebra comune), di **somma logica** di due variabili (o operazione **OR**, indicata dal simbolo +). I seguenti tre postulati definiscono le operazioni di prodotto e somma logici:

(P3) $0 \cdot 0 = 0$ (P3') $1 + 1 = 1$

(P4) $1 \cdot 1 = 1$ (P4') $0 + 0 = 0$

(P5) $1 \cdot 0 = 0 \cdot 1 = 0$ (P5') $0 + 1 = 1 + 0 = 1$

Si noti che i postulati sono stati definiti sempre a coppie, il cui secondo membro è stato indicato da un apostrofo ('). Si noti anche che ciascun secondo membro è ottenibile dal primo mediante scambio dei valori 0 e 1 e scambio degli operatori · e +. Questo è un

esempio del **principio di dualità**, vero per tutti i teoremi dell'algebra di commutazione poiché risulta vero per i suoi postulati.

3.2 TEOREMI DELL'ALGEBRA DI COMMUTAZIONE

Partendo dai postulati precedentemente introdotti, si possono dimostrare i seguenti teroremi. Dal momento che l'algebra di commutazione opera su variabili a soli due valori, è possibile dimostrare quasi tutti tali teoremi mediante *induzione perfetta*, cioè verificandone la validità per tutte le possibili combinazioni di valori assegnati alle variabili logiche che compaiono nei teoremi stessi.

3.2.1 Teoremi a una variabile

(T1)	$X + 0 = X$	(T1')	$X \cdot 1 = X$	(Identità)
(T2)	$X + 1 = 1$	(T2')	$X \cdot 0 = 0$	(Elementi nulli)
(T3)	$X + X = X$	(T3')	$X \cdot X = X$	(Idempotenza)
(T4)	$\overline{(\overline{X})} = X$			(Involuzione)
(T5)	$X + \overline{X} = 1$	(T5')	$X \cdot \overline{X} = 0$	(Complementi)

3.2.2 Teoremi a due o tre variabili

(T6)	$X + Y = Y + X$	(T6')	$X \cdot Y = Y \cdot X$	(Commutatività)
(T7)	$X + X \cdot Y = X$	(T7')	$X \cdot (X + Y) = X$	(Assorbimento)
(T8)	$(X + \overline{Y}) \cdot Y = X \cdot Y$	(T8')	$X \cdot \overline{Y} + Y = X + Y$	
(T9)	$(X + Y) + Z = X + (Y + Z) = X + Y + Z$			
(T9')	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$			(Associatività)
(T10)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$			
(T10')	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$			(Distributività)
(T11)	$(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \cdot (\overline{X} + Z)$			
(T11')	$X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$			(Consenso)
(T12)	$(X + Y) \cdot (\overline{X} + Z) = X \cdot Z + \overline{X} \cdot Y$			

3.2.3 Teoremi a n variabili

(T13) $\overline{(X_1 + X_2 + ... + X_n)} = \overline{X_1} \cdot \overline{X_2} \cdot ... \cdot \overline{X_n}$

(T13') $\overline{(X_1 \cdot X_2 \cdot ... \cdot X_n)} = \overline{X_1} + \overline{X_2} + ... + \overline{X_n}$ (De Morgan)

(T14) $\overline{f(X_1, X_2, ... X_n, +, \cdot)} = f(\overline{X_1}, \overline{X_2}, ... \overline{X_n}, \cdot, +)$ (T13 generalizz.)

(T15) $f(X_1, X_2, ... X_n) = X_1 \cdot f(1, X_2, ... X_n) + \overline{X_1} \cdot f(0, X_2, ... X_n)$ (Espansione)

(T15') $f(X_1, X_2, ... X_n) = [X_1 + f(0, X_2, ... X_n)] \cdot [\overline{X_1} + f(1, X_2, ... X_n)]$

3.3 PERCHÉ L'ALGEBRA DI COMMUTAZIONE

Il fatto che l'algebra di commutazione consenta una trattazione simbolica di grandezze che possono assumere solo due valori — quindi grandezze che possono facilmente essere associate ai dispositivi elettrici di un circuito digitale binario (dal momento che tali dispositivi sono realizzati in modo tale da assumere solo due stati elettrici estremi) — non sarebbe di per sè sufficiente a giustificarne l'uso nella trattazione della progettazione delle reti logiche.

Una prima considerazione da fare è la seguente. Abbiamo visto nel capitolo precedente l'uso di codifiche binarie per informazioni (ad esempio, i numeri interi) che richiedono elaborazioni (come le somme) diverse da quelle realizzabili con gli operatori NOT, AND e OR: può l'algebra di commutazione consentire di realizzare tali operazioni? Non abbiamo qui la pretesa di essere esaustivi, ma ci limitiamo a un semplicissimo esempio per indicare la strada concettuale che seguiremo.

Supponiamo di voler realizzare la somma aritmetica di due numeri ciascuno codificato mediante un solo bit. Il valore di ciascun addendo può quindi essere 0 oppure 1, e la somma può assumere i valori 0, 1 o 2 a seconda che entrambi gli addendi valgano 0, che uno solo dei due addendi valga 1, che entrambi gli addendi valgano 1. Tuttavia, questo ultimo caso (entrambi gli addendi che valgono 1) è un caso particolare, nel quale il risultato non può più essere codificato con lo stesso numero di bit degli addendi: si è infatti generato un *riporto* che richiede un secondo bit (il coefficiente del termine 2^1) per essere rappresentato. Possiamo dunque immaginare il nostro circuito sommatore di numeri a due bit come un circuito che riceve due bit di ingresso (gli addendi *A*1 e *A*2) e genera due bit di uscita (la somma *S* e il riporto *R*). Il legame fra il valore degli ingressi e quello delle uscite può essere rappresentato in forma tabellare come in Tabella 3.1.

*A*1	*A*2	*S*	*R*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 3.1 — Comportamento del circuito sommatore di due bit

Dall'esame di Tabella 3.1, è immediato verificare che l'uscita *R* è il prodotto logico dei due ingressi, quindi che il segnale di riporto può essere realizzato tramite l'operatore AND come:

$$R = A1 \cdot A2$$

Meno immediata è la definizione dell'uscita *S*, per la quale si può comunque notare che risulta 1 quando *A*1 vale 0 **e** *A*2 vale 1, **oppure** quando *A*1 vale 1 **e** *A*2 vale 0. Esprimendo gli stessi concetti in algebra di commutazione, arriviamo alla formula seguente:

$$S = (\overline{A1} \cdot A2) + (A1 \cdot \overline{A2})$$

Vedremo nel prossimo capitolo di individuare metodi più generali di passaggio dalla descrizione del comportamento di una rete logica alla sua sintesi secondo l'algebra di commutazione.

Passiamo invece ora a una seconda considerazione di fondamentale importanza per giustificare l'uso dell'algebra di commutazione: possiamo realizzare facilmente gli operatori dell'algebra di commutazione mediante dispositivi elettronici, che ci consentano di tradurre in effettive macchine elettroniche le espessioni logiche che descrivono il comportamento desiderato (come le due precedenti espressioni di *R* e *S*)? Ancora una volta senza pretese di esaustività, ma con il solo intento di aiutare la comprensione dei diversi risvolti del problema, facciamo riferimento al dispositivo rappresentato nella Figura, e denominato ***transistore a giunzione bipolare***, o più semplicemente transistore bipolare.

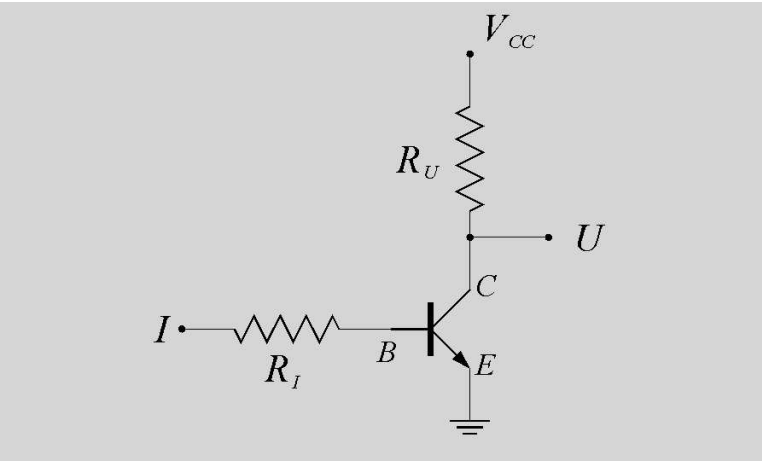


Figura 3.1 — Transistore bipolare collegato per comportarsi come l'operatore NOT

Il comportamento di tale dispositivo, collegato come in figura, è il seguente:

- se l'ingresso *I* viene tenuto a 0 volt (a "massa"), non c'è differenza di potenziale elettrico fra *I* e il polo *E* (*E*mettitore) del transistore, quindi non transita corrente nel tratto *B*-*E* (*B*ase-*E*mettitore) del transistore; il transistore stesso si trova nello stato di *interdizione*, nel quale viene inibito qualsiasi passaggio di corrente nel tratto *C*-*E* (*C*ollettore-*E*mettitore). Il fatto che la resistenza di uscita R_U non sia attraversata da corrente, fa sì che l'uscita *U* si trovi alla

tensione di alimentazione del circuito V_{CC} (tensione di alimentazione in *C*orrente *C*ontinua, pari a 5 Volt nei tradizionali circuiti digitali basati su transistori bipolari);

- se l'ingresso *I* viene portato alla tensione V_{CC} si crea una differenza di potenziale elettrico fra *I* e il polo *E* (*E*mettitore) del transistore, quindi transita corrente nel tratto *B*-*E* (*B*ase-*E*mettitore) del transistore; il transistore stesso viene portato nello stato di *conduzione*, nel quale viene attivato il passaggio di corrente nel tratto *C*-*E* (*C*ollettore-*E*mettitore). Dimensionando opportunamente la resistenza di uscita R_U si può forzare il transistore nello stato estremo di *saturazione*, nel quale la differenza di potenziale fra *C* e *E* (quindi la tensione dell'uscita *U*) sia trascurabile, quindi approssimabile al valore 0.

Si noti che il comportamento elettrico del transistore bipolare, collegato con due resistenze come in figura, realizza circuitalmente l'operatore NOT dell'algebra di commutazione. Un simile circuito prende il nome di **porta NOT** (dall'inglese **NOT gate**), e viene schematizzato nella descrizione realizzativa delle reti elettriche con uno dei due simboli riportati in Figura 3.2:

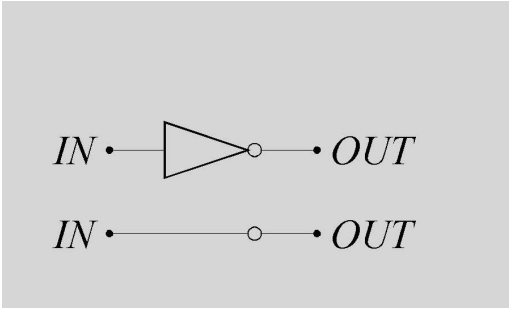


Figura 3.2 — Simboli circuitali dell'operatore NOT

In modo analogo, si possono definire la **porta AND** e la **porta OR**.

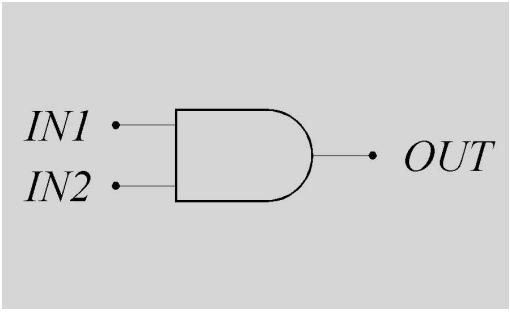


Figura 3.3 — Simbolo circuitale dell'operatore AND

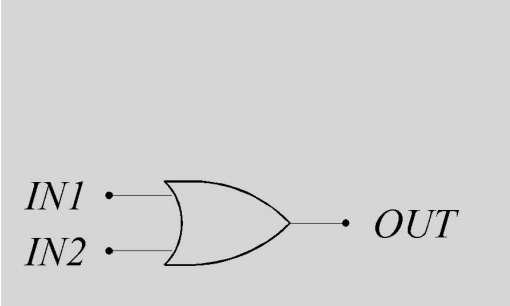


Figura 3.4 — Simbolo circuitale dell'operatore OR

Da un punto di vista tecnologico, si rivelano spesso più facilmente realizzabili porte logiche diverse dalle porte elementari NOT, AND e OR. Sono in particolare da ricordare la **porta NAND** e la **porta NOR**, definite e rappresentate come segue:

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

$$X \text{ NOR } Y = \overline{(X + Y)}$$

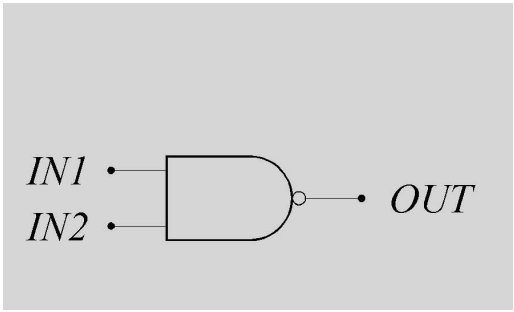


Figura 3.5 — Simbolo circuitale dell'operatore NAND

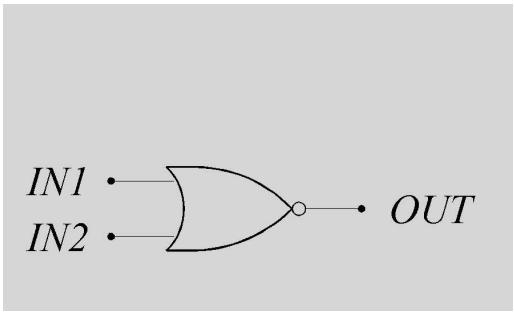


Figura 3.6 — Simbolo circuitale dell'operatore NOR

che possono essere realizzate collegando rispettivamente in serie (emettitore del primo collegato al collettore del secondo) e in parallelo (collettore del primo collegato al collettore del secondo) due transistori bipolari.

È poi da ricordare la porta logica **OR ESCLUSIVO** (o somma modulo-2, o **XOR**) la cui uscita vale 1 se **uno solo** dei due ingressi vale 1, a non entrambi. Tale porta viene indicata come:

$$X \text{ XOR } Y = X \oplus Y = \overline{X} \cdot Y + X \cdot \overline{Y}$$

Infine, è importante sottolineare come la proprietà di associatività consenta di definire porte logiche AND, OR, NAND e NOR a *n* ingressi, nelle quali il comportamento di somma e di prodotto logici (eventualmente negati) si applica a tutte le variabili di ingresso.

4 RETI COMBINATORIE

4.1 DEFINIZIONE

Con il termine di **rete logica combinatoria** (in inglese, *combinational logic network*) si definisce un circuito elettronico digitale realizzato mediante dispositivi elettronici in grado di svolgere funzioni di *porte logiche* (AND, OR, NOT, NAND, NOR, XOR) e caratterizzato dal fatto che **i valori di uscita in ogni istante dipendono unicamente dai valori applicati in tale istante agli ingressi** (ovvero, tali reti non hanno *storia* del funzionamento passato).

Si noti che tale definizione si applica correttamente solo a *reti ideali*, poiché implica che *non ci siano ritardi* fra una modifica di un valore di ingresso e la corrispondente modifica dei valori di uscita. La definizione è applicabile a *reti reali* nell'ipotesi di analizzarne il comportamento a transitorio esaurito. Il comportamento in transitorio sarà oggetto di alcune considerazioni successive.

4.2 ANALISI DELLE RETI COMBINATORIE

L'analisi di una rete combinatoria — allo scopo di valutarne il comportamento, di analizzarne i problemi in transitorio, ecc. — sottintende l'uso di un *metodo descrittivo* che consenta di descriverne le caratteristiche funzionali.

Da un punto di vista realizzativo, il metodo descrittivo più utilizzato è lo **schema logico** della rete combinatoria, ovvero il grafo che rappresenta gli operatori logici utilizzati (cioè le porte logiche) e le interconnessioni reciproche. Un esempio di schema logico di una funzione combinatoria a 4 ingressi e 1 uscita è riportato in Figura 4.1:

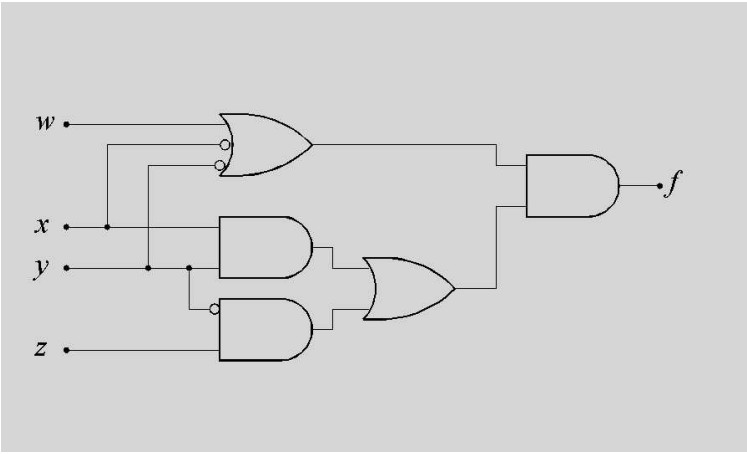


Figura 4.1 — Esempio di rete logica che realizza una funzione combinatoria

Dallo schema logico è possibile dedurre il comportamento funzionale della rete logica: si tratta di applicare una qualsiasi configurazione di ingresso e ricostruire mediante uso delle definizioni degli operatori logici il valore corrispondente dell'uscita.

Un metodo più comodo di definizione del comportamento di una rete logica è la descrizione mediante **espressione logica**, cioè formula che esprime il valore delle uscite in funzione dei letterali delle variabili di ingresso e dei simboli degli operatori logici (in questo caso, AND, OR, NOT, eventualmente XOR) che si applicano a tali letterali. È possibile ricavare tale espressione logica dallo schema logico componendola progressivamente dagli ingressi alle uscite dello schema, come indicato nella Figura 4.2:

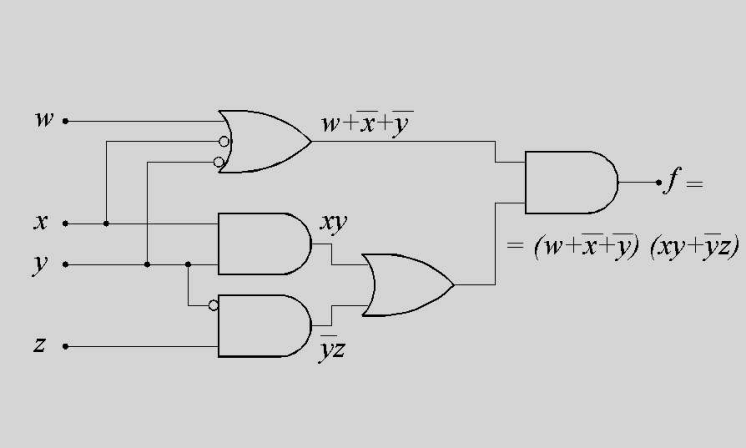


Figura 4.2 — Legame fra rete logica ed espressione logica

Un metodo completamente diverso di descrizione delle funzioni combinatorie — che fa riferimento ai *valori* assunti dalle uscite in corrispondenza delle diverse configurazioni di ingresso piuttosto che alla *dipendenza funzionale* delle uscite dagli ingressi — è la rappresentazione tabellare mediante **tabella delle verità**, cioè elencazione di tutte le possibili configurazioni dei valori di ingresso, associate ai corrispondenti valori assunti dalle uscite. Ad esempio, la tabella delle verità della funzione combinatoria il cui schema logico è stato prima riportato ha l'aspetto di Tabella 4.1:

#	*w*	*x*	*y*	*z*	*f*
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Tabella 4.1 — Tabella delle verità della rete di Figura 4.1

Dalla tabella discendono direttamente due ulteriori metodi di rappresentazione.

Il primo — utile per la sua sinteticità — è costituito dal **numero designante** (*designation number*), ovvero dalla stringa di bit corrispondente al valore assunto dall'uscita per valori crescenti del numero binario che rappresenta ciascuna configurazione di ingresso. Nel caso appena visto, tale numero designante è:

DN(*f*) = (0100010001000111)

Il secondo adotta invece la *rappresentazione geometrica* dei numeri binari per associare alla funzione la cosiddetta **mappa di Karnaugh**, la cui utilità diverrà evidente in fase di sintesi:

f	*yz*			
wx	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	1	1
10	0	1	0	0

Tabella 4.2 — Mappa di Karnaugh equivalente alla Tabella 4.1

4.3 SINTESI DELLE RETI COMBINATORIE

Il processo di sintesi di una rete combinatoria richede che i requisiti funzionali della rete vengano espressi come adeguate *specifiche*, che consentano di esprimere in modo non ambiguo il comportamento atteso dalla rete, e che da tali specifiche sia possibile arrivare a definire lo *schema logico* della rete da realizzare mediante opportuni metodi di sintesi.

Si noti che tali metodi di sintesi possono essere molteplici e arrivare a vari possibili schemi logici che realizzano la medesima funzione logica, dal momento che esistono diverse espressioni logiche aventi il medesimo comportamento in termini di relazione ingressi/uscite (si pensi ai teoremi dell'algebra di commutazione). Quale funzione si sceglie dipende quindi da quale obbiettivo ci si pone (in termini di ottimizzazione della rete risultante dal punto di vista del numero di porte logiche usate, del numero di ingressi per porta logica, del tipo di porte logiche usate, dello spazio di silicio richiesto, ecc.) e da quali potenzialità il metodo di sintesi ci offre (ad esempio, capacità di trattare reti a due o più livelli di porte logiche).

4.3.1 Specifica di reti combinatorie

Per quanto riguarda le tecniche di **specifica del funzionamento** delle reti combinatorie, si ricorre spesso alla rappresentazione tabellare, che consente facilmente di indicare il funzionamento del sistema digitale come relazione ingresso/uscita.

Ad esempio, un semplice antifurto elettronico con chiave numerica a quattro bit costituita dalla sequenza 1011 è specificabile ricorrendo alla seguente tabella delle verità (Tabella 4.3):

#	*w*	*x*	*y*	*z*	*f*
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Tabella 4.3

È opportuno notare che non sempre la funzione che si vuole realizzare richiede di specificare i valori delle uscite in corrispondenza di *tutte* le configurazioni degli ingressi, dal momento che è possibile che alcune configurazioni non si possano mai presentare nell'utilizzo pratico del sistema digitale in oggetto. Si parla in quesi casi di **rete non completamente specificata**, nella quale compare il simbolo × per indicare valori non specificati delle uscite (o **indifferenze**) corrispondenti a configurazioni di ingresso che non si ritiene possano presentarsi.

Un esempio di rete non completamente specificata è il convertitore da codice BCD a codice 7-segmenti rappresentato dalla seguente tabella delle verità (Tabella 4.4):

#	*w*	*x*	*y*	*z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	×	×	×	×	×	×	×
11	1	0	1	1	×	×	×	×	×	×	×
12	1	1	0	0	×	×	×	×	×	×	×
13	1	1	0	1	×	×	×	×	×	×	×
14	1	1	1	0	×	×	×	×	×	×	×
15	1	1	1	1	×	×	×	×	×	×	×

Tabella 4.4

Come avremo modo di vedere, un uso opportuno delle indifferenze può risultare molto utile per migliorare i risultati dell'attività di sintesi.

4.3.2 Sintesi di reti combinatorie

Una volta specificata la funzione attesa da una rete combinatoria, si tratta ora di individuare metodi per **sintetizzare la rete combinatoria** stessa, cioè per derivarne lo schema logico in termini di porte logiche e di struttura di interconnessione fra queste.

Come abbiamo visto, esiste una corrispondenza molto stretta fra schema logico di una rete combinatoria e espressione logica di una funzione combinatoria: la stesura dello schema logico può pertanto essere facilmente ottenuta se si è in grado di convertire una tabella delle verità in una espressione logica che consenta di definire le uscite come funzioni logiche degli ingressi.

Il metodo più semplice e diretto per effettuare tale conversione è quello di:

- considerare separatamente ogni uscita (quindi trattare solo funzioni logiche a singola uscita)
- ricavare dalla tabella corrispondente a ciascuna uscita una delle due **forme canoniche** della tabella stessa, corrispondenti a due particolari espressioni logiche che effettuano le trasformazioni ingresso/uscita indicate dalla tabella.

A tale scopo, si può notare che ad ogni riga di una tabella delle verità si possono associare due funzioni logiche particolari: una che assume valore 1 solo quando gli ingressi presentano la configurazione associata a tale riga, una che assume valore 0 solo quando gli ingressi presentano la configurazione associata a tale riga.

La prima funzione prende il nome di **prodotto fondamentale** (o **mintermine**) dal momento che è costituita dal prodotto logico dei letterali delle variabili di ingresso, presi negati se nella riga considerata la corrispondente variabile di ingresso vale 0, non negati se tale variabile vale 1. Ad esempio, il mintermine associato alla riga individuata dal numero 13 nelle tabelle precedenti è la funzione logica $wx\overline{y}z$.

La seconda funzione prende il nome di **somma fondamentale** (o **maxtermine**) poiché è costituita dalla somma logica dei letterali delle variabili di ingresso, presi non negati se nella riga considerata la corrispondente variabile di ingresso vale 0, negati se tale variabile vale 1. Ad esempio, il maxtermine associato alla riga individuata dal numero 13 nelle tabelle precedenti è la funzione logica $\overline{w} + \overline{x} + y + \overline{z}$.

Una generica funzione logica descritta da una tabella delle verità può a questo punto essere associata a due diverse funzioni logiche, dette appunto **forme canoniche**.

1. La *prima forma canonica* è costituita dalla somma logica dei mintermini associati alle righe della tabella nelle quali l'uscita assume valore 1. Per la definizione di somma logica, tale funzione — detta **somma canonica** — assume infatti valore 1 quando uno qualsiasi dei suoi addendi vale 1 (quindi in corrispondenza delle righe considerate) e valore 0 quando nessuno dei suoi addendi vale 1 (quindi in corrispondenza di tutte le altre righe).

   Ad esempio, la somma canonica della funzione specificata in Tabella 4.1 è la seguente:

   $$\sum_{1}(1,5,9,13,14,15) = \overline{w}\,\overline{x}\,\overline{y}z + \overline{w}x\overline{y}z + w\overline{x}\,\overline{y}z + wx\overline{y}z + wxy\overline{z} + wxyz$$

   Si noti che questa rappresentazione della somma canonica è simile al numero designante, nel senso che vengono indicate le posizioni degli uni nella tabella delle verità.

   Possiamo a questo punto introdurre una notazione convenzionale per descrivere la complessità realizzativa della funzione indicata. Tale notazione riporta il numero di **L**ivelli di porte logiche che il segnale elettrico associato a un qualsiasi letterale di ingresso deve attraversare per raggiungere l'uscita (quindi un'indicazione della rapidità della rete), il numero di **G**ate (cioè di porte logiche) necessari alla sintesi della rete (quindi un'indicazione della dimensione della rete), il numero di **I**ngressi totali alle porte logiche usate (quindi un'indicazione della dimensione media delle porte stesse).

   Nel caso sopra indicato, ogni ingresso deve superare due livelli di porte (le porte AND associate ai vari mintermini e la porta OR finale), ci sono in tutto 6 porte AND e una porta OR, le porte AND hanno tutte 4 ingressi mentre la porta OR ha 6 ingressi. La notazione che descrive questa rete è dunque **2L7G30I**.

2. la *seconda forma canonica* è costituita dal prodotto logico dei maxtermini associati alle righe della tabella nelle quali l'uscita assume valore 0; per la definizione di prodotto logico, tale funzione — detta **prodotto canonico** — assume infatti valore 0 quando uno qualsiasi dei suoi fattori vale 0 (quindi in

corrispondenza delle righe considerate) e valore 1 quando nessuno dei suoi addendi vale 0 (quindi in corrispondenza di tutte le altre righe).

Ad esempio, il prodotto canonico della funzione specificata in Tabella 4.1 è il seguente:

$$\prod_0 (0,2,3,4,6,7,8,10,11,12) = (w + x + y + z)\,(w + x + \overline{y} + z)\,(w + x + \overline{y} + \overline{z})$$
$$(w + \overline{x} + y + z)\,(w + \overline{x} + \overline{y} + z)\,(w + \overline{x} + \overline{y} + \overline{z})\,(\overline{w} + x + y + z)\,(\overline{w} + x + \overline{y} + z)\,(\overline{w} + x + \overline{y} + \overline{z})\,(\overline{w} + \overline{x} + y + z)$$

Si noti che anche questa rappresentazione della somma canonica è simile al numero designante, nel senso che vengono indicate le posizioni degli zeri nella tabella delle verità.

Volendo applicare anche a questa sintesi la notazione descrittiva della complessità realizzativa della funzione, otteniamo **2L11G50I**.

Questo metodo consente dunque di tradurre una tabella delle verità in una funzione immediatamente traducibile in una *rete logica a due livelli*, caratterizzata dai seguenti elementi:

- un numero di porte di tipo AND (OR) pari al numero di mintermini (maxtermini) utilizzati nella somma canonica (nel prodotto canonico); ciascuna delle suddette porte AND (OR) ha un numero di ingressi pari al numero di variabili di ingresso;
- una porta OR (AND) con un numero di ingressi pari al numero di mintermini (maxtermini) utilizzati nella somma canonica (nel prodotto canonico);
- un numero adeguato di negatori (o la disponibilità di *two-rail inputs*, cioè di valori diritti e negati delle variabili di ingresso).

4.4 MINIMIZZAZIONE DELLE RETI COMBINATORIE

Le forme canoniche sopra ottenute sono espressioni logiche che consentono la sintesi della corrispondente funzione tabellata, ma che molto raramente sono *ottimizzate* dal punto di vista del numero e della complessità delle porte logiche utilizzate.

Un primo metodo per l'ottimizzazione di tali espressioni consiste nell'applicazione dei teoremi dell'algebra di commutazione al fine di ridurre il numero di termini presenti e il numero di letterali che compaiono in ogni termine.

Ad esempio, la somma canonica della funzione specificata in tabella 4.1 può essere così elaborata:

$$\overline{w}\,\overline{x}\,\overline{y}\,z + \overline{w}\,x\,\overline{y}\,z + w\,\overline{x}\,\overline{y}\,z + wx\,\overline{y}\,z + wxy\,\overline{z} + wxyz =$$

$$= \overline{w}\,\overline{y}\,z \cdot (\overline{x} + x) + w\,\overline{y}\,z \cdot (\overline{x} + x) + wxy \cdot (\overline{z} + z) = \qquad \text{(distributività)}$$

$$= \overline{w}\,\overline{y}\,z \cdot 1 + w\,\overline{y}\,z \cdot 1 + wxy \cdot 1 = \qquad \text{(complementi)}$$

$= \overline{w}\ \overline{y}\ z + w\overline{y}\ z + wxy =$ (identità)

$= \overline{y}\ z \cdot (\overline{w} + w) + wxy =$ (distributività)

$= \overline{y}\ z \cdot 1 + wxy =$ (complementi)

$= \overline{y}\ z + wxy$ (identità)

ottenendo una rete **2L3G7I**, decisamente molto più semplice di entrambe le forme canoniche viste in precedenza.

La stessa somma canonica può però essere elaborata anche in modo completamente diverso:

$\overline{w}\ \overline{x}\ \overline{y}\ z + \overline{w}\ x\ \overline{y}\ z + w\ \overline{x}\ \overline{y}\ z + wx\ \overline{y}\ z + wxy\ \overline{z} + wxyz =$

$= w\ \overline{x}\ \overline{y}\ z + wx\ \overline{y}\ z + \overline{w}\ \overline{x}\ \overline{y}\ z + \boldsymbol{w\ \overline{x}\ \overline{y}\ z} + \overline{w}\ x\ \overline{y}\ z +$

$+ \boldsymbol{\overline{w}\ x\ \overline{y}\ z} + \boldsymbol{w\ x\ \overline{y}\ z} + \boldsymbol{wx\ \overline{y}\ z} + wxy\ \overline{z} + wxyz =$ (idempotenza: aggiunta termini)

$= w\ \overline{y}\ z \cdot (\overline{x} + x) + \overline{x}\ \overline{y}\ z \cdot (\overline{w} + w) +$

$+ \overline{y}\ z \cdot (\overline{w}\ x + \overline{w}\ \overline{x} + w\ \overline{x} + wx) + wxy \cdot (\overline{z} + z) =$ (distributività)

$= w\ \overline{y}\ z + \overline{x}\ \overline{y}\ z + \overline{y}\ z + wxy =$ (complementi e identità)

$= w\ \overline{y}\ z + \overline{x}\ \overline{y}\ z + \overline{y}\ \overline{y}\ z + wxy =$ (idempotenza)

$= \overline{y}\ z \cdot (w + \overline{x} + \overline{y}) + wxy + 0 + 0 =$ (distributività e identità)

$= \overline{y}\ z \cdot (w + \overline{x} + \overline{y}) + wxy + \overline{x}\ xy + \overline{y}\ xy =$ (complementi)

$= \overline{y}\ z \cdot (w + \overline{x} + \overline{y}) + xy \cdot (w + \overline{x} + \overline{y}) =$ (distributività)

$= (\overline{y}\ z + xy) \cdot (w + \overline{x} + \overline{y})$ (distributività)

In questo caso, la funzione dà luogo alla rete **3L5G11I** rappresentata in Figura 4.1.

Quale delle due forme sia la migliore (come pure, quale fra altre possibili forme possa essere considerata ottimale) dipende naturalmente dai criteri di ottimalità che si ritiene di dover adottare.

Nell'ipotesi di considerare solo *reti a due livelli* — costituite cioè da un certo numero di termini prodotto (AND) o somma (OR) dei letterali delle variabili di ingresso, seguiti da un termine somma o prodotto che genera l'uscita — esistono metodi di ottimizzazione basati sul criterio di ridurre al minimo *il numero di porte logiche utilizzate nel primo livello*. Si cerca in altre parole di identificare la **somma minima** (o il **prodotto minimo**) caratterizzati da avere il minimo numero di termini prodotto (somma), ciascuno avente il minimo numero di fattori (addendi).

Tali metodi partono dalla struttura delle forme canoniche (che sono come ovvio espressioni immediatamente traducibili in reti a due livelli) e sfruttano la possibilità di applicare le uguaglianze:

$xy + x\overline{y} = x \cdot (y + \overline{y}) = x \cdot 1 = x$

e

$(x + y) \cdot (x + \overline{y}) = x + (y \cdot \overline{y}) = x + 0 = x$

ai vari mintermini o maxtermini.

4.4.1 Minimizzazione di reti combinatorie mediante mappe di Karnaugh

Il primo metodo di minimizzazione di reti combinatorie a due livelli sfrutta la possibilità di effettuare una *rappresentazione geometrica dei numeri binari.*

Le configurazioni binarie sono rappresentabili come *n-cubi* in uno spazio a *n* dimensioni. La costruzione di un *n-cubo* è una operazione ricorsiva, nella quale si proietta una copia di un (*n-1*)*-cubo,* e si premette 0 a tutti gli identificatori degli elementi dell'(*n-1*)*-cubo* di partenza, e si premette un 1 a tutti gli identificatori degli elementi dell'(*n-1*)*-cubo* proiettato. Lo 0*-cubo* è un punto privo di identificatore.

Da un punto di vista grafico, la costruzione avviene nel modo indicato in Figura 4.3:

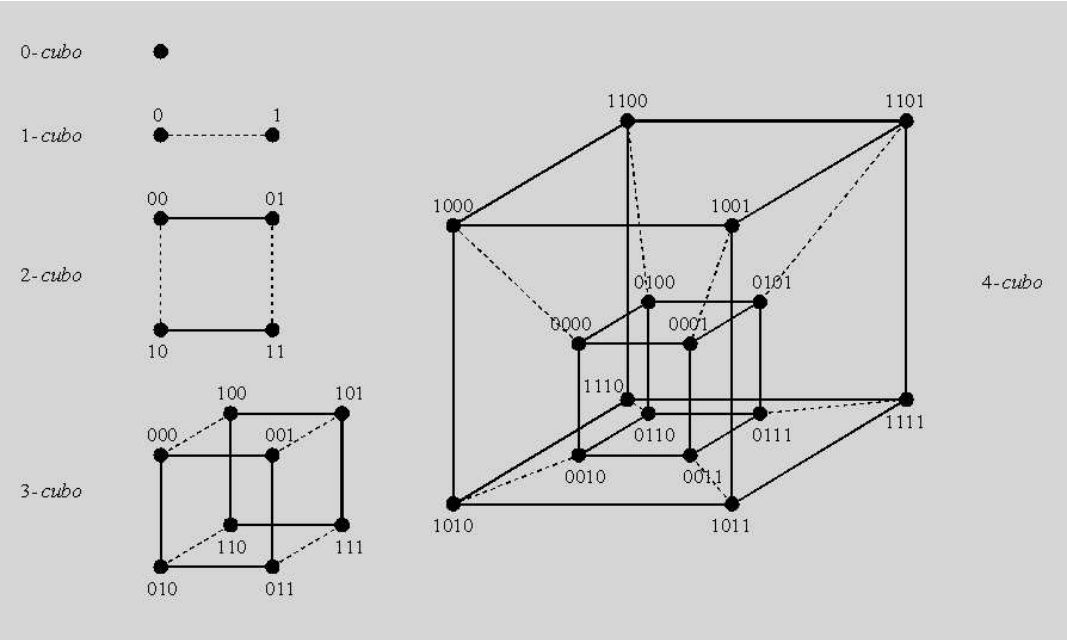


Figura 4.3 — Costruzione degli *n*-cubi per la rappresentazione geometrica dei numeri binari

Proprietà:

- i punti di un *n-cubo* sono connessi con punti a *distanza di Hamming* unitaria;

- la distanza fra due punti generici è data dal percorso più breve per connetterli, e viceversa ∃ percorso di lunghezza pari alla distanza di Hamming;
- un *p-sottocubo* di un *n-cubo* è l'insieme di 2^p punti aventi (*n-p*) bit corrispondenti uguali nei loro identificatori

Esistono altre due rappresentazioni geometriche degli *n-cubi*:

mesh *n-cubo* "aperto" sul piano, con connessioni sottintese fra gli elementi estremi (Figura 4.4);

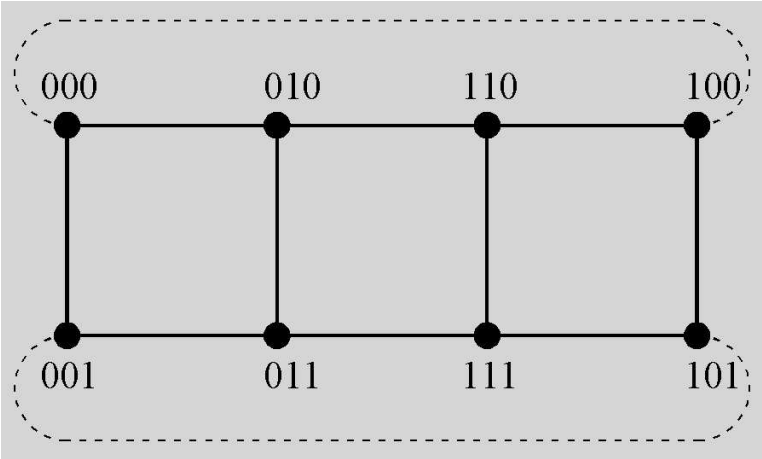


Figura 4.4 — Rappresentazione geometrica a *mesh* dei numeri binari

mappa nota anche come **mappa di Karnaugh**, associa a ogni vertice dell'*n-cubo* una casella di una matrice, secondo il seguente schema:

	00	01	11	10
0	000	010	110	100
1	001	011	111	101

3-*cubo*

	00	01	11	10
00	0000	0100	1100	1000
01	0001	0101	1101	1001
11	0011	0111	1111	1011
10	0010	0110	1110	1010

4-*cubo*

Il primo metodo di minimizzazione delle reti combinatorie si basa appunto sulle **mappe di Karnaugh**, e sfrutta il fatto che ogni casella della mappa è adiacente a caselle corrispondenti a mintermini (maxtermini) aventi distanza di Hamming unitaria dal mintermine (maxtermine) corrispondente alla casella considerata. Se si utilizza la forma canonica SP (somma di prodotti, cioè somma canonica di mintermini) si nota che due uni situati in due caselle adiacenti corrispondono a due mintermini nei quali si possono raccogliere a fattore comune le variabili che non cambiano valore nelle espressioni dei due mintermini, eliminando invece l'unica variabile che cambia valore. Allo stesso modo,

quattro uni situati in caselle disposte a quadrato o allineate corrispondono a quattro mintermini nei quali si possono raccogliere a fattore comune le variabili che non cambiano valore nelle espressioni dei due mintermini, eliminando invece le uniche due variabili che cambiano valore.

Si deve rifarsi alla nozione di **sottocubo** come una parte di un *n*-cubo costituita da 2^p elementi (con $p \leq n$) caratterizzati dal fatto di avere *n-p* coordinate uguali fra loro, e dal fatto di far assumere alle restanti *p* coordinate tutte le possibili configurazioni.

Un sottocubo nel quale la funzione da sintetizzare assume sempre valore 1 è dunque associabile a un prodotto logico, che prende il nome di **implicante** della funzione e che è costituito dalle sole *n-p* variabili di ingresso che non cambiano il loro valore. In modo duale, un sottocubo nel quale la funzione da sintetizzare assume sempre valore 0 è associabile a una somma logica, che prende il nome di **implicato** della funzione e che è costituita dalle sole *n-p* variabili di ingresso che non cambiano il loro valore.

A titolo di esempio, si evidenziano in Figura 4.5 i vertici del 4-cubo associati agli uni della funzione descritta in Tabella 4.1. Come si può vedere, tali vertici individuano due 1-sottocubi e un 2-sottocubo che consentono di ricavare gli implicanti della funzione.

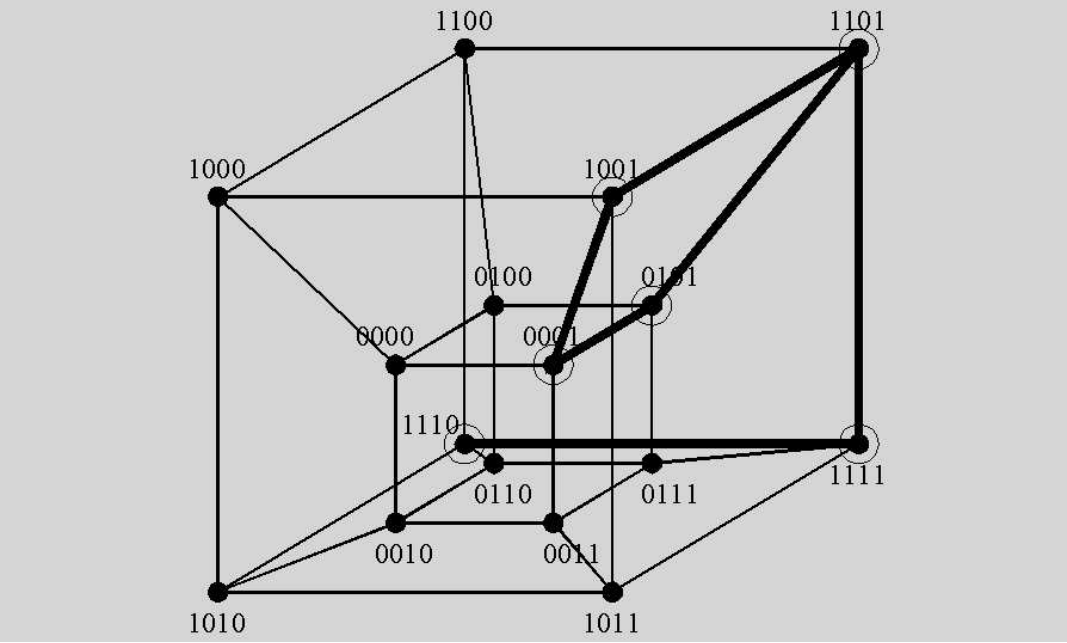


Figura 4.5 — Identificazione geometrica degli implicanti della funzione descritta in Tabella 4.1

Un implicante corrisponde dunque a un prodotto logico dei letterali delle sole variabili di ingresso che non cambiano valore, presi negati se la corrispondente variabile di ingresso vale 0, non negati se tale variabile vale 1. In modo duale, un implicato corrisponde alla somma logica dei letterali delle sole variabili di ingresso che non cambiano valore, presi negati se la corrispondente variabile di ingresso vale 1, non negati se tale variabile vale 0.

Ecco alcuni esempi di implicanti:

f	*yz*			
wx	00	01	11	10
00	1	0	0	0
01	0	1	1	0
11	0	1	1	1
10	1	0	0	0

Tabella 4.5

Si noti che un implicante può essere contenuto in un implicante di dimensioni maggiori. Un implicante non contenuto in nessun implicante di dimensioni maggiori prende il nome di **implicante primo**.

f	*yz*			
wx	00	01	11	10
00	1	0	0	0
01	0	1	1	0
11	0	1	1	1
10	1	0	0	0

Tabella 4.6

Si può notare come la somma minima debba contenere solo implicanti primi, anche se non necessariamente *tutti* gli implicanti primi presenti, come dimostra la seguente mappa:

f	*xy*			
w	00	01	11	10
0	1	1	0	0
1	1	0	0	1

Tabella 4.7

Per identificare quali prodotti devono essere presenti nella somma minima, si può ricorrere al concetto di **implicanti essenziali**: viene definito implicante essenziale un implicante primo che abbia la caratteristica di contenere almeno un mintermine non contenuto in nessun altro implicante primo (cioè un implicante primo che "copra" almeno un mintermine non coperto da altri). Sicuramente, ogni implicante essenziale deve essere contenuto nella somma minima. Qualora rimangano mintermini isolati non coperti da nessun implicante essenziale, si includono scegliendo i maggiori implicanti primi che li coprono:

f	*yz*			
wx	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	1	1	1
10	1	1	0	0

Tabella 4.8

Può capitare che non esistano implicanti essenziali (almeno in alcune zone della mappa) come dimostra la seguente mappa. In questi casi, si è costretti ad adottare una tecnica di *trial and error*.

f *wx* \ *yz*	00	01	11	10
00	0	0	1	0
01	0	1	1	0
11	0	1	0	0
10	0	1	1	0

Tabella 4.9

4.4.2 Reti non completamente specificate

Come già introdotto nella sezione dedicata alla specifica, può capitare di dover progettare reti logiche che **non** prevedono di dover gestire tutte le possibili configurazioni delle variabili di ingresso, ma solo un sottoinsieme. Sono esempi di queste reti i convertitori da codice BCD a qualsiasi altro codice (solo le prime 10 configurazioni di quattro cifre binarie costituiscono configurazioni BCD valide), i convertitori da codice eccesso-3 a qualsiasi altro codice, ecc.

Le configurazioni di uscita i cui valori non sono specificati (o **indifferenze**, indicate dal simbolo "×") possono essere proficuamente utilizzate per minimizzare la sintesi della rete, assegnando a tali configurazioni il valore 0 o 1 in base alla possibilità di realizzare implicanti/implicati primi di dimensioni maggiori.

Un esempio di tale possibilità è la seguente sintesi delle sette funzioni logiche corrispondenti al convertitore da BCD a 7-segmenti tabellato in Tabella 4.4. In tale sintesi, ogni indifferenza forzata a 0 o 1 viene indicata rispettivamente come $\times_0$ e $\times_1$

a *wx* \ *yz*	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	$\times_1$	$\times_1$	$\times_1$	$\times_1$
10	1	1	$\times_1$	$\times_1$

$$a = w + y + xz + \overline{x}\,\overline{z}$$

Tabella 4.10.a

b *wx* \ *yz*	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	$\times_1$	$\times_0$	$\times_1$	$\times_0$
10	1	1	$\times_1$	$\times_1$

$$b = \overline{x} + \overline{y}\,\overline{z} + yz$$

Tabella 4.10.b

c *yz*

$c = x + \overline{y} + z$

wx	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	$\times_1$	$\times_1$	$\times_1$	$\times_1$
10	1	1	$\times_1$	$\times_0$

Tabella 4.10.c

d *yz*

$d = \overline{x}\,\overline{z} + y\overline{z} + \overline{x}y + x\overline{y}z$

wx	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	$\times_0$	$\times_1$	$\times_0$	$\times_1$
10	1	0	$\times_1$	$\times_1$

Tabella 4.10.d

e *yz*

$e = \overline{x}\,\overline{z} + y\overline{z}$

wx	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	$\times_0$	$\times_0$	$\times_0$	$\times_1$
10	1	0	$\times_0$	$\times_1$

Tabella 4.10.e

f *yz*

$f = \overline{y}\,\overline{z} + x\overline{y} + w + x\overline{z}$

wx	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	$\times_1$	$\times_1$	$\times_1$	$\times_1$
10	1	1	$\times_1$	$\times_1$

Tabella 4.10.f

g *yz*

$g = w + y\overline{z} + \overline{x}y + x\overline{y}$

wx	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	$\times_1$	$\times_1$	$\times_1$	$\times_1$
10	1	1	$\times_1$	$\times_1$

Tabella 4.10.g

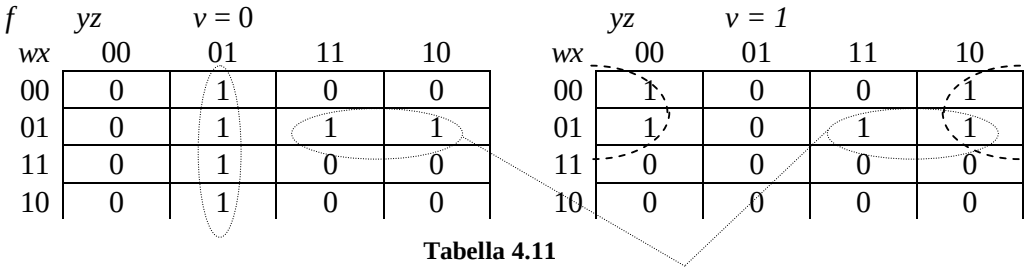
4.4.3 Funzioni con 5 o 6 variabili

La tecnica delle mappe di Karnaugh si può applicare anche a funzioni con 5 o 6 variabili, rappresentando in questo caso i 5-cubi come una mappa di quattro variabili con le celle divise a metà (una parte associata a valore 0 della quinta variabile, l'altra a valore 1 della quinta variabile) e i 6-cubi come gruppi di 4 mappe da 4 variabili, contraddistinte da valori adiacenti della quinta e della sesta variabile.

Come primo esempio, consideriamo la seguente funzione di 5 variabili:

$$f(v,w,x,y,z) = \sum_{1} (1,5,6,7,9,13,16,18,20,22,23)$$

Tale funzione può essere schematizzata dalla seguente coppia di mappe di Karnaugh:



f	*yz*		*v* = 0			*yz*		*v* = *1*	
wx	00	01	11	10	*wx*	00	01	11	10
00	0	1	0	0	00	1	0	0	1
01	0	1	1	1	01	1	0	1	1
11	0	1	0	0	11	0	0	0	0
10	0	1	0	0	10	0	0	0	0

Tabella 4.11

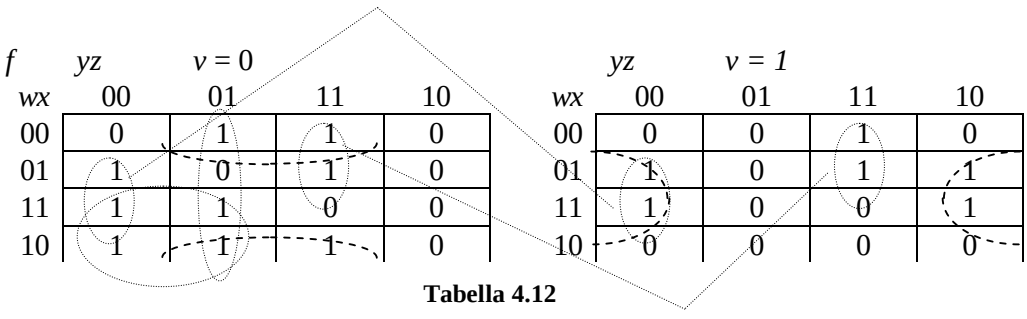
La sintesi (ottenibile mediante somma dei soli implicanti essenziali) risulta essere la seguente:

$$f = v\,\overline{w}\,\overline{z} + \overline{v}\,\overline{y}\,z + \overline{w}\,x\,y$$

Come secondo esempio, consideriamo la seguente funzione di 5 variabili:

$$f(v,w,x,y,z) = \sum_{1} (1,3,4,7,8,9,11,12,13,19,20,22,23,28,30)$$

Tale funzione può essere schematizzata dalla seguente coppia di mappe di Karnaugh:



f	*yz*		*v* = 0			*yz*		*v* = *1*	
wx	00	01	11	10	*wx*	00	01	11	10
00	0	1	1	0	00	0	0	1	0
01	1	0	1	0	01	1	0	1	1
11	1	1	0	0	11	1	0	0	1
10	1	1	1	0	10	0	0	0	0

Tabella 4.12

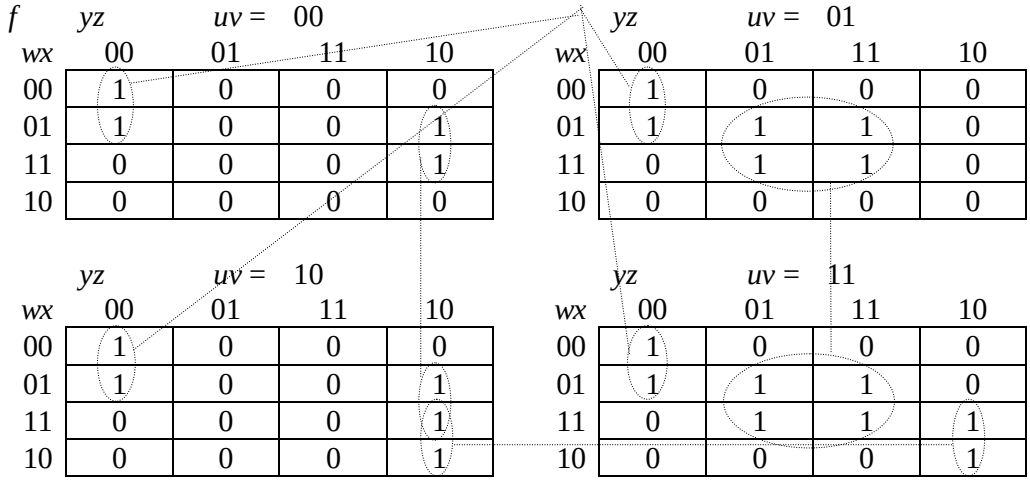
La sintesi (ottenibile mediante somma dei soli implicanti essenziali) risulta essere la seguente:

$$f = x\,\overline{y}\,\overline{z} + \overline{w}\,yz + vx\,\overline{z} + \overline{v}\,\overline{x}\,z + \overline{v}\,w\,\overline{y}$$

Si voglia ora sintetizzare la funzione di sei variabili:

$$f(u,v,w,x,y,z) = \sum_{1} (0,4,6,14,16,20,21,23,29,31,32,36,38,42,46,48,52,53,55,58,61,62,63)$$

corrispondente alle quattro mappe di Karnaugh:



f	*yz* / *wx*	00	01	11	10
uv = 00	00	1	0	0	0
	01	1	0	0	1
	11	0	0	0	1
	10	0	0	0	0
uv = 01	00	1	0	0	0
	01	1	1	1	0
	11	0	1	1	0
	10	0	0	0	0
uv = 10	00	1	0	0	0
	01	1	0	0	1
	11	0	0	0	1
	10	0	0	0	1
uv = 11	00	1	0	0	0
	01	1	1	1	0
	11	0	1	1	1
	10	0	0	0	1

Tabella 4.13

La cui sintesi porta a:

$$f = \overline{w}\,\overline{y}\,\overline{z} + \overline{v}\,xy\,\overline{z} + uwy\,\overline{z} + vxz$$

4.4.4 Metodo tabellare (Quine-McCluskey) per la minimizzazione di reti combinatorie

Nel caso si abbia a che fare con funzioni di più di 6 variabili, il metodo delle mappe di Karnaugh diventa difficilmente gestibile, e conviene ricorrere a tecniche automatizzabili come il metodo tabellare di Quine-McCluskey.

Il metodo — applicato alla prima forma canonica (somma di prodotti) — si basa su due passi successivi:

1. identificazione di tutti gli implicanti primi;
2. determinazione dell'insieme minimo di implicanti primi che coprono tutti gli uni della funzione.

Identificazione di tutti gli implicanti primi

Poiché un implicante copre sempre uni appartenenti a mintermini adiacenti (quindi a distanza di Hamming unitaria) è possibile identificare tali mintermini raggruppandoli per valori crescenti del numero totale di uni presenti nei mintermini stessi, e confrontando ogni mintermine del gruppo *i*-esimo con tutti i mintermini del gruppo (*i*+1)-esimo.

Se fra tali mintermini vi è adiacenza (un solo valore di variabili di ingresso diverso), essi possono essere riuniti in un implicante che li comprenda entrambi, marcando i mintermini suddetti come "riuniti" (simbolo "✓"). Tale implicante segnala la sua indipendenza dal valore della variabile che differenzia i due mintermini identificandola con il simbolo "-".

Le Tabelle 4.14.a e 4.14.b mostrano l'applicazione del passo appena descritto alla funzione di cinque variabili:

$$f(v,w,x,y,z) = \sum_{1} (0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$$

A partire dai mintermini di Tabella 4.14.a si identificano gli implicanti (gli 1-sottocubi) indicati in Tabella 4.14.b.

#	*v*	*w*	*x*	*y*	*z*	
0	0	0	0	0	0	✓
2	0	0	0	1	0	✓
4	0	0	1	0	0	✓
8	0	1	0	0	0	✓
16	1	0	0	0	0	✓
6	0	0	1	1	0	✓
10	0	1	0	1	0	✓
12	0	1	1	0	0	✓
18	1	0	0	1	0	✓
7	0	0	1	1	1	✓
11	0	1	0	1	1	✓
13	0	1	1	0	1	✓
14	0	1	1	1	0	✓
19	1	0	0	1	1	✓
29	1	1	1	0	1	✓
30	1	1	1	1	0	✓

Tabella 4.14.a

#	*v*	*w*	*x*	*y*	*z*	
0,2	0	0	0	-	0	✓
0,4	0	0	-	0	0	✓
0,8	0	-	0	0	0	✓
0,16	-	0	0	0	0	✓
2,6	0	0	-	1	0	✓
2,10	0	-	0	1	0	✓
2,18	-	0	0	1	0	✓
4,6	0	0	1	-	0	✓
4,12	0	-	1	0	0	✓
8,10	0	1	0	-	0	✓
8,12	0	1	-	0	0	✓
16,18	1	0	0	-	0	✓
6,7	0	0	1	1	-	p_1
6,14	0	-	1	1	0	✓
10,11	0	1	0	1	-	p_2
10,14	0	1	-	1	0	✓
12,13	0	1	1	0	-	p_3
12,14	0	1	1	-	0	✓
18,19	1	0	0	1	-	p_4
13,29	-	1	1	0	1	p_5
14,30	-	1	1	1	0	p_6

Tabella 4.14.b

A questo punto, si itera la verifica precedente sugli implicanti (1-sottocubi) individuati, avendo cura di confrontare solo quelli fra loro compatibili (cioè aventi il simbolo "-" di indicazione della variabile ininfluente nella medesima posizione) e si costruiscono dei nuovi imlicanti di dimensioni maggiori (2-sottocubi) ogniqualvolta si trovano due 1-sottocubi adiacenti (cioè compatibili, e con distanza di Hamming unitaria nei restanti valori delle variabili di ingresso). Il risultato della ricerca di 2-sottocubi e 3-sottocubi è riportato nelle Tabelle 4.14.c e 4.14.d (per la funzione qui considerata esiste un unico implicante costituito da un 3-sottocubo, quindi la ricerca di 4-sottocubi è inutile).

#	*v*	*w*	*x*	*y*	*z*	
0,2,4,6	0	0	-	-	0	✓
0,2,8,10	0	-	0	-	0	✓
0,2,16,18	-	0	0	-	0	p_7
0,4,8,12	0	-	-	0	0	✓
2,6,10,14	0	-	-	1	0	✓
4,6,12,14	0	-	1	-	0	✓
8,10,12,14	0	1	-	-	0	✓

Tabella 4.14.c

#	*v*	*w*	*x*	*y*	*z*	
0,2,4,6,8,10,12,14	0	-	-	-	0	p_8

Tabella 4.14.d

Al termine del procedimento, rimangono non marcati (cioè non indicati dal simbolo "✓") tutti gli implicanti che non sono stati inclusi in implicanti di dimensioni maggiori, pertanto tutti gli *implicanti primi* della funzione (inclusi eventuali 1 isolati, costituiti da implicanti primi associati a 0-sottocubi).

Per la funzione in esame, sono stati individuati i seguenti implicanti primi:

$p_1 = 6,7$ $p_2 = 10,11$ $p_3 = 12,13$ $p_4 = 18,19$

$p_5 = 13,29$ $p_6 = 14,30$ $p_7 = 0,2,16,18$ $p_8 = 0,2,4,6,8,10,12,14$

Costruzione della somma minima

Per scegliere quali implicanti primi devono essere inclusi nella forma minima, si costruisce a questo punto la **tabella di copertura**, contenente una riga per ogni implicante primo e una colonna per ogni mintermine della funzione. In tale tabella, si indica con un simbolo "×" il fatto che il mintermine presente sulla *i*-esima colonna sia coperto dall'implicante primo associato alla *j*-esima riga.

L'esempio di funzione qui utilizzato dà luogo alla Tabella 4.15.

	0	2	4	8	16	6	10	12	18	7	11	13	14	19	29	30
p_1						×				×						
p_2							×				×					
p_3								×				×				
p_4									×					×		
p_5												×			×	
p_6													×			×
p_7	×	×			×				×							
p_8	×	×	×	×		×	×	×					×			

Tabella 4.15

A questo punto, si procede come segue.

Innanzitutto, si cercano gli *implicanti essenziali*, costituiti dagli implicanti primi che includono almeno un mintermine non coperto da nessun altro implicante primo. Per far questo, si individuano le colonne contenenti un unico simbolo "×" ("colonne singole") e si marcano come essenziali gli implicanti primi che coprono tali "×". Si procede poi a eliminare dalla tabella tutte le colonne associate a simboli "×" coperti dagli implicanti essenziali così individuati, in modo da ridurre il lavoro successivo a tutti i mintermini non coperti da implicanti essenziali.

Nel caso in esame, tutti i mintermini vengono cancellati da questa operazione, quindi si può sintetizzare la seguente forma minima della funzione:

$f = p_1 + p_2 + p_4 + p_5 + p_6 + p_7 + p_8$

In altri casi, le cose possono andare in modo diverso, come succede ad esempio per la funzione di quattro variabili:

$$f(w,x,y,z) = \sum_{1} (1,3,6,7,8,9,12,13)$$

In questo caso, la prima fase porta a individuare come primi gli implicanti non marcati nelle tabelle 4.16.a, 4.16.b e 4.16.c.

#	*w*	*x*	*y*	*z*	
1	0	0	0	1	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
6	0	1	1	0	✓
9	1	0	0	1	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	0	1	✓

Tabella 4.16.a

#	*w*	*x*	*y*	*z*	
1,3	0	0	-	1	p_1
1,9	-	0	0	1	p_2
8,9	1	0	0	-	✓
8,12	1	-	0	0	✓
3,7	0	-	1	1	p_3
6,7	0	1	1	-	p_4
9,13	1	-	0	1	✓
12,13	1	1	0	-	✓

Tabella 4.16.b

#	*w*	*x*	*y*	*z*	
8,9,12,13	1	-	0	-	p_5

Tabella 4.16.c

Questo porta a individuare i cinque implicanti primi

$p_1 = 1,3$ $p_2 = 1,9$ $p_3 = 3,7$ $p_4 = 6,7$ $p_5 = 8,9,12,13$

che possono essere organizzati nella seguente tabella di copertura (Tabella 4.17.a):

	1	8	3	6	9	12	7	13
p_1	×		×					
p_2	×				×			
p_3			×				×	
p_4				×			×	
p_5		×			×	×		×

Tabella 4.17.a

Il procedimento di ricerca delle colonne singole porta a individuare come essenziali gli implicanti primi p_4 e p_5, la cui inclusione nella somma minima permette di eliminare i mintermini 8, 6, 9, 12, 7, 13. Si giunge dunque alla tabella di copertura ridotta (Tabella 4.17.b):

	1	3
p_1	×	×
p_2	×	
p_3		×

Tabella 4.17.b

A una tabella ridotta come questa si possono applicare i seguenti due concetti.

1. **Dominanza fra righe**, che si verifica ogniqualvolta un implicante p_i copre **tutti** i mintermini coperti dall'implicante p_j più almeno uno. Si dice in tal caso che l'implicante p_i **domina** l'implicante p_j. L'implicante dominato può essere cancellato dal momento che qualsiasi forma della funzione che utilizzasse p_j non sarebbe minima rispetto alla stessa forma utilizzante p_i.
2. **Dominanza fra colonne**, che si verifica ogniqualvolta ogni implicante primo che copre la colonna c_j copre anche la colonna c_i ma non viceversa. Si dice in tal caso che la colonna c_i **domina** la colonna c_j. La colonna *dominante* c_i può in questo caso essere eliminata, dato che qualsiasi scelta che garantisca la copertura della più restrittiva colonna c_j assicura anche la copertura di c_i.

Nel caso della funzione in esame, al riga p_1 domina sia la riga p_2 sia la riga p_3. Si possono dunque eliminare le due righe dominate, e ricavare l'ultimo implicante primo da inserire nella forma minima (p_1). Tale forma risulta essere:

$f = p_1 + p_4 + p_5$

I due concetti sopra indicati possono essere applicati in modo iterativo. Quando non è più possibile individuare alcun tipo di dominanza siamo di fronte a una **tabella ciclica**, per la quale si deve procedere in modo esaustivo o mediante opportune euristiche.

Per esercizio, si può applicare il metodo di Quine-McCluskey alla seguente funzione di 5 variabili:

$$f(v,w,x,y,z) = \sum_{1} (1,4,5,7,8,9,11,13,14,15,18,19,20,21,23,24,25,26,27,28,29,30)$$

Il metodo di Quine-McCluskey si può applicare anche a funzioni non completamente specificate, contenenti cioè indifferenze associate ad alcune configurazioni di ingresso. In questo caso, si procede nel modo seguente:

- durante la ricerca degli implicanti primi, si considerano uni le indifferenze, onde individuare gli implicanti di massime dimensioni consentite dallo sfruttamento delle indifferenze stesse;
- durante la costruzione della tabella di copertura, si inseriscono solo gli uni effettivi della funzione, per non inserire nella sintesi minima implicanti resi necessari unicamente dalla forzatura a uno di alcune indifferenze.

Come esempio, consideriamo la seguente funzione non completamemnte specificata di 4 variabili:

$$f(w,x,y,z) = \sum_{1} (3,12,13) + \sum_{\times} (5,6,7,15)$$

La sintesi avviene nel modo seguente:

#	*w*	*x*	*y*	*z*	
3	0	0	1	1	✓
5	0	1	0	1	✓
6	0	1	1	0	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	0	1	✓
15	1	1	1	1	✓

Tabella 4.18.a

#	*w*	*x*	*y*	*z*	
3,7	0	-	1	1	p_1
5,7	0	1	-	1	✓
5,13	-	1	0	1	✓
6,7	0	1	1	-	p_2
12,13	1	1	0	-	p_3
7,15	-	1	1	1	✓
13,15	1	1	-	1	✓

Tabella 4.18.b

#	*w*	*x*	*y*	*z*	
5,7,13,15	-	1	-	1	p_4

Tabella 4.18.c

Questo porta a individuare i quattro implicanti primi

$p_1 = 3,7$ $p_2 = 6,7$ $p_3 = 12,13$ $p_4 = 5,7,13,15$

che devono essere organizzati nella seguente tabella di copertura (Tabella 4.19):

	3	12	13
p_1	×		
p_2			
p_3		×	×
p_4			×

Tabella 4.19

Come si può notare, l'implicante p_2 non copre nessun uno della funzione (è infatti un implicante costituito da sole indifferenze) mentre altri implicanti (p_1 e p_4) coprono un solo uno della funzione ciascuno, poiché contengono anche indifferenze.

È anche facile ricavare, osservando le colonne singolari che consentono di identificare gli implicanti essenziali, la sintesi minima della funzione:

$f = p_1 + p_3$

4.5 RETI COMBINATORIE A PIÙ USCITE

Spesso le reti combinatorie che devono essere sintetizzate sono previste per generare più uscite a partire dagli stessi ingressi. La sintesi "scorrelata" di ciascuna uscita è ovviamente possibile, ma conduce a risultati non ottimizzati, come mostra il seguente esempio:

$f_1(x,y,z) = \sum_1 (1,3,7)$ $\qquad$ $f_2(x,y,z) = \sum_1 (3,6,7)$

f_1	yz			
x	00	01	11	10
0	0	1	1	0
1	0	0	1	0

f_2	yz			
x	00	01	11	10
0	0	0	1	0
1	0	0	1	1

Tabella 4.20

La sintesi separata delle due funzioni porta a:

$f_1 = \overline{x}\, z + yz$ $\qquad$ $f_2 = yz + xy$

ciascuna di tipo **2L3G6I**, quindi con una complessità totale definibile come **2L6G12I**.

È ovvio osservare che il termine *yz* compare in entrambe le forme minime, e può quindi essere realizzato con un'unica porta AND, la cui uscita venga collegata sia a un ingresso della porta OR che realizza f_1 sia a un ingresso della porta OR che realizza f_2. In questo caso, la rete finale risulta di tipo **2L5G10I**.

Meno ovvia è la situazione seguente:

$f_1(x,y,z) = \sum_1 (1,3,7)$ $\qquad$ $f_2(x,y,z) = \sum_1 (2,6,7)$

f_1	yz			
x	00	01	11	10
0	0	1	1	0
1	0	0	1	0

f_2	yz			
x	00	01	11	10
0	0	0	0	1
1	0	0	1	1

Tabella 4.21

In questo caso, la sintesi separata porta a:

$f_1 = \overline{x}\, z + yz$ $\qquad$ $f_2 = y\overline{z} + xy$

mentre si può osservare che il mintermine *xyz*, pur non essendo un implicante primo di nessuna delle due funzioni, può essere condiviso nella sintesi seguente:

$f_1 = \overline{x}\, z + xyz$ $\qquad$ $f_2 = y\overline{z} + xyz$

nella quale il numero totale di porte AND utilizzate scende da 4 a 3, pur se una di queste aumenta da 2 a 3 il numero di ingressi. Si passa in altre parole da una sintesi disgiunta, di tipo **2L6G12I**, a una sintesi comune di tipo **2L5G11I**.

In effetti, il mintermine *xyz* risulta essere un implicante primo (l'unico) della funzione:

$f = f_1 \cdot f_2$

o come si dice un **implicante primo di più uscite**.

La sintesi di una rete a più uscite richiede dunque di individuare in primo luogo tutti gli implicanti primi a più uscite del gruppo di funzioni che si vogliono sintetizzare. Tali implicanti sono gli implicanti primi di ciascuna funzione, quelli del prodotto logico di ogni possibile coppia di funzioni, quelli del prodotto logico di ogni possibile terna di funzioni, e così via.

Una volta ottenuti tali implicanti, si tratta di individuare quelli essenziali. Si procede nel modo usuale, facendo attenzione al fatto che un implicante, presente nella mappa di una qualsiasi funzione f_i, che copra un uno coperto anche da un implicante presente in un qualsiasi prodotto di funzioni fra le quali compaia f_i, non è un implicante essenziale.

Il metodo tabellare proposto da Quine e McCluskey per individuare gli implicanti primi di più uscite è simile a quello che gestisce funzioni a uscita singola, con la differenza che ogni mintermine viene associato a un identificatore binario costituito da una stringa di n bit (dove n è il numero di uscite della rete, cioè il numero di funzioni da sintetizzare) in cui l'i-esimo bit vale 1 se tale mintermine fa parte della somma canonica di f_i, zero altrimenti.

Nel caso delle tre funzioni:

$$f_1(w,x,y,z) = \sum_1 (2,3,5,7,8,9,10,11,13,15)$$

$$f_2(w,x,y,z) = \sum_1 (2,3,5,6,7,10,11,14,15)$$

$$f_3(w,x,y,z) = \sum_1 (6,7,8,9,13,14,15)$$

la tabella diventa la seguente:

#	w	x	y	z	f_1	f_2	f_3	
2	0	0	1	0	1	1	0	✓
8	1	0	0	0	1	0	1	✓
3	0	0	1	1	1	1	0	✓
5	0	1	0	1	1	1	0	✓
6	0	1	1	0	0	1	1	✓
9	1	0	0	1	1	0	1	✓
10	1	0	1	0	1	1	0	✓
7	0	1	1	1	1	1	1	✓
11	1	0	1	1	1	1	0	✓
13	1	1	0	1	1	0	1	✓
14	1	1	1	0	0	1	1	✓
15	1	1	1	1	1	1	1	✓

Tabella 4.22.a

La ricerca di implicanti di dimensioni maggiori procede come nel caso di uscita singola, con l'accortezza di riportare in ciascuna posizione f_i dell'identificatore associato a ogni

implicante un 1 se *entrambi* i mintermini che danno luogo a tale implicante sono presenti in f_i.

Inoltre, si deve ricordare che un mintermine viene marcato come non primo (simbolo "✓") solo se viene "fuso" con un altro mintermine a formare un implicante di dimensioni maggiori, e se *l'identificatore dell'implicante è uguale a quello del mintermine*, cioè se tale mintermine non è implicante primo in *nessuna* delle funzioni in cui compare.

La cosa si ripete fino all'identificazione di tutti gli implicanti primi di più uscite. Ovviamente, un eventuale implicante che abbia tutti zeri nell'identificatore viene cancellato, poiché non risulta applicabile ad alcuna funzione.

#	*w*	*x*	*y*	*z*	f_1	f_2	f_3	
2,3	0	0	1	-	1	1	0	✓
2,6	0	-	1	0	0	1	0	✓
2,10	-	0	1	0	1	1	0	✓
8,9	1	0	0	-	1	0	1	
8,10	1	0	-	0	1	0	0	✓
3,7	0	-	1	1	1	1	0	✓
3,11	-	0	1	1	1	1	0	✓
5,7	0	1	-	1	1	1	0	
5,13	-	1	0	1	1	0	0	✓
6,7	0	1	1	-	0	1	1	✓
6,14	-	1	1	0	0	1	1	✓
9,11	1	0	-	1	1	0	0	✓
9,13	1	-	0	1	1	0	1	
10,11	1	0	1	-	1	1	0	✓
10,14	1	-	1	0	0	1	0	✓
7,15	-	1	1	1	1	1	1	
11,15	1	-	1	1	1	1	0	✓
13,15	1	1	-	1	1	0	1	
14,15	1	1	1	-	0	1	1	✓

Tabella 4.22.b

#	*w*	*x*	*y*	*z*	f_1	f_2	f_3	
2,3,6,7	0	-	1	-	0	1	0	✓
2,3,10,11	-	0	1	-	1	1	0	
2,6,10,14	-	-	1	0	0	1	0	✓
8,9,10,11	1	0	-	-	1	0	0	
3,7,11,15	-	-	1	1	1	1	0	
5,7,13,15	-	1	-	1	1	0	0	
6,7,14,15	-	1	1	-	0	1	1	
9,11,13,15	1	-	-	1	1	0	0	
10,11,14,15	1	-	1	-	0	1	0	✓

Tabella 4.22.c

#	w	x	y	z	f_1	f_2	f_3
2,3,6,7,10,11,14,15	-	-	1	-	0	1	0

Tabella 4.22.d

Una volta individuati gli implicanti primi di più uscite, si costruisce la tabella di copertura, nella quale ogni mintermine compare tante volte quante sono le funzioni in cui è presente, e gli implicanti primi sono raggruppati a seconda del loro identificatore (implicanti primi di f_1, di f_2, di f_3, di $f_1 \cdot f_2$, ...).

Si tratta dunque di inserire i seguenti implicanti primi:

$p_1 = 8,9\ (f_1 \cdot f_3)$ $p_2 = 5,7\ (f_1 \cdot f_2)$ $p_3 = 9,13\ (f_1 \cdot f_3)$

$p_4 = 7,15\ (f_1 \cdot f_2 \cdot f_3)$ $p_5 = 13,15\ (f_1 \cdot f_3)$ $p_6 = 2,3,10,11\ (f_1 \cdot f_2)$

$p_7 = 8,9,10,11\ (f_1)$ $p_8 = 3,7,11,15\ (f_1 \cdot f_2)$ $p_9 = 5,7,13,15\ (f_1)$

$p_{10} = 6,7,14,15\ (f_2 \cdot f_3)$ $p_{11} = 9,11,13,15\ (f_1)$ $p_{12} = 2,3,6,7,10,11,14,15\ (f_2)$

	2	3	5	7	8	9	10	11	13	15	2	3	5	6	7	10	11	14	15	6	7	8	9	13	14	15	
p_7					×	×	×	×																			
p_9			×	×					×	×																	f_1
p_{11}						×		×	×	×																	
p_{12}											×	×		×	×	×	×	×	×								f_2
p_2			×	×									×		×												
p_6	×	×					×	×			×	×				×	×										$f_1 \cdot f_2$
p_8		×		×				×		×		×			×		×		×								
p_1					×	×																×	×				
p_3						×			×														×	×			$f_1 \cdot f_3$
p_5									×	×														×		×	
p_{10}															×	×			×	×						×	$f_2 \cdot f_3$
p_4				×						×						×			×		×					×	$f_1 \cdot f_2 \cdot f_3$
				f_1												f_2								f_3			

Tabella 4.23.a

Il metodo prevede ora la ricerca degli **implicanti essenziali**, dove è necessario fare attenzione al fatto che alcuni implicanti possono essere essenziali per una funzione (è il caso di p_6 per f_1) e non per altre (ancora p_6 per f_2). In tabella sono indicati in neretto i mintermini associati a implicanti essenziali.

Dopo aver rimosso gli implicanti essenziali *nelle sole funzioni per le quali essi risultano tali*, la tabella si è così ridotta:

	5	7	8	9	13	15	2	3	6	10	11	14	15	13		C_P	C_I
p_7			×	×												1	3
p_9	×	×			×	×									f_1	1	3
p_{11}				×	×	×										1	3
p_{12}							×	×	×	×	×	×	×		f_2	0	1
p_2	×	×														0	1
p_6							×	×		×	×				$f_1 \cdot f_2$	0	1
p_8		×				×		×			×		×			1	3,4
p_1			×	×												0	1
p_3				×	×									×	$f_1 \cdot f_3$	1	4,5
p_5					×	×								×		1	4,5
p_{10}									×				×		$f_2 \cdot f_3$	0	1
p_4		×				×							×		$f_1 \cdot f_2 \cdot f_3$	1	4,5
			f_1						f_2					f_3			

Tabella 4.23.b

Si può ora applicare il concetto di dominanza, tenendo presente però che la dominanza fra colonne deve riferirsi a mintermini appartenenti alla stessa funzione. Per consentire di decidere quando eliminare righe o colonne, è opportuno a questo punto introdurre funzioni di costo, in particolare di **costo in termini di porte aggiuntive** (C_P) e **costo in termini di ingressi aggiuntivi** (C_I).

Ad esempio, per l'implicante primo p_{11} vale $C_P = 1$ (serve una porta AND per la sua realizzazione) e $C_I = 3$ (l'espressione di p_{11} richiede l'uso di una porta AND a 2 ingressi, più un ingresso aggiuntivo alla porta OR finale di f_1).

Per l'implicante primo p_2 vale invece $C_P = 0$ (p_2 è già stato realizzato per sintetizzare f_1, quindi una sua eventuale scelta non comporta l'uso di porte aggiuntive) e $C_I = 1$.

Per l'implicante primo p_8 vale infine $C_P = 1$ (una porta AND) e $C_I = 3{,}4$ a seconda che p_8 sia usato per sintetizzare una sola o entrambe le funzioni f_1 e f_2.

Dall'esame di Tabella 4.23.b si nota che:

- p_7 e p_1 sono uguali, ma p_1 costa meno;
- p_{12} domina p_6 e p_{10} e ha costo uguale, quindi p_6 e p_{10} possono essere eliminate;
- p_8 domina p_4 e ha costo minore, quindi p_4 può essere eliminata;
- p_9 domina p_2 ma ha costo maggiore, quindi p_2 non può essere eliminata;
- le colonne 7 e 9 di f_1 possono essere eliminate, poiché dominano le colonne 5 e 8;
- le colonne 2 e 6 di f_2 sono uguali rispettivamente alle colonne 10 e 14, che possono essere eliminate;
- le colonne 3, 11 e 15 di f_2 possono essere eliminate, poiché dominano le colonne 2 e 6.

La tabella risulta dunque essere la seguente:

	5	8	13	15	2	6	13		C_P	C_I
p_9	×		×	×				f_1	1	3
p_{11}			×	×					1	3
p_{12}					×	×		f_2	0	1
p_2	×							$f_1 \cdot f_2$	0	1
p_8				×					1	3
p_1		×							0	1
p_3			×				×	$f_1 \cdot f_3$	1	4,5
p_5			×	×			×		1	4,5
		f_1			f_2		f_3			

Tabella 4.23.c

Si nota ora che:

- ci sono colonne uniche, che danno luogo agli implicanti essenziali p_{12} e p_1 (la scelta di p_{12} completa la sintesi di f_2);
- p_{11} e p_8 sono dominati dall'implicante di uguale costo p_9, quindi possono essere eliminati;
- p_3 è dominato dall'implicante di uguale costo p_5, quindi può essere eliminato.

La nuova tabella risultante è la seguente:

	5	13	15	13		C_P	C_I
p_9	×	×	×		f_1	1	3
p_2	×				$f_1 \cdot f_2$	0	1
p_5		×	×	×	$f_1 \cdot f_3$	1	4,5
		f_1		f_3			

Tabella 4.23.d

Si deve ora scegliere forzatamente p_5 per completare la sintesi di f_3, e la tabella si riduce ulteriormente a:

	5	13	15		C_P	C_I
p_9	×	×	×	f_1	1	3
p_2	×			$f_1 \cdot f_2$	0	1
p_5		×	×	$f_1 \cdot f_3$	0	1
		f_1				

Tabella 4.23.e

La tabella risulta ciclica per considerazioni di costo, anche se si arriva facilmente a discriminare come più conveniente la scelta dei due mintermini p_2 e p_5.

4.6 SINTESI DI RETI A PIÙ LIVELLI

Le tecniche viste sin qui sono pensate per realizzare reti logiche a **due livelli** di porte (più gli eventuali negatori necessari per ottenere letterali negati delle variabili di ingresso qualora non sia disponibile una logica a "doppio binario", o *two-rail*).

Per la sintesi di reti a più livelli — ovviamente più lente ma potenzialmente più compatte — non esistono tecniche generali soddisfacenti. Si può ricorrere a manipolazioni delle forme a due livelli o ad analisi delle funzioni per identificare la possibilità di eventuali realizzazioni modulari.

4.6.1 Fattorizzazione

Questa tecnica si basa sull'utilizzo ripetuto della proprietà distributiva (da destra a sinistra) applicata a una forma minima ottenuta con i metodi prima discussi.

Consideriamo ad esempio la funzione:

$f = ac\overline{d} + ae + \overline{b}\,c\,\overline{d} + \overline{b}\,e + a\,\overline{c}\,d$

La forma minima così ottenuta richiede tre porte AND a tre ingressi, due porte AND a due ingressi e una porta OR a cinque ingressi (**2L6G18I**). Si possono fattorizzare i primi quattro termini nel modo seguente:

$f = a\cdot(c\,\overline{d} + e) + \overline{b}\cdot(c\,\overline{d} + e) + a\,\overline{c}\,d$

$f = (a + \overline{b}) \cdot (c\,\overline{d} + e) + a\,\overline{c}\,d$

Per realizzare questa funzione, che introduce però un livello aggiuntivo di porte logiche, servono ancora sei porte logiche, di cui solo una porta AND a tre ingressi, tutte le altre a due ingressi (rete di tipo **3L6G13I**).

Un altro esempio è la funzione:

$f = abdf + abef + abg + cdf + cef + cg$ (**2L7G25I**)

che può essere fattorizzata nel modo seguente:

$f = ab \cdot (df + ef) + c \cdot (df + ef) + g \cdot (ab + c)$

$f = (ab + c) \cdot (df + ef) + g \cdot (ab + c)$ (**3L8G16I**)

che richiede 8 porte invece di 7, ma con un numero di ingressi decisamente minore.

La stessa espressione logica può essere ulteriormente elaborata raccogliendo a fattor comune il termine $(ab + c)$:

$f = (ab + c) \cdot (df + ef + g)$ (**3L6G13I**)

e si può infine effettuare un raccoglimento a fattor comune del termine f fra i primi due addendi della seconda parentesi:

$f = (ab + c) \cdot (f \cdot (d + e) + g)$ (**4L6G12I**)

Un ulteriore esempio significativo è il generatore di bit di parità p. Su due bit y e z, la rete si sintetizza come:

y \ p_2 \ z	0	1
0	0	1
1	1	0

Tabella 4.24

$p_2 = y\,\overline{z} + \overline{y}\,z = y \oplus z$

Su quattro bit, la mappa di Karnaugh diventa:

p_4 / wx \ yz	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Tabella 4.25

e la funzione non è minimizzabile oltre una delle sue forme canoniche:

$p_4 = \overline{w}\,\overline{x}\,\overline{y}\,z + \overline{w}\,\overline{x}\,y\,\overline{z} + \overline{w}\,x\,\overline{y}\,\overline{z} + \overline{w}\,xyz + wx\,\overline{y}\,z + wxy\,\overline{z} + w\,\overline{x}\,\overline{y}\,\overline{z} + w\,\overline{x}\,yz$

È possibile effettuare la seguente fattorizzazione:

$p_4 = \overline{y}\,z \cdot (\overline{w}\,\overline{x} + wx) + y\,\overline{z} \cdot (\overline{w}\,\overline{x} + wx) + \overline{y}\,\overline{z} \cdot (\overline{w}\,x + w\,\overline{x}) + yz \cdot (\overline{w}\,x + w\,\overline{x})$

$p_4 = (\overline{w}\,\overline{x} + wx) \cdot (\overline{y}\,z + y\,\overline{z}) + (\overline{w}\,x + w\,\overline{x}) \cdot (\overline{y}\,\overline{z} + yz)$

Ricordando che:

$\overline{a}\,\overline{b} + ab = \overline{a\,\overline{b} + \overline{a}\,b}$

si ottiene la rete seguente, composta da tre porte XOR. In generale, la fattorizzazione della rete che genera il bit di parità di una stringa di n bit (con $n = 2^k$) richiede $n/2$ porte XOR nel primo stadio, $n/4$ nel secondo, ecc. per un totale di n-1 porte XOR e $\log_2 n$ livelli di porte XOR, pari a $3 \cdot \log_2 n$ livelli di porte AND, OR e NOT.

4.6.2 Scomposizione funzionale

Questo metodo si propone di realizzare una funzione complessa come composizione di funzioni più semplici (cioè dipendenti da un numero ridotto di variabili). Fra le varie possibilità, consideriamo la **scomposizione semplice disgiuntiva**, che costruisce una rete definita come:

$f(x_1, x_2, \ldots x_n) = f(v, z_1, z_2, \ldots z_k)$

dove valgono le seguenti relazioni:

$v = v(y_1, y_2, \ldots y_{n-k})$

$(y_1, y_2, \ldots y_{n-k}) \cup (z_1, z_2, \ldots z_k) = (x_1, x_2, \ldots x_n)$

$(y_1, y_2, \ldots y_{n-k}) \cap (z_1, z_2, \ldots z_k) = \varnothing$

Sintetizzando ad esempio la funzione:

$f(w,x,y,z) = \sum_{1} (5,6,7,13,14,15)$

si nota che è possibile effettuare la scomposizione semplice disgiuntiva data da:

$y_1 = w$ $y_2 = x$ $z_1 = y$ $z_2 = z$

Riportata in una tabella (attenzione! non è assolutamente necessario utilizzare una mappa di Karnaugh) nella quale le variabili y_i contraddistinguano le colonne e le variabili z_i le righe (tale tabella prende il nome di *mappa di scomposizione*) si nota che esistono solo *due diverse configurazioni di colonna*.

f / *yz* \ *wx*	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	1	0	1

Tabella 4.26

Questo è sempre vero per le funzioni che ammettono scomposizione semplice disgiuntiva. Infatti, intuitivamente:

- la condizione è *necessaria*, poiché se la funzione v (y_1, y_2, ... y_{n-k}) deve comparire come variabile binaria nella funzione f (v, z_1, z_2, ... z_k), essa può dare luogo solo a *due* diverse configurazioni in base al valore delle proprie variabili di ingresso;
- la condizione è *sufficiente*, poiché raggruppando i due sottoinsiemi di colonne identiche si ottiene una nuova mappa con le due colonne contraddistinte da v=0 e v=1.

Se si considera ad esempio la funzione:

$$f(a,b,c,d,e) = \sum_{1} (2,3,6,7,10,11,14,15,18,19,21,25,29)$$

e la si organizza come:

$y_i = (a,b,c)$ $z_i = (d,e)$

si nota dalla mappa di scomposizione che la proprietà è soddisfatta.

f / *de* \ *abc*	000	001	011	010	100	101	111	110
00	0	0	0	0	0	0	0	0
01	0	0	0	0	0	1	1	1
10	1	1	1	1	1	0	0	0
11	1	1	1	1	1	0	0	0

Tabella 4.27

Per sintetizzare la funzione è sufficiente scegliere una delle righe non banali (cioè non costituite da soli zeri o soli uni) per sintetizzare v (a,b,c), da inserire poi come variabile nella sintesi di f (v,d,e).

Si noti che il procedimento può essere molto lungo, poiché si tratta di considerare qualsiasi possibile combinazione delle variabili di ingresso, e che diventa anche più lungo se si considerano condizioni di indifferenza, per le quali si devono effettuare scelte mirate a ridurre a due le colonne diverse.

Ad esempio, la rete sopra considerata ammette anche la scomposizione:

$y_i = (b,c)$ $z_i = (a,d,e)$

4.7 ERRORI DI COMPORTAMENTO LOGICO

A causa dei ritardi differenti nell' "attraversamento" di porte logiche da parte dei segnali, si possono verificare malfunzionamenti transitori, costituiti da variazioni temporanee — non corrette dal punto di vista logico — dei valori delle uscite al variare degli ingressi. La complessità del problema non è affrontabile se si vuole considerare ogni possibile variazione degli ingressi: diventa invece gestibile se ci si limita a considerare reti logiche **funzionanti in modo fondamentale**, nelle quali solo un ingresso alla volta può variare il proprio valore, e la rete deve avere il tempo di assestarsi prima che avvenga una successiva variazione di ingresso.

Si distinguono due tipi di malfunzionamenti:

- **alee statiche**, che causano una temporanea variazione in un valore di uscita che dovrebbe rimanere costante poiché la funzione realizzata dalla rete combinatoria associa ai nuovi valori degli ingressi lo stesso valore di uscita; si parla di alea statica *di tipo 1* se l'uscita corretta ha valore 1, *di tipo 0* in caso contrario;
- **alee dinamiche**, che causano oscillazioni prima di stabilizzare un'uscita al valore corretto, che in questo caso è differente dal valore associato alla configurazione di ingressi precedente.

4.7.1 Alee statiche

Un esempio di rete a due livelli che presenta un'alea statica di tipo 1 è quello associato alla funzione:

$f = \overline{x}\, z + y\, \overline{z}$

Si può notare che la transizione di ingresso da 011 a 010 (nella quale — come fisicamente corretto — varia un solo valore di ingresso) quasi certamente provoca un'oscillazione dell'uscita al valore scorretto 0, poiché il negatore inserito a monte del secondo prodotto logico rallenta il passaggio a 1 della relativa porta AND rispetto al passaggio a 0 della porta AND associata al primo prodotto logico.

L'eliminazione dell'alea statica non si risolve inserendo un ritardo sul percorso dell'ingresso z verso il primo prodotto logico, poiché ciò genererebbe un'alea statica sulla transizione di ingresso opposta: da 010 a 011.

Il motivo dell'esistenza dell'alea statica è dato dal fatto che esistono **due** persorsi che collegano l'ingresso z all'uscita f attraverso altrettante porte AND, e che lungo tali percorsi l'ingresso z assume valori diversi. Poiché per qualsiasi configurazione di ingresso solo una delle due porte AND ha valore 1, i ritardi di commutazione possono comportare l'alea statica di tipo 1.

Osservando la mappa di Karnaugh della funzione, si nota che le due configurazioni in questione sono associate a *due uni non coperti dal medesimo implicante*.

f	*yz*			
x	00	01	11	10
0	0	1	1	1
1	0	0	0	1

Tabella 4.28

Questo fa sì che la sintesi minima associ all'ingresso che cambia valore fra i due uni due e solo due percorsi diversi verso l'uscita, causando la suddetta alea.

Si noti che in una rete del tipo somma di prodotti non è invece possibile che si verifichino alee statiche di tipo 0, poiché sarebbe necessario avere in ingresso alla medesima porta AND sia l'ingresso x_i che l'ingresso negato $\overline{x_i}$.

Per quanto riguarda reti a più livelli, l'identificazione delle alee statiche si può effettuare trasformando tale rete in una rete del tipo somma di prodotti mediante applicazione *dei soli teoremi di* ***associatività***, ***distributività*** *e* ***De Morgan*** (che hanno la proprietà di mantenere le alee statiche). Non devono pertanto essere applicati teoremi del tipo:

$x\overline{x} = 0$ $\qquad$ $x + \overline{x} = 1$ $\qquad$ $x + \overline{x}y = x + y$ $\qquad$ $xy + \overline{x}z + yz = xy + \overline{x}z$

Come esempio, si può considerare la funzione

$f\,(wxyz) = (x + y) \cdot (\overline{x} + \overline{z}) + xzw$

che può essere trasformata nell'espressione del tipo somma di prodotti:

$f'\,(wxyz) = x\overline{x} + \overline{x}y + x\overline{z} + y\overline{z} + xzw$

Riportata in una mappa di Karnaugh, questa forma assume l'aspetto seguente:

f	*yz*			
wx	00	01	11	10
00	0	0	1	1
01	1	0	0	1
11	1	1	1	1
10	0	0	1	1

Tabella 4.29

Come si nota, esistono tre coppie di uni adiacenti (indicate in tabella con tratto singolo) non coperti dal medesimo implicante, che possono quindi dare luogo ad alee statiche di tipo 1.

Inoltre, il prodotto logico $x\overline{x}$ potrebbe generare un'alea statica di tipo 0 qualora esistano sulla mappa due zeri adiacenti, tra i quali ci si muova variando la variabile *x*. Si può vedere che tali configurazioni esistono, e sono 0001 e 0101 (evidenziate nella mappa da tratto doppio).

Si noti che, nel caso un prodotto del tipo $x\overline{x}$ potrebbe comparire moltiplicato per altre variabili di ingresso (ad esempio, $x\overline{x}yz$), la ricerca di eventuali alee statiche di tipo 0 va limitata alle configurazioni di ingresso per le quali sia y sia z valgono 1, poiché solo in questo caso variazioni asincrone di x e di $\overline{x}$ risultano avvertibili dall'uscita.

4.7.2 Alee dinamiche

In questo caso, perché esista un'alea dinamica, devono esistere almeno tre percorsi diversi fra un ingresso e l'uscita. Per le reti riportate nella notazione a due livelli secondo le regole viste sopra, vale la proprietà che un'alea dinamica è presente se e solo se, date due configurazioni di ingresso adiacenti I_1 e I_2, differenti per il solo valore di ingresso x_i e a cui corrispondono valori *diversi* dell'uscita, nella funzione sintetizzata compare un termine prodotto che riceve in ingresso contemporaneamente x_i e $\overline{x_i}$, e nel quale tutti gli altri ingressi valgono 1 sia in I_1 sia in I_2.

Nell'esempio sopra riportato, si tratta quindi di cercare, nella mappa di Karnaugh di Tabella 4.29, coppie di caselle adiacenti associate a valori diversi dell'uscita, attraverso le quali ci si muova variando l'ingresso x: tali coppie sono evidenziate nella mappa da un tratto triplo.

4.7.3 Progetti di reti prive di alee

L'eliminazione delle alee (sia statiche sia dinamiche) si raggiunge assicurandosi che:

- non esista alcuna coppia di uni adiacenti non coperta da un implicante comune (ricorrendo quindi a sintesi ridondanti)
- non esista alcun termine prodotto che contenga la coppia di ingressi x_i e $\overline{x_i}$.

Esiste la possibilità di effettuare una sintesi con il metodo tabellare di Quine-McCluskey, a patto di utilizzare una tabella di copertura nella quale, su ogni colonna, compaiano non i mintermini della funzione, ma le coppie di uni adiacenti (che devono essere sintetizzati da sottocubi comuni per soddisfare la prima delle due condizioni sopra indicate) e gli eventuali uni isolati.

4.8 IL PROBLEMA DEI MALFUNZIONAMENTI

Una rete combinatoria può presentare malfunzionamenti dovuti anche a problemi di tipo *fisico*, cioè legati a componenti mal realizzati o guasti. Si distinguono a questo proposito:

- guasti **permanenti** (*hard*) dovuti a componenti non funzionanti;

- guasti **temporanei** (*soft*) a loro volta distinti in *transitori* (dovuti in genere a cause esterne, come particelle alfa o disturbi elettromagnetici) e *intermittenti* (spesso legati a componenti che stanno per guastarsi in modo permanente).

La ricerca dei guasti (o **testing**, inteso come l'identificazione e l'eventuale localizzazione della loro presenza) è un problema economicamente molto significativo e sempre più oneroso a causa della densità di integrazione sempre crescente, accompagnata da un aumento molto modesto del *pinout* dei dispositivi, che riduce l'osservabilità (cioè la possibilità di "vedere" lo stato logico di un punto interno alla rete mediante i valori assunti dalle uscite) e la controllabilità (ovvero la possibilità di "forzare" uno stato logico in un punto interno alla rete mediante i valori assegnati agli ingressi) dei dispositivi stessi.

Una trattazione formalizzata del testing richiede innanzitutto la definizione di un *modello di guasto*, cioè di un insieme di guasti possibili, che si intende rilevare. Il modello è il risultato di un compromesso fra generalità dei tipi di guasti considerati e gestibilità del problema di rilevarli, e si rivela tanto migliore quanto più è vicino ai problemi tipici della tecnologia implementativa considerata. Esempi di modelli di guasto sono:

- *stuck-at singolo* — una sola linea della rete è bloccata al valore logico 0 o 1;
- *stuck-at multiplo* — più linee bloccate;
- *unidirezionale* — più linee bloccate ma tutte allo stesso valore (0 o 1);
- *bridging* — accoppiamento di due linee, che assumono sempre lo stesso valore;
- *pattern sensitive* — guasto sensibile alla configurazione di ingressi presente.

Una volta scelto il modello di guasto ritenuto più conveniente, si tratta di decidere la tecnica che consenta di individuare la sequenza di collaudo (*test pattern*) che ottimizza la **copertura** del test, cioè massimizza il numero di guasti possibili che si è in grado di individuare. A questo proposito, si possono citare le tecniche seguenti:

1) controllo *esaustivo*, nel quale si applicano agli ingressi *tutte* le configurazioni possibili e si verifica la rispondenza della rete alla tabella delle verità; ha un'ottima copertura ma è ovviamente molto onerosa in termini di tempo di collaudo e di memoria richiesta al dispositivo di collaudo per memorizzare la tabella delle verità;
2) controllo a *lunghezza minima*, nel quale si riduce la lunghezza della sequenza di test ricercando le *classi* di guasti fra loro indistinguibili, che possono essere rilevate da un'unica configurazione di ingresso (ad esempio, in una porta AND, i guasti *stuck-at-0* di uno qualsiasi degli ingressi o dell'uscita);
3) controllo mediante *sensibilizzazione del percorso* (*path sensitizing*) nel quale, per ogni ingresso alla rete, si cerca un percorso verso l'uscita, che possa rendere l'uscita stessa sensibile al variare del particolare ingresso (definendo opportunamente il valore degli altri ingressi); in questo modo, è possibile valutare se variazioni dell'ingresso considerato si riflettono effettivamente in variazioni dell'uscita.

5 RETI SEQUENZIALI

5.1 DEFINIZIONE

Con il termine di **rete logica sequenziale** (spesso indicata anche con il termine *macchina sequenziale*) si definisce un circuito elettronico digitale realizzato mediante dispositivi elettronici in grado di svolgere funzioni di *porte logiche* (AND, OR, NOT, NAND, NOR, XOR) e caratterizzato dal fatto che **i valori di uscita in ogni istante dipendono non solo dai valori applicati in tale istante agli ingressi, ma anche dai valori applicati agli ingressi precedentemente** (ovvero, tali reti hanno *storia* del funzionamento passato).

Più precisamente, una rete sequenziale è definibile mediante gli elementi seguenti:

- un **alfabeto di ingresso *I*** (finito) costituito dai 2^n **simboli** che si possono presentare sulle *n* linee di ingresso;
- un **alfabeto di uscita *Z*** (finito) costituito dai 2^m **simboli** che si possono presentare sulle *m* linee di uscita;
- un **insieme di stati *S*** nei quali la rete può trovarsi; nel seguito, si considerano *macchine a stati finiti*, per le quali l'insieme *S* sia per l'appunto finito;
- una **funzione stato prossimo δ**, che per ogni stato in cui la rete possa trovarsi e per ogni simbolo di ingresso che possa venire applicato specifica *in modo deterministico* lo stato in cui la rete verrà a trovarsi. Tale funzione è definita sul prodotto cartesiano $S \times I$ come $\delta: S \times I \rightarrow S$.
- una **funzione di uscita λ** che genera il simbolo di uscita. Tale simbolo di uscita può essere associato allo stato presente e al simbolo di ingresso attuale (macchina di *Mealy*: $\lambda: S \times I \rightarrow Z$) oppure essere associato al solo stato presente (macchina di *Moore*: $\lambda: S \rightarrow Z$).

Una generica rete sequenziale è pertanto definita dalla tupla $<I, Z, S, \lambda, \delta>$ e richiede in pratica la realizzazione di *due* funzioni combinatorie (λ e δ) che dipendono dai due insiemi di valori (*I* e *S*). Inoltre, sono necessari dispositivi in grado di *memorizzare* lo stato prossimo e presentarlo come stato presente nell'intervallo di lavoro successivo della rete sequenziale.

Vale la pena sottolineare subito come le macchine alla Mealy, nelle quali le uscite sono funzione sia degli ingressi sia dello stato presente, siano adatte a situazione nelle quali gli ingressi cambiano più lentamente degli stati, altrimenti potremmo avere delle variazioni delle uscite anche mentre la macchina non cambia stato. D'altro canto, le macchine alla Mealy sono più rapide a reagire a stimoli di ingresso, e portano a sintesi con un numero di stati sicuramente non maggiore delle sintesi alla Moore.

Al contrario, le macchine alla Moore — pur essendo meno rapide e strutturalmente più grandi — hanno il pregio di reagire ai valori di ingresso e di modificare le proprie uscite

solo in corrispondenza dei cambiamenti di stato, quindi consentono di gestire anche situazioni nelle quali gli ingressi variano più rapidamente del segnale di sincronismo.

Le reti sequenziali trattate in questo capitolo si dividono in tre grandi classi:

1) le reti **sincrone**, dotate di un segnale di sincronismo (*clock*) costituito da un'onda quadra che dà il ritmo al funzionamento della rete;
2) le reti **asincrone impulsive**, prive di segnale di sincronismo, che rispondono a *impulsi* sulle linee di ingresso, cioè a doppie transizioni da uno stato inattivo a uno stato attivo e di nuovo allo stato inattivo;
3) le reti **asincrone funzionanti in modo fondamentale**, nelle quali ogni transizione degli ingressi provoca una risposta da parte della macchina. La trattazione di queste macchine è possibile solo nell'ipotesi appunto di *funzionamento in modo fondamentale*, che impone che *un solo ingresso alla volta cambi il proprio valore* (o meglio, che fra due variazioni su due ingressi passi un tempo sufficiente a consentire l'assestamento interno della rete).

5.2 ELEMENTI DI MEMORIA NELLE RETI SEQUENZIALI

Come già detto, la realizzazione di una rete sequenziale presuppone l'esistenza di un elemento circuitale capace di memorizzare lo stato assunto in base a una determinata *storia* della rete stessa, cioè alla sequenza di configurazioni di ingresso che si sono presentate.

Il più semplice elemento in grado di mantenere memoria è il cosiddetto **bistabile**, così chiamato perché presenta **due** stati stabili da un punto di vista energetico, ma chiaramente differenziati dal punto di vista dei valori quantitativi delle grandezze fisiche utilizzate nella sua realizzazione.

Una delle realizzazioni circuitalmente meno onerose di un bistabile è il **latch di tipo S-R** (**S**et-**R**eset) costituito da due porte NOR retroazionate, come rappresentato in Figura 5.1.

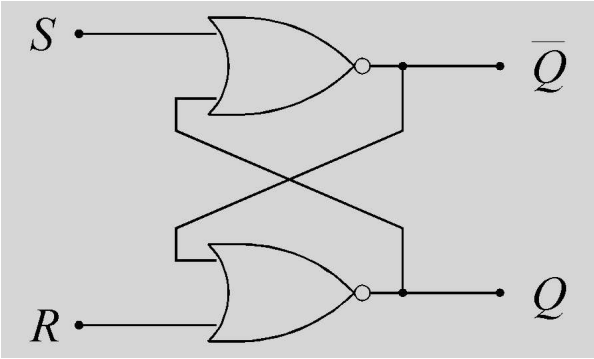


Figura 5.1 — Struttura interna del latch S-R

Il comportamento del circuito mostra come l'uscita *Q* sia in grado di memorizzare dove sia stato attivato l'ultimo segnale di ingresso: vale infatti 1 se l'ultimo segnale di ingresso portato a 1 è stato il segnale di set *S*, 0 se l'ultimo segnale di ingresso portato a 1 è stato

il segnale di reset *R*. Impredicibile è invece il comportamento del latch qualora entrambi gli ingressi *S* e *R* siano portati al valore 1 e simultaneamente riportati a 0.

Le forme d'onda associate agli ingressi *S* e *R* e alle uscite *Q* e $\overline{Q}$ mostrano inoltre come qualsiasi variazione degli ingressi si rifletta immediatamente in uscita: si enfatizza questo comportamento parlando di **transparent latch**.

Un secondo tipo di latch è il **latch S-R a ingressi sincroni** (o più precisamente controllati: ***gated***), nel quale esiste un ingresso di controllo *C* che abilita la risposta del latch ai segnali di ingresso *S* e *R*. Si ottiene dal latch S-R inserendo due porte AND prima degli ingressi *S* e *R*, in modo che tali ingressi vengano attivati solo se il segnale di controllo *C* — collegato a uno dei due ingressi di entrambe le porte AND — vale 1. Si noti come la trasparenza sia mantenuta, nel senso che qualsiasi variazione di *S* e/o *R* mentre *C* è attivo si riflette immediatamente sulle uscite.

È possibile anche avere un latch con ingressi sia sincroni sia asincroni: in questo caso, ci si riferisce agli ingressi asincroni come *preset* (per l'ingresso *S*) e *clear* (per l'ingresso *R*).

Un altro tipo di latch è il **latch di tipo D**, nel quale il valore presente all'ingresso *D* viene riportato all'uscita *Q* quando il segnale di controllo *C* è attivo. In tal caso, si parte da un latch S-R *gated*, e si collega l'ingresso *D* diritto al segnale *S* e negato al segnale *R*.

Il comportamento dei latch è definito dai seguenti tempi di funzionamento:

- t_{su} (tempo di *setup*) tempo minimo durante il quale gli ingressi di dato devono stare stabili immediatamente *prima* che il segnale di controllo *C* forzi il campionamento (cioè passi dal valore attivo al valore non attivo);
- t_h (tempo di *hold*) tempo minimo durante il quale gli ingressi di dato devono stare stabili immediatamente *dopo* che il segnale di controllo *C* abbia forzato il campionamento (cioè sia passato dal valore attivo al valore non attivo);
- t_w (*width*) ampiezza minima dell'impulso sul segnale di controllo *C*;
- $t_{PLH/PHL}$ (ritardi di propagazione) tempi necessari alle uscite per effettuare le transizioni da valore logico basso (*Low*) a valore logico alto (*High*) e viceversa.

5.3 ANALISI DELLE RETI SEQUENZIALI

Per ricavare un metodo formale di analisi del comportamento di una rete sequenziale — idoneo ad essere trasformato in un metodo di *sintesi* se ripercorso al contrario — prendiamo come esempio il circuito indicato in Figura 5.2, nel quale sono presenti due ingressi x_1 e x_2, due uscite z_1 e z_2, due latch di tipo S-R le cui uscite sono definite y_1 e y_2.

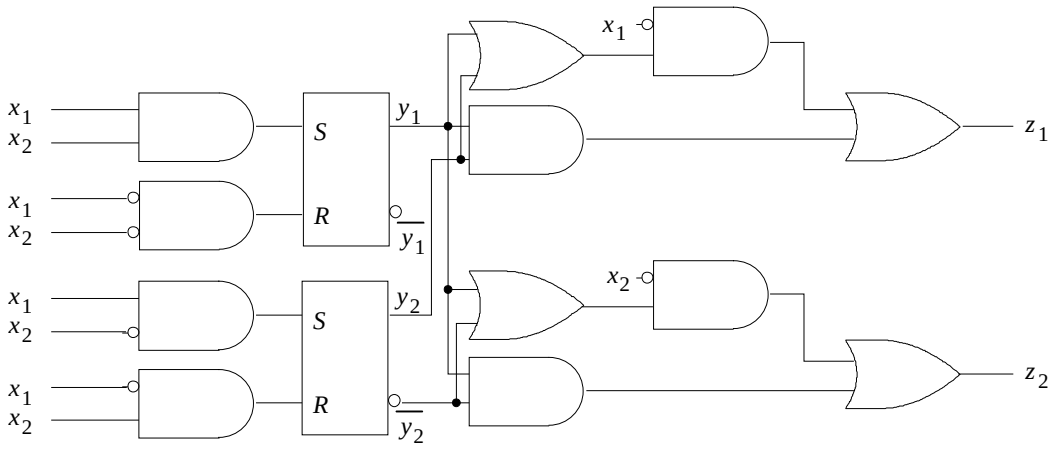


Figura 5.2 — Esempio di rete sequenziale

Le equazioni logiche che legano i segnali della rete sono le seguenti:

$S_1 = x_1 x_2$ $\qquad R_1 = \overline{x_1}\ \overline{x_2}$ $\qquad S_2 = x_1\ \overline{x_2}$ $\qquad R_2 = \overline{x_1}\ x_2$

$z_1 = \overline{x_1}\ (y_1 + y_2) + y_1 y_2$ $\qquad z_2 = \overline{x_2}\ (y_1 + \overline{y_2}\ ) + y_1\ \overline{y_2}$

Come si può vedere, le uscite della rete sono una funzione combinatoria degli ingressi e delle uscite dei latch, e possono pertanto essere descritte con i metodi già discussi per le reti combinatorie. In modo analogo, gli ingressi dei latch sono funzioni combinatorie degli ingressi della rete. Ciò che costituisce un aspetto innovativo è il legame fra le uscite dei latch e gli ingressi della rete, poiché tale legame introduce il concetto di "memoria" tipico della rete sequenziale.

Supponiamo ad esempio che in un certo istante la configurazione associata agli ingressi x_1 e x_2 e alle uscite dei latch y_1 e y_2 sia 0000, e che l'ingresso x_1 passi a 1. In questo caso, anche l'ingresso S_2 al secondo latch passa a 1, e ciò comporta che l'uscita y_2 di tale latch debba a sua volta passare a 1. In altre parole, lo stato 1000 è uno stato *instabile* per la rete sequenziale (dovuto alla funzione sequenziale del latch) destinato a trasformarsi nello stato stabile 1001.

La cosa può essere ripetuta per tutte le configurazioni di x_1, x_2, y_1 e y_2, come riportato in Tabella 5.1. In tale tabella, gli stati *stabili* sono rappresentati in **neretto**, mentre i numeri accanto agli stati *instabili* indicano in quali stati stabili sono destinati a trasformarsi.

	x_1	x_2	y_1	y_2	n.	z_1	z_2
0	**0**	**0**	**0**	**0**		0	1
1	**0**	**0**	**0**	**1**		1	0
2	0	0	1	0	0	1	1
3	0	0	1	1	1	1	1
4	**0**	**1**	**0**	**0**		0	0
5	0	1	0	1	4	1	0
6	**0**	**1**	**1**	**0**		1	1
7	0	1	1	1	6	1	0
8	1	0	0	0	9	0	1
9	**1**	**0**	**0**	**1**		0	0
10	1	0	1	0	11	0	1
11	**1**	**0**	**1**	**1**		1	1
12	1	1	0	0	14	0	0
13	1	1	0	1	15	0	0
14	**1**	**1**	**1**	**0**		0	1
15	**1**	**1**	**1**	**1**		1	0

Tabella 5.1

In Tabella 5.1 le transizioni fra stati instabili e stati stabili sono indicate da numeri. Un modo più comodo per indicare tali transizioni è quello di specificare, per ogni stato, il *prossimo* valore delle uscite dei latch (y_1 e y_2) solitamente indicato con lettere maiuscole (Y_1 e Y_2). Dal momento che nelle reti funzionanti in modo fondamentale gli ingressi cambiano uno alla volta e solo quando i latch hanno già stabilizzato le proprie uscite, è sufficiente indicare i prossimi valori delle sole uscite dei latch. Le uscite attuali (y_1 e y_2) prendono il nome di **variabili interne** della rete sequenziale (contrapposte alle **variabili esterne** associate agli ingressi) mentre le uscite prossime (Y_1 e Y_2) prendono il nome di **variabili dello stato prossimo**. In Tabella 5.2, si riportano i valori di queste variabili.

	x_1	x_2	y_1	y_2	Y_1	Y_2	z_1	z_2
0	0	0	0	0	**0**	**0**	0	1
1	0	0	0	1	**0**	**1**	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	1	1
4	0	1	0	0	**0**	**0**	0	0
5	0	1	0	1	0	0	1	0
6	0	1	1	0	**1**	**0**	1	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	0	1	0	1
9	1	0	0	1	**0**	**1**	0	0
10	1	0	1	0	1	1	0	1
11	1	0	1	1	**1**	**1**	1	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	1	0	0
14	1	1	1	0	**1**	**0**	0	1
15	1	1	1	1	**1**	**1**	1	0

Tabella 5.2

Un modo più usuale di indicare le stesse informazioni riportate in Tabella 5.2 è quello di adottare la **tabella delle transizioni** riportata in Tabella 5.3.a, insieme alla tabella delle uscite riportata in Tabella 5.3.b. Si tratta di tabelle nelle quali gli indici di colonna sono le configurazioni delle variabili esterne, gli indici di riga sono le configurazioni delle variabili interne attuali (cioè gli stati presenti) e i valori riportati in ciascuna casella sono — rispettivamente — i valori delle variabili di stato prossimo e i valori delle uscite.

Y_1Y_2 \ x_1x_2	00	01	11	10
00	**00**	**00**	10	01
01	**01**	00	11	**01**
11	01	10	**11**	**11**
10	00	**10**	**10**	11

Tabella 5.3.a

z_1z_2 \ x_1x_2	00	01	11	10
00	01	00	00	01
01	10	10	00	00
11	11	10	10	11
10	11	11	01	01

Tabella 5.3.b

Da uno stato stabile della tabella delle transizioni, una qualsiasi variazione degli ingressi provoca uno spostamento di colonna. Se la casella in cui ci si sposta è caratterizzata da uno stato prossimo *diverso* dallo stato presente associato alla riga della tabella in cui ci si trova, si verifica anche uno spostamento di riga per raggiungere il nuovo stato stabile. Il funzionamento della rete sequenziale è dunque associabile ai movimenti di un "punto di lavoro" che si sposta all'interno della tabella delle transizioni.

Una forma più sintetica della tabella delle transizioni è la **tabella degli stati**, nella quale ogni configurazione di stato presente (*s*) e prossimo (*S*) è rappresentata da una lettera, come indicato in Tabella 5.4.

S \ x_1x_2	00	01	11	10
A	**A**	**A**	D	B
B	**B**	A	C	**B**
C	B	D	**C**	**C**
D	A	**D**	**D**	C

Tabella 5.4

In alcuni casi — che esamineremo in seguito — può capitare che una certa configurazione di ingresso porti il punto di lavoro a uno stato prossimo (indicato nella tabella degli stati sulla stessa riga) a sua volta instabile per tale configurazione di ingressi, e che sia dunque necessaria una seconda transizione per raggiungere lo stato finale stabile. In questi casi, è possibile ricorrere anche a una descrizione in termini di **tabella di flusso**, che differisce dalla tabella degli stati per il solo fatto che associa a ogni configurazione di ingressi lo stato stabile finale, indipendentemente da eventuali stati intermedi per raggiungerlo.

L'ultimo aspetto rimasto per l'analisi della rete sequenziale è il legame fra le tabelle delle transizioni e degli stati è lo schema logico della rete stessa. Questo legame si ottiene ricavando innanzitutto le **funzioni di eccitazione**, cioè le equazioni logiche che esprimono gli ingressi dei latch in funzione degli ingressi alla rete. Per l'esempio considerato, tali funzioni sono:

$S_1 = x_1 x_2$ $\qquad$ $R_1 = \overline{x_1}\,\overline{x_2}$ $\qquad$ $S_2 = x_1\,\overline{x_2}$ $\qquad$ $R_2 = \overline{x_1}\,x_2$

Le funzioni di eccitazione sono poi utilizzate per ricavare la **tabella delle eccitazioni**, analoga alla tabella delle transizioni salvo il fatto che ogni casella contiene gli ingressi ai latch invece delle uscite prossime, come indicato in Tabella 5.5.

S_1R_1,S_2R_2 / y_1y_2	x_1x_2 = 00	01	11	10
00	01,00	00,01	10,00	00,10
01	01,00	00,01	10,00	00,10
11	01,00	00,01	10,00	00,10
10	01,00	00,01	10,00	00,10

Tabella 5.5

L'ultimo anello è il passaggio dalla tabella delle eccitazioni alla tabella delle transizioni, cioè il legame fra gli ingressi ai latch e i valori delle variabili di stato. Tale legame può essere ricavato costruendo la **tabella delle combinazioni**, che per ogni valore di ingresso a un certo tipo di latch e per ogni valore di stato presente definisce lo stato prossimo. La tabella delle combinazioni per il latch S-R è riportata in Tabella 5.6.

S	R	y	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	×
1	1	1	×

Tabella 5.6

Da questa tabella, utilizzando le tecniche della sintesi delle reti combinatorie, è possibile costruire la **funzione caratteristica** del tipo di latch considerato, che consente di ricavare la tabella delle transizioni da quella delle eccitazioni. Nel caso del latch di tipo S-R tale sintesi porta al seguente risultato:

Y / y	SR = 00	01	11	10
0	**0**	**0**	×	1
1	**1**	0	×	**1**

Tabella 5.7

che permette di ricavare la funzione caratteristica:

$$Y = S + \overline{R}\, y$$

È opportuno sottolineare come la tabella degli stati e la tabella delle eccitazioni — prima riferite all'intera rete sequenziale considerata — possano anche essere costruite per il singolo elemento di memoria. In questo caso, tali tabelle caratterizzano il comportamento dell'elemento stesso in termini di risposta alle diverse configurazioni di ingresso.

Per le operazioni di sintesi, è però più comodo costruire la **tabella di eccitazione dei bistabili** in modo leggermente diverso da quanto visto prima: la cosa più utile è infatti sapere, a partire da un certo stato presente e per raggiungere un certo stato prossimo, quale/i configurazione/i di ingresso utilizzare. Ad esempio, la Tabella 5.8 costituisce la tabella delle eccitazioni del latch S-R.

y	*Y*	*S*	*R*
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Tabella 5.8

5.4 ELEMENTI DI MEMORIA NON TRASPARENTI: I FLIP-FLOP

Oltre ai latch visti in precedenza, esistono bistabili che non hanno la proprietà della trasparenza; tali bistabili — denominati **flip-flop** — sono in grado di variare le proprie uscite solo in risposta a una transizione sul segnale di controllo (o di *clock*) oppure a una variazione di un eventuale segnale di ingresso asincrono.

L'esempio più semplice è il **flip-flop di tipo D**, costituito da due latch di tipo D in cascata (l'uscita *Q* del primo è collegata all'ingresso *D* del secondo) il primo dei quali riceve il segnale di controllo *C* negato, il secondo non negato. In questo modo, il valore presente all'ingresso *D* viene *campionato* sul fronte di salita di *C* (che provoca il disaccoppiamento fra l'ingresso e l'uscita del primo latch, denominato **master**) e passato al secondo latch (**slave**) che assesta la propria uscita in base a tale valore e la disaccoppia dall'ingresso sul fronte di discesa di *C*. In pratica, il latch master rimane trasparente fino al momento in cui il segnale *C* ha un fronte di salita, poi passa al latch slave il valore dell'ingresso *D* presente al momento in cui tale fronte si è verificato.

Un flip-flop di questo genere — denominato *edge triggered* o E-T — consente ad esempio la realizzazione di un registro a scorrimento, impossibile con semplici latch senza ricorrere a complesse tarature dei tempi di funzionamento.

È possibile costruire un flip-flop di tipo S-R a partire da un latch S-R, ma lo stesso funzionamento è garantito dal ben più diffuso **flip-flop di tipo J-K**. In tale flip-flop, l'ingresso J gioca il ruolo di S, e l'ingresso K quello di R, con l'ulteriore vantaggio di consentire la presenza di due 1 simultanei sugli ingressi J e K (in tale situazione, il flip-flop *commuta* la propria uscita, complementandola rispetto al valore precedente).

Esiste infine il **flip-flop di tipo T**, che commuta la propria uscita ad ogni attivazione del segnale di sincronismo.

Oltre ai flip-flop sopra indicati, sono disponibili anche flip-flop **non-gated**, cioè privi del segnale di controllo *C*. In tal caso, le transizioni di stato avvengono in seguito a un *impulso* (cioè un doppio cambio di valore logico) su uno dei segnali di ingresso.

Il più diffuso è il **flip-flop non-gated di tipo T**, nel quale l'uscita commuta ad ogni fronte di salita del segnale *T*.

Meno diffuso ma spesso utile è anche il **flip-flop non-gated di tipo S-R**, che modifica il proprio stato in corrispondenza a un fronte di discesa su *S* o *R*.

5.5 TABELLE DELLE ECCITAZIONI DEI BISTABILI

Riassumiamo in questa sezione le tabelle delle eccitazioni dei bistabili (latch e flip-flop) visti fin qui.

y	*Y*	*S*	*R*
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

Tabella 5.9 — Bistabile S-R

y	*Y*	*D*
0	0	0
0	1	1
1	0	0
1	1	1

Tabella 5.10 — Bistabile D

y	*Y*	*J*	*K*
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

Tabella 5.11 — Bistabile J-K

y	*Y*	*T*
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 5.12 — Bistabile T

5.6 SINTESI DELLE RETI SEQUENZIALI

La sintesi delle reti sequenziali richiede di percorrere all'indietro i passi seguiti in fase di analisi, per passare dalla descrizione del funzionamento desiderato alla realizzazione del corrispondente circuito logico.

In generale, tale sintesi prevede i passi seguenti.

1. Traduzione dalla descrizione a parole del funzionamento della rete desiderata alle *specifiche di progetto* dettagliate.
2. Realizzazione del ***diagramma degli stati***, che costituisce una rappresentazione grafica dell'evoluzione della rete, di più facile stesura e comprensione rispetto alla tabella degli stati. Durante questa fase — che per molti aspetti è la più importante dell'intero progetto — viene definito l'insieme S (e in particolare la sua cardinalità) e le funzioni λ e δ.
3. Traduzione del diagramma degli stati nella *tabella degli stati*, che contiene in forma più ordinata le stesse informazioni.
4. *Ottimizzazione* della tabella degli stati. Questo passo è mirato a ridurre la complessità del funzionamento sequenziale in vista di una più economica realizzazione finale. L'ottimizzazione può essere effettuata in base a diversi criteri, fra cui la riduzione del numero totale di stati.
5. Passaggio dalla tabella degli stati alla *tabella delle transizioni*. In questa fase, si tratta di decidere il numero di variabili di stato necessarie per rappresentare gli stati presenti nella tabella, e di assegnare a ciascuno stato una configurazione di tali variabili. Tale assegnamento può essere arbitrario o mirato a ottimizzare la successiva sintesi della rete.
6. Scelta dei bistabili da utilizzare e scrittura della *tabella delle eccitazioni*, derivata a partire dalla tabella delle transizioni e dalle tabelle delle eccitazioni tipiche dei bistabili utiizzati. Nel caso si decida di adottare come bistabili dei flip-flop gated di tipo D, questo passo diventa inutile poiché la tabella delle transizioni viene a coincidere con quella delle eccitazioni.
7. Sintesi delle funzioni combinatorie di generazione dello stato prossimo e delle uscite, effettuata utilizzando le tecniche discusse a proposito delle reti combinatorie.

Nel seguito, applichiamo i passi sopra enunciati a tre categorie di reti sequenziali, caratterizzate da diverse scelte realizzative e da diverse complessità di sintesi.

5.7 SINTESI DELLE RETI SEQUENZIALI SINCRONE

Queste reti sono basate sull'esistenza di un segnale di sincronismo (*clock*) generale, disponibile a ogni bistabile della rete. Più precisamente, le ipotesi sulle quali si fonda la realizzazione di una rete sequenziale sincrona sono le seguenti:

- il clock è costituito da un *treno di impulsi* di durata sufficiente a consentire l'esaurimento dei transitori;

- i segnali di ingresso alla rete *non variano* durante le fasi attive del clock;
- qualsiasi numero e tipo di variazioni dei segnali di ingresso è ammesso durante le fasi non attive del clock;
- i bistabili scelti sono flip-flop *gated*, quindi attivati da un ingresso di controllo *C* collegato al segnale di clock.

Un esempio di descrizione testuale di una rete sequenziale è il seguente:

> "si vuole realizzare il controllore elettronico di un semaforo posto all'incrocio fra due strade, mediante una rete sequenziale sincrona avente segnale di sincronismo (clock) con periodo di 1 minuto. La rete deve alternativamente dare il verde e il rosso alle due direzioni ogni minuto, a meno che venga premuto il pulsante di attraversamento pedonale, nel qual caso si dà simultaneamente per due minuti il rosso a entrambe le strade, poi si dà un turno di verde a ciascuna — indipendentemente dal fatto che qualcuno prema la richiesta di attraversamento pedonale — e infine si riabilita la funzione di richiesta attraversamento."

Una traduzione in specifiche di progetto della precedente richiesta può essere così effettuata:

> "la rete da realizzare ha un ingresso P corrispondente al pulsante di richiesta di attraversamento pedonale (P vale 1 quando il pulsante è premuto) e due uscite D_1 e D_2 corrispondenti alle luci del semaforo nella direzione delle due strade (D_i vale 0 quando deve essere accesa la luce rossa, 1 quando deve essere accesa la verde). Finché $P = 0$, D_1 e D_2 si alternano ad ogni periodo del clock ai valori 0 e 1; quando P diventa 1, si forzano entrambe D_1 e D_2 a 0 per due periodi di clock, quindi si riprende l'alternanza di rosso e verde interrotta per almeno un turno prima di riconsiderare il valore di P".

A questo punto, è necessario decidere se si intende sintetizzare la rete sequenziale come **macchina di Mealy** — nella quale le uscite sono funzione sia degli ingressi sia delle variabili di stato — oppure come **macchina di Moore** — nella quale le uscite sono funzione delle sole variabili di stato.

Nel primo caso, il diagramma degli stati viene costruito associando i valori delle uscite alle *transizioni* (cioè agli archi del diagramma), mentre nel secondo caso, il diagramma viene costruito associando i valori delle uscite agli *stati* (cioè ai nodi del diagramma).

5.7.1 Sintesi come macchina di Mealy

Questo tipo di sintesi non è una scelta corretta: le variazioni degli ingressi — in questo caso, il pulsante di richiesta pedonale P — possono verificarsi con frequenza superiore al tempo di variazione del segnale di sincronismo, che avviene ogni minuto. Si procede comunque alla sintesi per mostrarne i passi fondamentali, ma si rimanda alla successiva sintesi alla Moore per la soluzione corretta.

Il diagramma degli stati può essere costruito a partire da uno stato caratterizzato ad esempio da $P = 0$, da $D_1D_2 = 10$, non preceduto immediatamente dalla situazione $P = 1$. Il risultato della costruzione è mostrato in Figura 5.3.

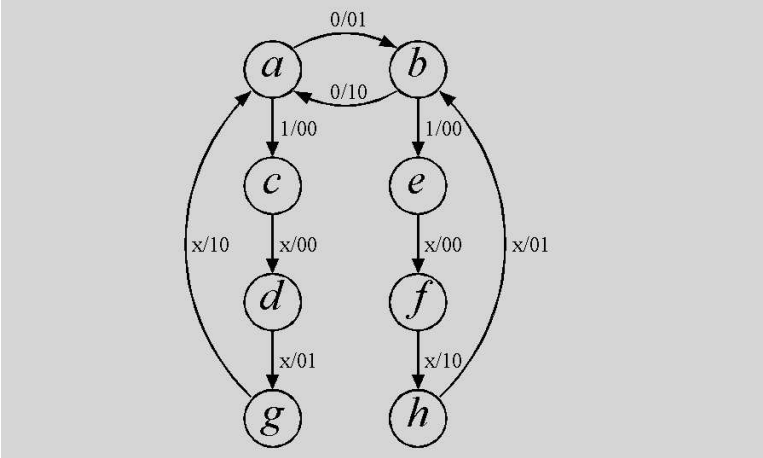


Figura 5.3 — Diagramma degli stati di una macchina alla Mealy

La tabella degli stati e delle uscite corrispondente ha il seguente aspetto:

S - D_1D_2 P

s	0	1
a	b - 01	c - 00
b	a - 10	e - 00
c	d - 00	d - 00
d	g - 01	g - 01
e	f - 00	f - 00
f	h - 10	h - 10
g	a - 10	a - 10
h	b - 01	b - 01

Tabella 5.13

Effettuando una scelta arbitraria della codifica degli 8 stati (che richiedono $\log_2 8 = 3$ variabili di stato, y_1 y_2 e y_3) si ottiene la seguente tabella delle transizioni e delle uscite:

$Y_1Y_2Y_3$ - D_1D_2 P

$y_1y_2y_3$	0	1
000	001 - 01	011 - 00
001	000 - 10	100 - 00
011	010 - 00	010 - 00
010	111 - 01	111 - 01
100	101 - 00	101 - 00
101	110 - 10	110 - 10
111	000 - 10	000 - 10
110	001 - 01	001 - 01

Tabella 5.14

La sintesi delle uscite è già effettuabile con le tecniche della sintesi combinatoria:

$$D_1 = y_1 y_3 + \overline{y_2}\, y_3\, \overline{P}$$

$$D_2 = y_2\, \overline{y_3} + \overline{y_1}\, \overline{y_3}\, \overline{P}$$

Per la sintesi degli ingressi ai bistabili, è necessario scegliere il tipo di bistabile e costruire la tabella delle eccitazioni. Nell'ipotesi di utilizzare flip-flop J-K gated, si ricava la seguente tabella delle eccitazioni:

J_1K_1, J_2K_2, J_3K_3 - D_1D_2 P		
$y_1y_2y_3$	0	1
000	0×,0×,1× - 01	0×,1×,1× - 00
001	0×,0×,×1 - 10	1×,0×,×1 - 00
011	0×,×0,×1 - 00	0×,×0,×1 - 00
010	1×,×0,1× - 01	1×,×0,1× - 01
100	×0,0×,1× - 00	×0,0×,1× - 00
101	×0,1×,×1 - 10	×0,1×,×1 - 10
111	×1,×1,×1 - 10	×1,×1,×1 - 10
110	×1,×1,1× - 01	×1,×1,1× - 01

Tabella 5.15

La sintesi combinatoria porta a costruire le funzioni logiche che legano gli ingressi dei bistabili (stato prossimo) agli ingressi della rete e alle uscite dei bistabili (stato presente):

$J_1 = y_2 \overline{y_3} + P \overline{y_2}\, y_3$ $\qquad K_1 = y_2$

$J_2 = y_1\, y_3 + P \overline{y_1}\ \overline{y_3}$ $\qquad K_2 = y_1$

$J_3 = 1$ $\qquad K_3 = 1$

5.7.2 Sintesi come macchina di Moore

In questo caso, il diagramma degli stati — ai quali vengono associati i valori delle uscite — assume l'aspetto di Figura 5.4.

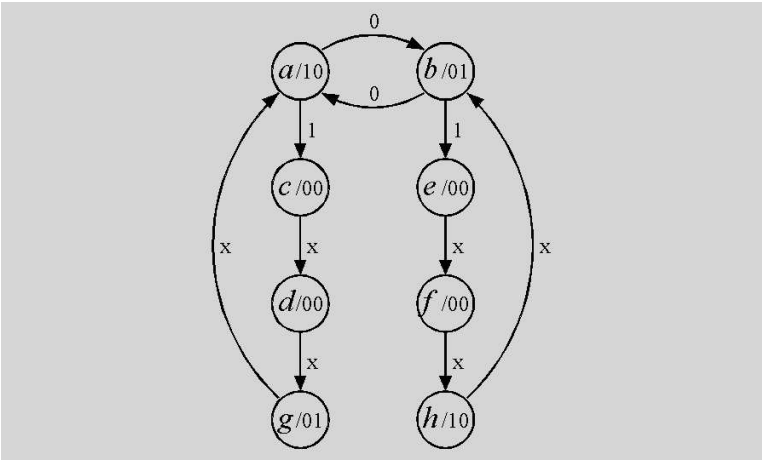


Figura 5.4 — Diagramma degli stati di una macchina alla Moore

La tabella degli stati viene costruita in modo leggermente diverso, inserendo i valori delle uscite in una colonna separata, non influenzata dai valori degli ingressi alla rete.

S	P		D_1D_2
s	0	1	
a	b	c	10
b	a	e	01
c	d	d	00
d	g	g	00
e	f	f	00
f	h	h	00
g	a	a	01
h	b	b	10

Tabella 5.16

Riassegnando le medesime configurazioni di variabili di stato utilizzate nella sintesi di Mealy, si ottiene la seguente tabella delle transizioni e delle uscite:

$Y_1Y_2Y_3$	P		D_1D_2
$y_1y_2y_3$	0	1	
000	001	011	10
001	000	100	01
011	010	010	00
010	111	111	00
100	101	101	00
101	110	110	00
111	000	000	01
110	001	001	10

Tabella 5.17

La sintesi delle uscite è già effettuabile con le tecniche della sintesi combinatoria:

$$D_1 = \overline{y_1}\,\overline{y_2}\,\overline{y_3} + y_1y_2\overline{y_3}$$

$$D_2 = \overline{y_1}\,\overline{y_2}\,y_3 + y_1\,y_2\,y_3$$

Per la sintesi degli ingressi ai bistabili, è necessario scegliere il tipo di bistabile e costruire la tabella delle eccitazioni. Adottando ancora bistabili J-K, si ottiene la seguente tabella:

J_1K_1, J_2K_2, J_3K_3	P	
$y_1y_2y_3$	0	1
000	0×,0×,1×	0×,1×,1×
001	0×,0×,×1	1×,0×,×1
011	0×,×0,×1	0×,×0,×1
010	1×,×0,1×	1×,×0,1×
100	×0,0×,1×	×0,0×,1×
101	×0,1×,×1	×0,1×,×1
111	×1,×1,×1	×1,×1,×1
110	×1,×1,1×	×1,×1,1×

Tabella 5.18

La sintesi combinatoria porta — come ovvio — agli stessi risultati del caso della macchina di Mealy:

$J_1 = y_2 \overline{y_3} + P \overline{y_2}\, y_3$ $\qquad K_1 = y_2$

$J_2 = y_1\, y_3 + P \overline{y_1}\ \overline{y_3}$ $\qquad K_2 = y_1$

$J_3 = 1$ $\qquad K_3 = 1$

5.7.3 La riduzione del numero degli stati

Un problema di fondamentale importanza per ottenere una sintesi efficace di una rete sequenziale è quello di stabilire se la tabella degli stati — ottenuta dalla costruzione del diagramma degli stati — sia o no *minima*, cioè se non esistano stati "ridondanti", inseriti in fase di stesura del diagramma.

Questo problema trova soluzione nella ricerca di eventuali **stati indistinguibili**, definiti nel modo seguente.

Date due macchine sequenziali:

$M_1 = <I, Z, S_1, \lambda_1, \delta_1>$ e $M_2 = <I, Z, S_2, \lambda_2, \delta_2>$

e indicate con $\underline{I}$ una qualsiasi sequenza di ingresso e con $\underline{Z}$ la corrispondente sequenza di uscita, lo stato $s_i \in M_1$ si definisce indistinguibile dallo stato $t_j \in M_2$ se vale la seguente relazione:

$\underline{Z}_1 = \lambda_1 (s_i , \underline{I}) = \underline{Z}_2 = \lambda_2 (t_j , \underline{I})$

per ogni possibile sequenza di ingresso $\underline{I}$.

È facile notare che la relazione di indistinguibilità — indicata con la notazione:

$$\boldsymbol{s_i \sim t_j}$$

gode delle proprietà riflessiva, simmetrica e transitiva, e costituisce pertanto una *relazione di equivalenza*. Questo consente — nell'ipotesi di considerare $M_1 \equiv M_2$, cioè di cercare relazioni di indistinguibilità fra gli stati di una stessa macchina — di definire una *partizione di equivalenza* π_e fra gli stati della macchina, costituita da blocchi di stati B_1, B_2, ... B_m caratterizzati dalle seguenti proprietà:

$B_1 \cup B_2 \cup ... \cup B_m = S$

$B_h \cap B_k = \varnothing \ \forall\ (h,k)$: $h \neq k$

iff $s_i \in B_h$ AND $s_j \in B_h \Rightarrow s_i \sim s_j$

Questa definizione di equivalenza è però di difficile applicabilità, dal momento che si tratterebbe di verificarla per *tutte* le possibii sequenze di ingresso. Esiste fortunatamente una regola (dovuta a Paull e Unger) che semplifica la ricerca di stati indistinguibili, e che — applicata a stati della medesima macchina sequenziale M — si riassume nel modo seguente:

$s_i \sim s_j$ iff:

1. $\lambda (s_i , i_x) = \lambda (s_j , i_x) \ \forall\ i_x \in I$

2. $\delta(s_i, i_x) \sim \delta(s_j, i_x) \; \forall \; i_x \in I$

La condizione 1. impone che sia uguale il primo simbolo delle due sequenze di uscita.

La condizione 2. che siano indistinguibili gli stati prossimi. Dal momento che il numero di stati è finito, la verifica (ricorsiva) della condizione 2. può essere effettuata dopo un numero finito di passi. Durante tale verifica ricorsiva, si possono infatti verificare le seguenti situazioni:

- le sequenze di uscita associate a due stati sono diverse ⇒ tutte le coppie di stati considerate nella verifica **non** sono indistinguibili;
- gli stati prossimi associati a due stati sono distinguibili ⇒ tutte le coppie di stati considerate nella verifica **non** sono indistinguibili;
- gli stati prossimi associati a due stati sono indistinguibili ⇒ tutte le coppie di stati considerate nella verifica sono indistinguibili;
- la sequenza di verifica presenta una circolarità, nel senso che due stati prossimi di cui si richiede l'indistinguibilità siano già stati incontrati come coppia della sequenza. In questo caso, si entra in un circolo apparentemente infinito, ma caratterizzato da simboli di uscita sempre identici ⇒ tutte le coppie di stati considerate nella verifica sono indistinguibili.

La verifica pratica di indistinguibilità si esegue ricorrendo alla **tabella delle implicazioni**. Supponiamo di avere ottenuto dal diagramma degli stati la seguente tabella degli stati:

S - *Z*	*I*	
s	0	1
a	*d* - 0	*b* - 0
b	*c* - 1	*a* - 0
c	*b* - 1	*e* - 0
d	*a* - 0	*b* - 0
e	*d* - 0	*a* - 0

Tabella 5.19

Si può costruire la seguente tabella delle implicazioni (dove il simbolo × indica stati *distinguibili*, e le coppie di stati la cui indistinguibilità è vincolata all'indistinguibilità degli stati prossimi riportano gli identificatori simbolici di tali stati):

b	×			
c	×	*a,e*		
d	~	×	×	
e	*a,b*	×	×	*a,d* *a,b*
	a	*b*	*c*	*d*

Tabella 5.20

La ricerca della partizione di equivalenza π_e che identifica i gruppi di stati indistinguibili è facilitata dalla costruzione del **grafo di indistinguibilità**, nel quale i nodi sono i nomi

degli stati di partenza, e gli archi connettono i nodi corrispondenti a stati indistinguibili. La macchina a numero di stati minimo viene realizzata scegliendo uno stato per ogni gruppo di stati indistinguibili identificato dalla suddetta partizione di equivalenza (per l'esempio sopra riportato, i gruppi [*ad*], [*b*], [*c*] e [*e*]).

Un procedimento analogo può essere applicato alle reti *non completamente specificate*, nelle quali non tutte le possibili configurazioni di ingresso si possono presentare in tutti gli stati. In questi casi, la tabella degli stati riporta — per ogni stato presente — i soli stati prossimi "possibili" (cioè associati a configurazioni di ingresso che si possono effettivamente presentare). Un esempio di tabella degli stati non completamente specificata è il seguente:

S - Z	x_1x_2 00	01	11	10
a	-	e - 1	a - ×	c - 1
b	-	b - ×	e - 1	f - 1
c	f - 0	-	-	b - 1
d	-	-	c - 0	-
e	a - ×	e - 0	f - ×	-
f	d - 0	f - ×	-	-

Tabella 5.21

Si può in questi casi definire la relazione di **compatibilità** fra stati, indicata come

$$s_i \vee t_j$$

che richiede che — applicando a entrambi gli stati ogni possibile sequenza di ingresso $\underline{I}$ — le sequenza di uscita della rete *ove entrambe specificate* siano identiche.

La relazione di compatibilità **non** è una relazione di equivalenza, poiché non gode della proprietà transitiva dal momento che la compatibilità stessa può essere ottenuta forzando opportunamente il valore di indifferenze sulle uscite o la compatibilità di stati prossimi.

La ricerca della macchina minima può essere fatta con il metodo visto in precedenza, salvo che in questo caso le compatibilità condizionate non si risolvono a livello di tabella di compatibilità, ma devono essere riportate sul grafo di compatibilità. In questo grafo — che può avere alcuni archi condizionati — si identificano le *classi massime* di compatibilità come poligoni chiusi, nei quali ogni vertice (stato) è connesso (compatibile) con tutti gli altri.

Ad esempio, la tabella degli stati sopra riportata dà luogo alla seguente tabella delle implicazioni:

b	*b,e* *a,e* *c,f*				
c	*b,c*	*b,f*			
d	*a,c*	×	∨		
e	×	*e,f*	*a,f*	*c,f*	
f	*e,f*	∨	*d,f*	∨	*a,d*
	a	*b*	*c*	*d*	*e*

Tabella 5.22

In questo caso, dopo la costruzione della tabella si possono eliminare le compatibilità "impossibili" (cioè condizionate a condizioni non verificate) mentre si devono mantenere esplicitate le condizioni necessarie per verificare compatibilità "possibili", dal momento che l'assenza della proprietà transitiva impedisce di associare in modo definitivo l'attributo di compatibilità a coppie di stati che richiedano condizioni di compatibilità su altre coppie di stati.

Riportata su grafo di compatibilità, la tabella sopra indicata permette di individuare le tre classi massime di compatibilità [*acdf*], [*bcef*] e [*cdef*], che come si può vedere non sono disgiunte.

Si può a questo punto definire il concetto di **copertura fra stati**, nel modo seguente.

Date due macchine sequenziali:

$M_1 = <I, Z, S, \lambda_1, \delta_1>$ e $M_2 = <I, Z, S, \lambda_2, \delta_2>$

si dice che lo stato $t_j \in M_2$ *copre* lo stato $s_i \in M_1$ se vale la seguente relazione:

$\underline{Z}_1 = \lambda_1 (s_i , \underline{I}) = \underline{Z}_2 = \lambda_2 (t_j , \underline{I})$

ogni volta che l'uscita $\underline{Z}_1 = \lambda_1 (s_i , \underline{I})$ è specificata e per ogni possibile sequenza di ingresso $\underline{I}$.

Si noti che la copertura è un concetto più restrittivo della compatibilità, poiché richiede che la macchina M_2 sia *non meno specificata* di M_1, nel senso che dove M_1 è specificata deve esserlo anche M_2.

Da quanto visto nasce il concetto di **copertura fra macchine**, che afferma che la macchina M_2 *copre* la macchina M_1 se ogni stato $s_i \in M_1$ è coperto da *almeno uno stato* $t_j \in M_2$.

La sintesi si può effettuare scegliendo le classi massime che coprono tutti gli stati e che sono vincolate a compatibilità che vengono soddisfatte dalla scelta: ad esempio, nel caso sopra indicato le due classi [*acdf*] e [*bcef*] coprono tutti gli stati e possono essere scelte, poiché la prima è condizionata a $b \vee c$ e $e \vee f$, e la seconda a $a \vee d$ e $d \vee f$. In tal caso, la sintesi procede assegnando alla classe [*acdf*] lo stato s_1 e alla classe [*bcef*] lo stato s_2.

Per ricavare la nuova tabella degli stati — nella quale lo stesso stato originario può mapparsi in più stati diversi — si applica la funzione stato prossimo "in parallelo" a tutti gli stati presenti confluiti in una medesima classe, e si verifica in quale classe vanno a

cadere i corrispondenti stati prossimi. Ad esempio, nel caso in questione la funzione stato prossimo δ (s_1,00) va applicata in parallelo ai quattro stati *a*, *c*, *d* e *f* della tabella originaria. Gli stati *a* e *d* non hanno stato prossimo per $x_1\ x_2 = 00$, mentre gli stati *c* e *f* hanno rispettivamente stato prossimo *f* e *d*. La classe di compatibilità che contiene entrambi gli stati prossimi *f* e *d* è ancora quella associata a s_1, per cui quest'ultimo è lo stato prossimo nella nuova tabella.

S - *Z*	x_1x_2			
s	00	01	11	10
s_1	s_1 - 0	s_2 - 1	s_1 - 0	s_2 - 1
s_2	s_1 - 0	s_2 - 0	s_2 - 1	s_2 - 1

Tabella 5.23

5.7.4 L'assegnamento delle configurazioni delle variabili di stato

La minimizzazione di una rete sequenziale non si ottiene soltanto riducendo il numero di stati, ma anche ottimizzando la scelta di quale configurazione delle variabili di ingresso rappresenta quale stato.

Esistono tecniche molto sofisticate e complesse per guidare in questa scelta. Si riportano in questa sezione solo alcune considerazioni di tipo generale, applicate mediante una tecnica empirica di assegnamento. In quest'ottica la scelta delle configurazioni può tenere conto degli aspetti seguenti:

1. se due stati s_i e s_j sono tali che $\delta(s_i , i_x) = \delta(s_j , i_x) = s_k$ è opportuno dare ai due stati s_i e s_j assegnamenti adiacenti, poiché comporteranno coppie di 1 o 0 adiacenti "in colonna" sulla tabella delle eccitazioni, quindi probabilità di individuare implicanti primi di dimensioni maggiori nella sintesi delle reti combinatorie che generano gli ingressi ai bistabili;
2. se uno stato s_i ha come stati prossimi s_j, s_k, ... s_m è opportuno dare agli stati s_j, s_k, ... s_m assegnamenti adiacenti, poiché comporteranno configurazioni adiacenti posizionate in caselle adiacenti "in riga" sulla tabella delle eccitazioni, quindi probabilità di individuare implicanti primi di dimensioni maggiori nella sintesi delle reti combinatorie che generano gli ingressi ai bistabili;
3. se due stati s_i e s_j sono tali che $\lambda(s_i , i_x) = \lambda(s_j , i_x) = z^*$ è opportuno dare ai due stati s_i e s_j assegnamenti adiacenti, poiché comporteranno configurazioni adiacenti in caselle adiacenti sulla tabella delle uscite della rete, quindi probabilità di individuare implicanti primi di dimensioni maggiori nella sintesi delle reti combinatorie che generano le uscite stesse.

Per individuare empiricamente l'assegnamento ottimo, si procede nel modo seguente. Dopo aver individuato le adiacenze suggerite dalle regole sopra dette, si costruisce una mappa di Karnaugh le cui coordinate siano costituite dai valori delle variabili di stato, e si inseriscono in tabella i nomi degli stati cercando di rispettare le suddette condizioni.

Consideriamo ad esempio la tabella degli stati relativa alla sintesi di una rete sequenziale come macchina di Moore (Tabella 5.16) qui riportata per convenienza:

S	P		D_1D_2
s	0	1	
a	b	c	10
b	a	e	01
c	d	d	00
d	g	g	00
e	f	f	00
f	h	h	00
g	a	a	01
h	b	b	10

Tabella 5.24

In questo caso, la regola 1. richiede le adiacenze:

a,*h* *b*,*g*

La regola 2. richiede le adiacenze:

b,*c* *a*,*e*

La regola 3. conferma le adiacenze:

a,*h* *b*,*g*

Quindi, complessivamente, deve essere:

- *a* adiacente a *e* e *h*
- *b* adiacente a *c* e *g*

La cosa è ottenibile inserendo gli stati in una mappa di Karnaugh a tre variabili nel modo seguente:

s	y_2y_3			
y_1	00	01	11	10
0	a	e	c	b
1	h	d	f	g

Tabella 5.25

Con questo assegnamento, la sintesi — sia pure non garantita minima — ottiene comunque una sostanziale semplificazione.

5.8 SINTESI DELLE RETI SEQUENZIALI ASINCRONE IMPULSIVE

In questo caso, si elimina l'ipotesi di un segnale di sincronismo generale, e si progetta una rete che cambia stato in conseguenza di un impulso su uno qualsiasi degli ingressi. Si può realizzare una macchina di Moore (uscita stabile associata allo stato) o una macchina di Mealy (l'uscita è significativa per intervalli di tempo associati agli impulsi sugli ingressi, mentre il suo valore fra un impulso e l'altro non è significativo).

Le ipotesi di corretto funzionamento richiedono che si verifichi in ingresso un solo impulso alla volta, e che la durata dell'impulso sia sufficiente a consentire alla rete di assestarsi.

Il diagramma degli stati si traccia indicando su quale linea di ingresso deve verificarsi l'impulso responsabile della transizione, e quale valore di uscita è associato alla transizione (Mealy) o allo stato (Moore).

La sintesi si effettua tenendo presente che ogni colonna della tabella degli stati — associata a una linea di ingresso sulla quale si possono manifestare impulsi — **non** implica alcuna adiacenza con le colonne accanto: si tratta infatti di:

1. prevedere un segnale di pseudo-sincronismo ottenuto come OR degli ingressi (si genera un impulso di sincronismo in corrispondenza di un qualsiasi impulso di ingresso;
2. sintetizzare separatamente ogni colonna della tabella degli stati, cioè ogni variazione degli stati dovuta a impulsi su ciascuno degli ingressi;
3. condizionare (mediante porta AND) la generazione dello stato prossimo alla effettiva presenza di un impulso sull'ingresso associato alla colonna;
4. ottenere la funzione stato prossimo globale di ciascun bistabile come somma logica (OR) delle risposte ai diversi impulsi;
5. nel caso di sintesi alla Mealy, condizionare mediante porta AND l'osservazione dell'uscita alla presenza di un impulso su uno degli ingressi.

5.9 SINTESI DELLE RETI SEQUENZIALI ASINCRONE FUNZIONANTI IN MODO FONDAMENTALE

In questo caso, la rete reagisce a qualsiasi variazione di livello degli ingressi. Si parla di *modo fondamentale* se solo un ingresso alla volta può variare, e se il tempo tra due variazioni consente alla rete di stabilizzarsi.

Il diagramma degli stati è costituito da stati dotati di *autoanelli*, cioè associati a configurazioni di ingresso che mantengono la rete stabile nello stato. La transizione può avvenire solo in risposta a configurazioni di ingresso adiacenti a quella associata all'autoanello, e una modifica delle uscite si verifica con ritardi dovuti al tipo di rete, e pertanto **non** viene esplicitata sull'arco del diagramma degli stati (dove compare un'indifferenza) ma piuttosto sull'autoanello dello stato destinatario.

Il fatto che ogni stato presente (stabile per una certa configurazione di ingresso) sia associato a stati prossimi associati a configurazioni di ingresso adiacenti e le indifferenze sulle uscite nelle transizioni portano a una tabella degli stati non completamente specificata, che richiede l'applicazione della tecnica di riduzione del numero di stati basata sulla compatibilità.

5.9.1 Le corse critiche

In alcuni casi, dopo la riduzione del numero di stati si può verificare un malfunzionamento legato ai diversi ritardi di commutazione dei bistabili, ogniqualvolta una transizione richiede a *più di una* variabile di stato di modificare il proprio valore.

Si consideri ad esempio il seguente diagramma degli stati (le uscite sono state omesse in quanto non significative in questo contesto) ottenuto a valle di una fase di minimizzazione, dove gli stati stabili sono evidenziati in neretto:

S	x_1x_2			
s	00	01	11	10
a	***a***	***a***	***a***	c
b	a	***b***	***b***	***b***
c	***c***	a	b	***c***

Tabella 5.26

Un assegnamento casuale porta alla seguente tabella delle transizioni:

Y_1Y_2	x_1x_2			
y_1y_2	00	01	11	10
00	**00**	**00**	**00**	11
01	00	**01**	**01**	**01**
11	**11**	00	01	**11**
10	×	×	×	×

Tabella 5.27

La transizione che si verifica quando a partire dallo stato $x_1x_2\ y_1y_2 = 0011$ varia a 1 l'ingresso x_2 comporta di arrivare allo stato stabile finale $x_1x_2\ y_1y_2 = 0100$.

Se y_2 si modifica per prima, si arriva allo stato intermedio $x_1x_2\ y_1y_2 = 0110$, la cui indifferenza può essere utilizzata per forzare lo stato finale corretto, mentre se si modifica prima y_1 si arriva allo stato intermedio 0101 stabile, e la rete si ferma in tale stato (scorretto).

Il primo caso è il fenomeno detto **corsa**, e si verifica tutte le volte che il passaggio dallo stato iniziale allo stato finale provoca un passaggio *temporaneo* per uno stato intermedio spurio. Non provoca malfunzionamenti permanenti nella rete, a patto che si attenda il termine del transitorio di assestamento degli stati prima fare uso dei valori assunti dalle variabili di stato.

Il secondo caso è il fenomeno detto **corsa critica**, e provoca un malfunzionamento *permanente* — quindi necessariamente da eliminare — poiché fa assumere alla rete uno stato finale diverso da quello desiderato. Può essere eliminata in vari modi, tra cui si può citare l'inserimento di *codifiche ridondanti* che consentono di associare a un medesimo stato più di una configurazione delle variabili di stato, e di utilizzare transizioni multiple per evitare il problema delle corse critiche.

In pratica, si tratta di costruire un **grafo delle adiacenze** che riporta sui nodi gli stati della rete e presenta un arco ogni volta che due stati sono collegati da una possibile transizione sul diagramma degli stati.

Se tale grafo può essere mappato su (parte di) un *n*-cubo, significa che si possono scegliere — come configurazioni delle variabili di stato associate ai vari stati — configurazioni tali da consentire movimenti fra stati sempre lungo gli archi dell'*n*-cubo, quindi con variazioni di singole variabili di stato (in questo modo, si eliminano tutte le corse, quindi anche le corse critiche).

Se tale grafo non può essere mappato su un *n*-cubo (ad esempio, esiste un triangolo che collega tre stati nel grafo) si tratta di inserire codifiche ridondanti per alcuni stati in modo tale che le codifiche associate a istanze diverse dello stesso stato costituiscano una spezzata su un *n*-cubo (cioè ci si possa muovere fra di esse con modifiche di singole variabili di stato) e che il grafo delle adiacenze risultante sia mappabile su un *n*-cubo.

Per far questo, si utilizza una mappa di Karnaugh di dimensioni sufficienti (con un numero di caselle maggiore del numero di stati) e — per tentativi — si inseriscono gli stati (e le diverse istanze dello stesso stato replicato) mantenendo le desiderate adiacenze, fino a completare i requisiti del diagramma delle adiacenze. A questo punto, si costrusce una nuova tabella degli stati, introducendo le righe corrispondenti agli stati ridondanti aggiunti e assicurandosi che non esistano mai transizioni che provocano corse.

Consideriamo ad esempio la seguente tabella degli stati:

S \ x_1x_2	00	01	11	10
a	***a***	***a***	*d*	*b*
b	*a*	*c*	***b***	***b***
c	*a*	***c***	*d*	-
d	*e*	*a*	***d***	***d***
e	***e***	*a*	-	*b*

Tabella 5.28

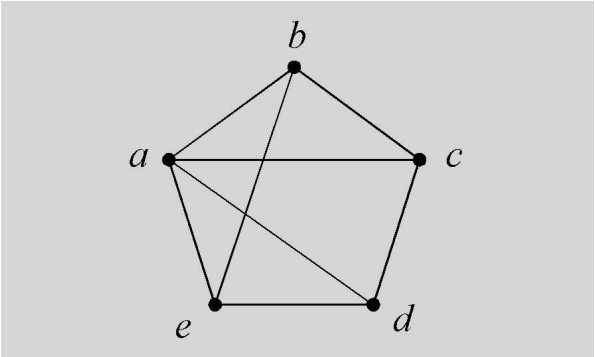


Figura 5.5

Il grafo delle adiacenze risulta quello riportato in Figura 5.5, che non può essere direttamente mappato su un *n*-cubo (esistono infatti i tre triangoli *abc*, *abe* e *ade*).

Una possibile scelta di stati ridondanti per eliminare le corse (quindi anche le corse critiche) è quella riportata nella seguente mappa di Karnaugh:

S	y_2y_3			
y_1	00	01	11	10
0	a_1	b_1	b_2	c
1	a_2	e_1	e_2	d

Tabella 5.29

Da cui si ricava la seguente tabella degli stati:

S	x_1x_2			
s	00	01	11	10
a_1	$\boldsymbol{a_1}$	$\boldsymbol{a_1}$	a_2	b_1
a_2	$\boldsymbol{a_2}$	$\boldsymbol{a_2}$	d	a_1
b_1	a_1	b_2	$\boldsymbol{b_1}$	$\boldsymbol{b_1}$
b_2	b_1	c	$\boldsymbol{b_2}$	$\boldsymbol{b_2}$
c	a_1	$\boldsymbol{c}$	d	-
d	e_2	a_2	$\boldsymbol{d}$	$\boldsymbol{d}$
e_1	$\boldsymbol{e_1}$	a_2	-	b_1
e_2	$\boldsymbol{e_2}$	e_1	-	b_2

Tabella 5.30

5.10 LE MACCHINE ITERATIVE

Una interessante applicazione delle tecniche di sintesi delle reti sequenziali è la progettazione di **macchine iterative**, cioè di *reti combinatorie* costituite da un array lineare di elementi identici (*celle*) ciascuno dei quali dotato di

- Ingressi primari $x_i = \{x_{i1}, x_{i2}, \ldots x_{iI}\}$ alla singola cella;
- Uscite primarie $z_i = \{z_{i1}, z_{i2}, \ldots z_{iU}\}$ dalla singola cella;
- Riporti in ingresso (o "ingressi laterali") $y_i = \{y_{i1}, y_{i2}, \ldots y_{iR}\}$ provenienti dalla cella precedente;
- Riporti in uscita (o "uscite laterali") $Y_i = \{Y_{i1}, Y_{i2}, \ldots Y_{iR}\}$ dirette alla cella successiva.

Il progetto di una rete di questo tipo può essere effettuato progettando la rete sequenziale che svolge *in sequenza nel tempo* le operazioni che dovrebbero essere svolte *in parallelo nello spazio* dalla rete iterativa.

Una volta progettata tale rete, la rete iterativa si ottiene "spezzando gli anelli di retroazione", ovvero eliminando i bistabili e trasformando le variabili di stato in riporti (rispettivamente, le variabili di stato presente in riporti di ingresso, le variabili di stato prossimo in riporti di uscita).

Supponiamo ad esempio di dover progettare una rete iterativa che riceva in ingresso due numeri binari $A = \{a_1, a_2, \ldots a_n\}$ e $B = \{b_1, b_2, \ldots b_n\}$ di n bit ciascuno e che generi tre uscite di confronto, definite nel modo seguente:

$z_1 = 1$ iff $A < B$

$z_2 = 1$ iff $A = B$

$z_3 = 1$ iff $A > B$

È opportuno iniziare il confronto dai bit più significativi, e progettare una rete sequenziale (di Moore) caratterizzata dalla seguente tabella degli stati:

S \ a_ib_i	00	01	11	10	z_{i1}, z_{i2}, z_{i3}
α	α	γ	α	β	0,1,0
β	β	β	β	β	0,0,1
γ	γ	γ	γ	γ	1,0,0

Tabella 5.31

Utilizzando la seguente tabella degli assegnamenti:

$Y_{i1}\ Y_{i2}$ \ a_ib_i	00	01	11	10	z_{i1}, z_{i2}, z_{i3}
$y_{i1}\ y_{i2}$ 00	00	10	00	01	0,1,0
01	01	01	01	01	0,0,1
10	10	10	10	10	1,0,0

Tabella 5.32

Questo consente di sintetizzare gli stati prossimi (cioè i riporti di uscita) nel modo seguente:

$$Y_{i1} = y_{i1} + \overline{a_i}\, b_i\, \overline{y_{i2}}$$

$$Y_{i2} = y_{i2} + a_i\, \overline{b_i}\, \overline{y_{i1}}$$

I riporti in ingresso alla prima cella della rete vanno naturalmente impostati alla configurazione associata allo stato α, cioè stato di uguaglianza dei due numeri:

$y_{11}\, y_{12} = 00$

Le uscite primarie si sintetizzano per la sola cella n-esima (meno significativa) nel modo seguente:

$$z_1 = z_{n1} = Y_{n1} = y_{n1} + \overline{a_n}\, b_n\, \overline{y_{n2}}$$

$$z_2 = z_{n2} = \overline{y_{n1}}\, \overline{y_{n2}}$$

$$z_3 = z_{n3} = Y_{n2} = y_{n2} + a_n\, \overline{b_n}\, \overline{y_{n1}}$$

6 RIFERIMENTI BIBLIOGRAFICI

1. Z.Kohavi: Switching and Finite Automata Theory. McGraw-Hill, 1970.
2. E.J.McCluskey: *Logic Design Principles*. Prentice-Hall, 1986.
3. R.H.Katz: *Contemporary Logic Design*. The Benjamin/Cummings Publ.Co., 1992.