

Alla base della gerarchia mostrata nella Figura 1.2 si trova il livello logico digitale, ovvero il vero e proprio hardware del calcolatore. In questo capitolo approfondiremo quegli aspetti della logica digitale che serviranno poi da base per lo studio dei livelli superiori. Anche se gli argomenti affrontati si trovano al confine tra l'informatica e l'elettronica il materiale è “autocontenuto”: per poterne seguire la trattazione non sono quindi necessarie specifiche conoscenze ingegneristiche o relative all’hardware.

Gli elementi base a partire dai quali sono costruiti tutti i calcolatori digitali sono incredibilmente semplici. Inizieremo il nostro studio partendo da questi componenti elementari e dalla particolare algebra a due valori (algebra di Boole) che si utilizza per analizzarli. Successivamente esamineremo alcuni circuiti fondamentali, tra cui quelli per eseguire calcoli aritmetici costruiti mediante semplici combinazioni di porte logiche. L’argomento successivo sarà l’organizzazione della memoria, ovvero com’è possibile combinare le porte logiche per memorizzare informazioni. In seguito verrà affrontato il tema delle CPU e in particolare il modo in cui CPU composte da un unico chip si interfacciano con memoria e periferiche. Nella parte finale del capitolo saranno presentati, a titolo di esempio, numerosi prodotti reali.

### 3.1 Porte logiche e algebra di Boole

I circuiti digitali possono essere costruiti combinando tra loro un piccolo numero di componenti elementari. Nei paragrafi successivi descriveremo questi elementi base, mostreremo come possono essere combinati e introdurremo una potente tecnica matematica utilizzabile per analizzarne il comportamento.

### 3.1.1 Porte logiche

Un circuito digitale è un circuito in cui sono presenti solo due valori logici<sup>1</sup>. Generalmente un valore (per esempio il numero binario 0) è rappresentato da un segnale compreso tra 0 e 0,5 volt, mentre l'altro valore (per esempio il numero binario 1) è rappresentato da un segnale compreso tra 1 e 1,5 volt; le tensioni al di fuori di questi intervalli non sono ammesse. La base hardware di tutti i calcolatori digitali è costituita da alcuni piccoli dispositivi elettronici, chiamati *porte logiche (gate)*, ciascuna delle quali calcola una diversa funzione di questi segnali.

I dettagli sul funzionamento interno delle porte logiche esulano dallo scopo di questo libro, in quanto rientrano nel campo del livello dei dispositivi, che si trova al di sotto del nostro livello 0. Ciononostante, faremo ora una piccola digressione al riguardo, senza però entrare nei dettagli più complessi, in modo da fornire un'immagine d'insieme dell'idea base. Tutta la moderna logica digitale si fonda, in ultima analisi, sul fatto che un transistor può essere costruito in modo da funzionare come un velocissimo interruttore binario. Nella Figura 3.1(a) è possibile vedere un transistor integrato in un semplice circuito. I transistor hanno tre connessioni verso il mondo esterno: il collettore, la base e l'emettitore. Quando la tensione in ingresso scende sotto un valore critico, chiamato  $V_{in}$ , il transistor viene disabilitato e si comporta come una resistenza infinita<sup>2</sup>. La conseguenza è che l'output del circuito,  $V_{out}$ , assume un valore vicino a  $V_{cc}$ : una tensione regolata esternamente che, per questo tipo di transistor, vale generalmente +5 volt. Quando, al contrario,  $V_{in}$  supera il valore critico, il transistor si attiva e si comporta come un conduttore ideale<sup>3</sup>, facendo scaricare  $V_{out}$  a terra (per convenzione, 0 volt).

È importante notare che quando  $V_{in}$  è basso,  $V_{out}$  è alto e viceversa. Questo circuito è quindi un invertitore che converte un valore logico 0 in un valore logico 1 e un valore logico 1 in un valore logico 0. La resistenza (la linea a zig zag) è necessaria per limitare la quantità di corrente nel transistor ed evitare che esso si fonda. In genere il tempo richiesto per passare da uno stato a un altro è di meno di un nanosecondo.

La Figura 3.1(b) mostra due transistor collegati in serie. Se  $V_1$  e  $V_2$  sono alte, allora entrambi i transistor saranno in conduzione e  $V_{out}$  sarà portato al valore basso, mentre, se almeno uno degli ingressi è basso, allora i transistor corrispondenti saranno disattivati e l'uscita sarà alta. In altre parole  $V_{out}$  sarà basso se e soltanto se sia  $V_1$  sia  $V_2$  sono alte.

Nella Figura 3.1(c) i due transistor sono collegati in parallelo, invece che in serie. In questa configurazione se uno dei due ingressi è alto, allora il transistor corrispondente sarà attivato e l'uscita sarà scaricata a terra, mentre, se entrambi gli ingressi sono bassi, l'uscita rimarrà alta.

Questi tre circuiti, così come altri con lo stesso funzionamento, formano le tre porte logiche più semplici, chiamate rispettivamente NOT, NAND e NOR. Dato che spesso le porte NOT vengono chiamate *invertitori*, nel corso del testo utilizzeremo i due termini in modo equivalente. Se assumiamo la convenzione di interpretare "alto" ( $V_{cc}$

volt) come un valore logico 1 e "basso" (terra) come un valore logico 0, allora è possibile esprimere il valore in uscita come una funzione dei valori in ingresso. La Figura 3.2(a)-(c) mostra i simboli usati per indicare queste tre porte logiche e il loro comportamento funzionale. Nelle figure, A e B rappresentano gli ingressi, X l'uscita e ogni riga specifica il valore in uscita data una particolare combinazione dei valori in ingresso. Se si fa passare il valore in uscita della Figura 3.1(b) in un circuito invertitore, si ottiene un nuovo circuito il cui comportamento è esattamente l'opposto di quello della portalogica NAND. In questo nuovo circuito l'uscita vale 1 se e solo se entrambi gli ingressi valgono 1; esso realizza la porta logica chiamata AND, il cui simbolo e la cui descrizione funzionale sono dati nella Figura 3.2(c). Analogamente è possibile collegare la porta logica NOR a un invertitore, in modo da ottenere un circuito la cui uscita valga 1 se almeno uno dei due ingressi vale 1, mentre valga 0 se entrambi gli ingressi valgono 0. La Figura 3.2(e) mostra il simbolo e la descrizione funzionale di questo circuito, chiamato porta logica OR. I piccoli cerchietti, utilizzati come parte dei simboli dell'invertitore, della porta NAND e della porta NOR, rappresentano sempre un'inversione.

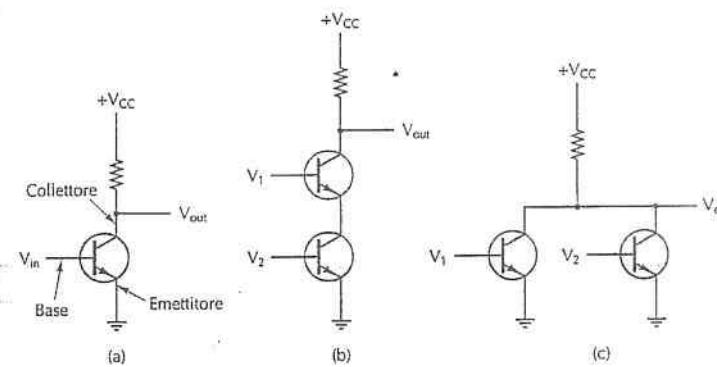


Figura 3.1 (a) Un invertitore. (b) Una porta NAND. (c) Una porta NOR.

Le porte logiche della Figura 3.2 sono i principali elementi costitutivi del livello logico digitale. Dall'analisi precedente è chiaro che le porte NAND e NOR necessitano di due transistor ciascuna, mentre le porte AND e OR ne richiedono tre; per questa ragione molti calcolatori sono basati sulle porte logiche NAND e NOR piuttosto che sulle più familiari porte AND e OR. In realtà tutte le porte logiche sono implementate in modo piuttosto diverso, ma in ogni caso le porte NAND e NOR rimangono più semplici di quelle AND e OR. Parlando di porte logiche vale la pena precisare che esse possono avere anche più di due ingressi; per fare un esempio una porta NAND potrebbe avere in teoria un numero di ingressi piuttosto elevato, anche se in pratica difficilmente ce ne sono più di otto.

<sup>1</sup> In perfetta analogia a quanto accade nella logica classica (N.d.R.).

<sup>2</sup> Circuito aperto (N.d.T.).

<sup>3</sup> Corto circuito (N.d.T.).

Dato che ci si riferirà frequentemente alle principali tecnologie di costruzione delle porte logiche ne citiamo ora le principali, anche se questo argomento fa parte del livello dei dispositivi. Le due tecnologie principali sono quella bipolare e quella MOS (*Metal Oxide Semiconductor*, “sempiconduttore metallo ossido”). Fra le bipolarie, i tipi più importanti sono la TTL (*Transistor-Transistor Logic*), per anni il motore dell’elettronica digitale, e la ECL (*Emitter-Coupled Logic*), utilizzata quando è richiesto un funzionamento a velocità molto elevate. Attualmente nei circuiti dei calcolatori si impiega largamente la tecnologia MOS.

Le porte logiche MOS sono più lente delle TTL e delle ECL, ma richiedono molta meno potenza e hanno una dimensione decisamente inferiore, permettendo così di metterne insieme un numero elevato in uno spazio limitato. Esistono diverse varietà di porte MOS, tra cui le PMOS, le NMOS e le CMOS. I transistor MOS, pur essendo costruiti in modo differente rispetto ai transistor bipolarie, sono tuttavia in grado di funzionare come degli interruttori elettronici. La maggior parte delle CPU e delle memorie moderne utilizza la tecnologia CMOS che funziona a una tensione di circa +1,5 volt. Questo è tutto ciò che occorre conoscere a proposito del livello dei dispositivi; i lettori interessati ad approfondire lo studio di questo livello possono consultare i riferimenti bibliografici suggeriti sul sito web del libro.

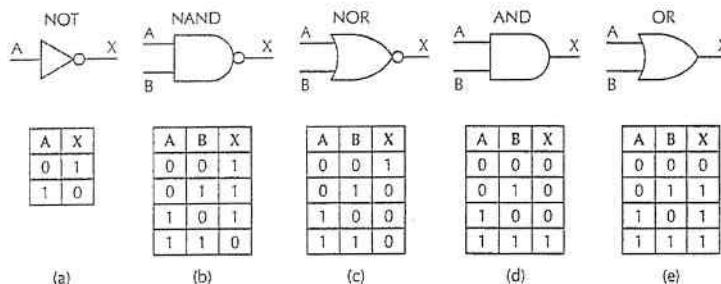


Figura 3.2 Simboli e comportamenti funzionali di cinque porte logiche elementari.

### 3.1.2 Algebra di Boole

Per poter descrivere i circuiti costruiti combinando le porte logiche occorre definire un nuovo tipo di algebra in cui variabili e funzioni possono assumere soltanto i valori 0 e 1. Questa struttura viene chiamata **algebra di Boole**, in quanto è stata sviluppata dal matematico inglese George Boole (1815-1864). Per essere precisi, ci riferiamo in realtà a un particolare tipo di algebra booleana, chiamata **algebra booleana minimale**; ciononostante il termine “algebra booleana” è utilizzato in modo talmente esteso che non faremo alcuna distinzione. Esattamente come esistono le funzioni nell’algebra ordinaria (cioè quella del liceo), esistono anche le funzioni nell’algebra booleana. Una funzione booleana ha una o più variabili di input e genera un valore di output che dipende soltanto dai valori di queste variabili. Una semplice funzione  $f$  può essere definita assumendo

che  $f(A)$  valga 1 se  $A$  vale 0 e  $f(A)$  valga 0 se  $A$  vale 1; essa corrisponde alla funzione NOT della Figura 3.2(a).

Una funzione booleana di  $n$  variabili può essere completamente descritta mediante una tabella di  $2^n$  righe; dato che esistono  $2^n$  possibili combinazioni dei valori di input, ciascuna riga può quindi indicare il valore della funzione per una data combinazione degli input. Una tabella di questo tipo è chiamata **tabella di verità**; tutte le tabelle della Figura 3.2 ne sono un esempio. Se si decide di elencare sempre le righe di una tabella di verità in ordine numerico, cioè seguendo la sequenza 00, 01, 10 e 11 nel caso di due variabili, la funzione può essere completamente descritta dal numero binario a 2<sup>n</sup> bit che si ottiene leggendo in verticale la colonna del risultato. Adottando questa convenzione, NAND è 1110, NOR è 1000, AND è 0001 e OR è 0111. Con due variabili esistono ovviamente soltanto 16 diverse funzioni booleane, corrispondenti alle 16 combinazioni possibili della stringa a 4 bit che rappresenta i risultati. Nell’algebra ordinaria esiste al contrario un numero infinito di funzioni a due variabili, nessuna delle quali può essere descritta dando una tabella che specifica i risultati per tutti i possibili valori di input. Nell’algebra ordinaria ogni variabile può infatti assumere un infinito numero di valori.

La Figura 3.3(a) mostra la tabella di verità di una funzione booleana a tre variabili:  $M = f(A, B, C)$ . Essa corrisponde alla funzione logica di maggioranza che vale 0 se la maggioranza dei suoi valori di input vale 0, mentre vale 1 se la maggioranza degli input vale 1. Anche se è possibile specificare in modo completo qualsiasi funzione booleana dandone la sua tabella di verità, all’aumentare del numero di variabili questa notazione diventa di dimensioni eccessivamente grandi. Per questo motivo si preferisce spesso adottare una notazione differente.

Per capire da dove nasce questa notazione alternativa si noti che ogni funzione booleana può essere descritta specificando quali combinazioni delle variabili di input producono, come risultato, 1. Nel caso della funzione mostrata nella Figura 3.3(a) ci sono quattro combinazioni di variabili di input che rendono  $M$  pari a 1. Per convenzione aggiungeremo una linea sopra una variabile di input per indicare che il suo valore è invertito (e non la apporremo in caso contrario). Utilizzeremo inoltre la notazione implicita della moltiplicazione, oppure un puntino, per indicare la funzione booleana AND (prodotto logico) e il simbolo + per indicare la funzione booleana OR (somma logica). Per esempio, seguendo queste convenzioni,  $\bar{A}\bar{B}C$  assume valore 1 solo quando  $A = 1$ ,  $B = 0$  e  $C = 1$ . Analogamente  $A\bar{B} + B\bar{C}$  assume valore 1 solo quando ( $A = 1$  e  $B = 0$ ) oppure ( $B = 1$  e  $C = 0$ ). Le quattro righe della Figura 3.3(a) che producono in output il bit 1 sono:  $\bar{A}\bar{B}C$ ,  $A\bar{B}C$ ,  $\bar{A}B\bar{C}$  e  $ABC$ ; dato che la funzione  $M$  è vera (cioè vale 1) se una qualsiasi di queste quattro combinazioni è vera, è possibile riscrivere in modo più compatto la tabella di verità usando la seguente notazione

$$M = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}B\bar{C} + ABC$$

Una funzione di  $n$  variabili può quindi essere descritta mediante una somma di un massimo di  $2^n$  termini, ciascuno dei quali è costituito dal prodotto logico di  $n$  variabili.

Come vedremo tra poco questa formulazione è particolarmente importante in quanto porta direttamente a un’implementazione della funzione per mezzo di porte logiche di tipo standard.

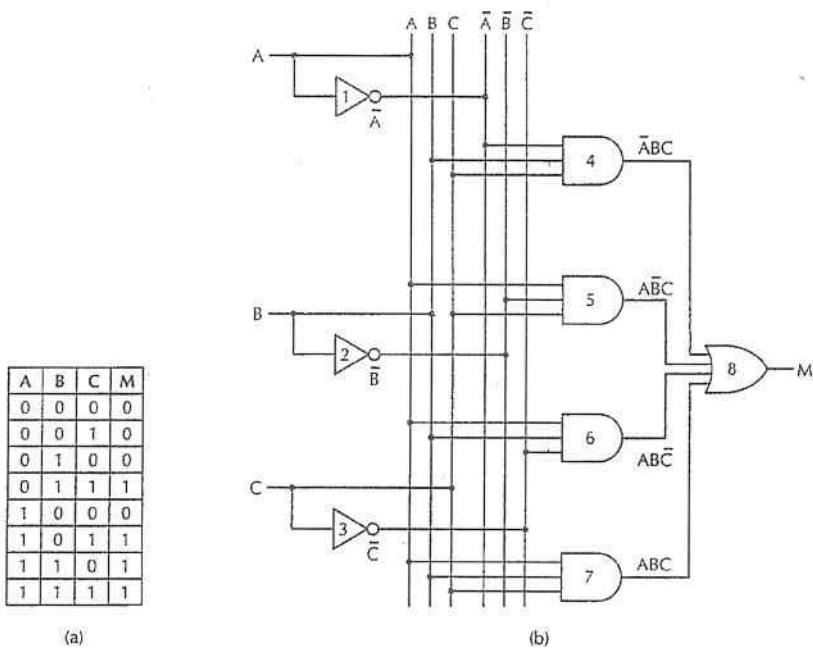


Figura 3.3 (a) Tabella di verità della funzione di maggioranza di tre variabili. (b) Un circuito per (a).

È importante tenere a mente la differenza tra una funzione booleana astratta e la sua implementazione circuitale. Una funzione booleana consiste di variabili, come A, B e C, e operatori booleani come AND, OR e NOT, ed è descritta per mezzo di una tabella di verità o una funzione booleana, per esempio

$$F = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$$

Una funzione booleana può essere implementata mediante un circuito elettronico (che, in generale, può essere realizzato in vari modi), che utilizza segnali per rappresentare le variabili di input e di output oltre ad alcune porte logiche come AND, OR e NOT. In seguito, anche se talvolta potrà essere ambiguo, utilizzeremo la notazione AND, OR e NOT per riferirci agli operatori booleani e AND, OR e NOT per riferirci alle porte logiche.

### 3.1.3 Implementazione delle funzioni booleane

Come detto precedentemente è possibile implementare facilmente una funzione booleana se la si formula come una somma di prodotti. Utilizzando la Figura 3.3 come esempio possiamo vedere come si realizza questa implementazione. Nella Figura 3.3(b) gli input A, B e C sono rappresentati sul lato sinistro, mentre l'output della funzione è mostrato a destra. Dato che è necessario disporre dei complementi (inversi) delle variabili di input,

essi sono generati intercettando gli input e facendoli passare attraverso gli invertitori indicati con i numeri 1, 2 e 3. Per rendere la figura più leggibile sono state disegnate sei linee verticali, tre delle quali connesse alle variabili di input e le restanti tre ai loro complementi. Queste linee forniscono un modo pratico per originare gli input delle porte logiche successive. Per esempio le porte 5, 6 e 7 usano A come input; in un circuito reale queste porte logiche potrebbero essere collegate direttamente ad A senza utilizzare alcun filo "verticale" intermedio.

Il circuito contiene quattro porte AND, una per ciascun termine che compone l'equazione di  $M$  (cioè una per ogni riga della tabella di verità in cui il bit della colonna risultato vale 1). Ciascuna porta AND calcola una riga della tabella di verità, com'è indicato nella figura. Il risultato finale viene poi calcolato effettuando l'OR fra tutti i prodotti.

Nel circuito mostrato nella Figura 3.3(b) è stata utilizzata una convenzione che verrà impiegata frequentemente nel corso del testo: l'incrocio fra due linee non implica alcuna connessione a meno che non sia presente un pallino nero nel punto di intersezione. Per esempio l'output della porta 3 incrocia tutte e sei le linee verticali, ma è connesso soltanto con  $\bar{C}$ . Dato che alcuni autori utilizzano convenzioni diverse, occorre prestare attenzione alla notazione impiegata.

Seguendo l'esempio della Figura 3.3 dovrebbe essere chiaro come si possa ottenere un metodo generale per implementare un circuito che realizzi una qualsiasi funzione booleana.

1. Si scrive la tabella di verità della funzione.
2. Ci si munisce di invertitori per generare la negazione di ciascun input.
3. Si utilizza una porta AND per ciascun termine il cui valore nella colonna risultato vale 1.
4. Si collegano le porte AND agli input appropriati.
5. Si connettono tutti gli output delle porte AND nella porta OR.

Anche se abbiamo dimostrato la possibile implementazione di una qualsiasi funzione booleana mediante le porte NOT, AND e OR, spesso è più vantaggioso implementare circuiti utilizzando un solo tipo di porta logica. Fortunatamente esiste un modo diretto per convertire i circuiti generati dall'algoritmo precedente in uno che utilizzi unicamente la porta NAND, oppure la NOR. L'unica cosa che serve per effettuare una simile conversione è un modo per implementare le porte NOT, AND e OR utilizzando un solo tipo di porta logica. La Figura 3.4 mostra l'implementazione di queste tre porte utilizzando soltanto porte NAND (riga in alto), oppure soltanto porte NOR (riga in basso). Questo è il modo più diretto, ma esistono tuttavia anche altri metodi di conversione.

Un metodo per implementare una funzione booleana utilizzando soltanto le porte NAND o NOR consiste nel realizzarla inizialmente mediante le porte NOT, AND e OR seguendo a precedente procedura. Successivamente occorre sostituire le porte logiche con più ingressi con circuiti equivalenti composti da porte con due soli ingressi.  $A + B + C + D$  può per esempio essere calcolata come  $(A + B) + (C + D)$  utilizzando tre porte logiche a due ingressi. Infine le porte NOT, AND e OR devono essere sostituite dai circuiti della Figura 3.4.

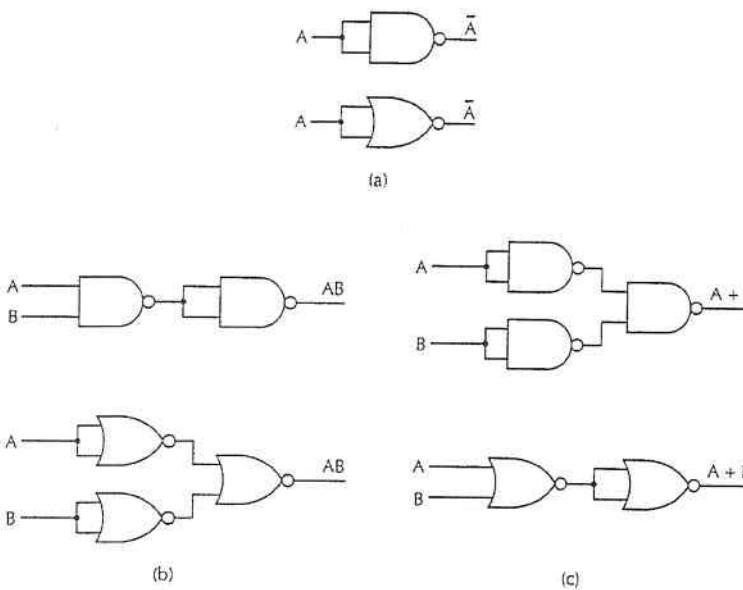


Figura 3.4 Costruzione delle porte logiche (a) NOT, (b) AND e (c) OR utilizzando soltanto porte NOR.

Questa procedura mostra che esiste sempre una possibile implementazione; tuttavia i circuiti ottenuti non sono ottimali, nel senso che non utilizzano il minimo numero possibile di porte logiche. Sia le porte NAND sia le porte NOR costituiscono un insieme di connettivi funzionalmente completo, nel senso che una qualsiasi funzione booleana può essere calcolata usando soltanto uno di questi due tipi di porte. Nessun'altra porta logica gode di questa proprietà; questo è un ulteriore motivo per il quale NAND e NOR sono spesso utilizzate come elementi base nella costruzione dei circuiti.

### 3.1.4 Equivalenza di circuiti

Spesso i progettisti cercano di ridurre il numero di porte logiche utilizzate nei loro prodotti per limitare le dimensioni delle schede con i circuiti stampati, diminuire il consumo energetico e aumentare la velocità. Per ridurre la complessità di un circuito un progettista deve trovare un altro circuito che calcoli la stessa funzione di quello originario, ma che sia composto da un numero inferiore di porte (oppure da porte più semplici, per esempio porte a due ingressi al posto di porte a quattro ingressi).

Come esempio di utilizzo di tecniche dell'algebra booleana per la ricerca di circuiti equivalenti, consideriamo il circuito e la tabella di verità per la funzione  $AB + AC$  mostrati nella Figura 3.5(a). Molte delle regole dell'algebra ordinaria valgono anche per l'algebra booleana; in particolare, usando la proprietà distributiva, è possibile fattorizzare  $AB + AC$  nella forma  $A(B + C)$ , la cui tabella di verità e il cui circuito sono mostra-

ti nella Figura 3.5(b). Dato che due funzioni sono equivalenti se e solo se hanno lo stesso valore di output per tutti i possibili input, è facile verificare dalle tabelle di verità della Figura 3.5 che  $A(B + C)$  e  $AB + AC$  sono effettivamente equivalenti. Pur essendo equivalenti il circuito della Figura 3.5(b) è chiaramente migliore di quello della Figura 3.5(a), in quanto è composto da un numero minore di porte logiche.

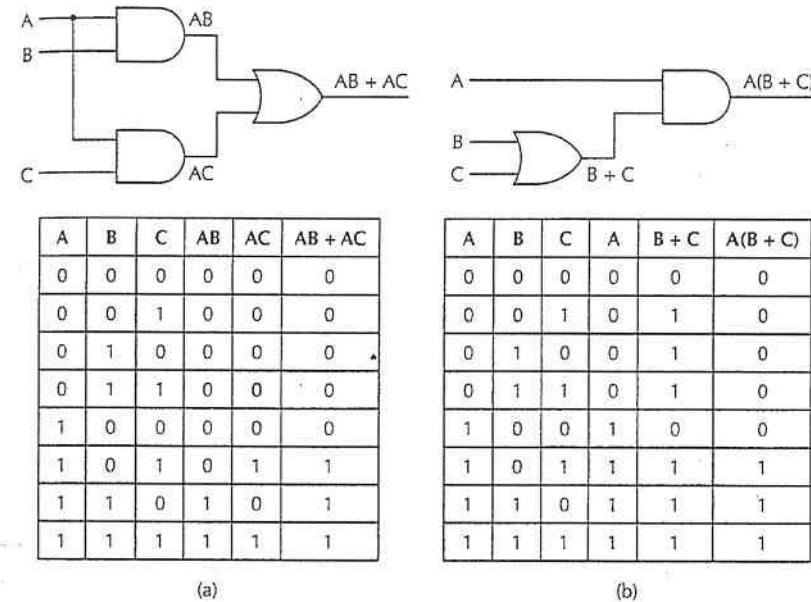


Figura 3.5 Due funzioni equivalenti. (a)  $AB + AC$ . (b)  $A(B + C)$ .

Di solito un progettista di circuiti parte da una formula e in seguito, applicando le leggi dell'algebra di Boole, cerca di semplificarla. Dalla funzione finale può infine essere costruito il circuito.

Per utilizzare questo approccio occorre conoscere alcune identità: le principali sono elencate nella Figura 3.6. È interessante notare che ciascuna legge compare in due forme, duali l'una rispetto all'altra. Scambiando AND con OR e anche 1 con 0 è possibile creare una delle due forme a partire dall'altra. È possibile verificare tutte le leggi costruendo le loro tabelle di verità; i risultati sono ragionevolmente intuitivi per tutte le leggi tranne che per la legge di De Morgan, la proprietà di assorbimento e la legge distributiva della somma rispetto al prodotto. La legge di De Morgan può inoltre essere estesa a più di due variabili, per esempio  $\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$ .

La legge di De Morgan suggerisce una notazione alternativa. La Figura 3.7(a) mostra la forma AND in cui la negazione è indicata, sia per gli input sia per gli output, median-

te i cerchietti di inversione. Una porta logica OR con gli input invertiti è quindi equivalente a una porta NAND. Dalla Figura 3.7(b), che rappresenta la forma duale della legge di De Morgan, dovrebbe essere chiaro che una porta NOR può essere disegnata come una porta AND con gli input invertiti. Negando entrambe le forme della legge di De Morgan si ottengono le Figure 3.7(c) e (d), che mostrano due rappresentazioni equivalenti delle porte logiche AND e OR. Esistono simboli analoghi per le forme a più variabili della legge di De Morgan (per esempio, una porta NAND a  $n$  input diventa una porta OR con  $n$  input invertiti).

Nome	Forma AND	Forma OR
Elemento neutro	$1A = A$	$0 + A = A$
Assorbimento	$0A = 0$	$1 + A = 1$
Idempotenza	$AA = A$	$A + A = A$
Complementazione	$\overline{A\bar{A}} = 0$	$A + \overline{A} = 1$
Proprietà commutativa	$AB = BA$	$A + B = B + A$
Proprietà associativa	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Proprietà distributiva	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Legge di assorbimento	$A(A + B) = A$	$A + AB = A$
Legge di De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\overline{B}$

Figura 3.6 Identità dell'algebra booleana.

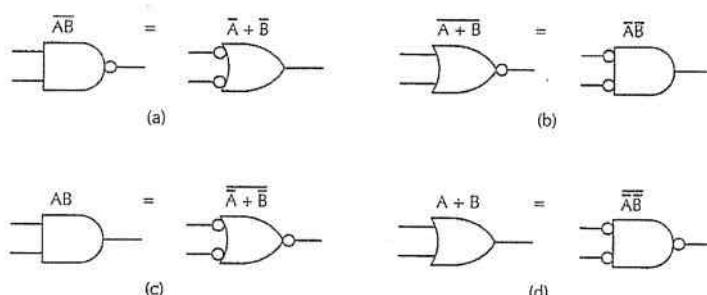


Figura 3.7 Simboli equivalenti per alcune porte logiche: (a) NAND. (b) NOR. (c) AND. (d) OR.

La conversione della tabella di verità dalla rappresentazione somma-di-prodotti alla forma pura NAND o NOR risulta agevolata dalle identità della Figura 3.7 e dalle corrispondenti versioni per le porte a più ingressi. Come esempio consideriamo la funzione XOR (OR esclusivo) della Figura 3.8(a), il cui circuito nella forma standard somma-di-prodotti è rappresentato nella Figura 3.8(b). Per convertirlo nella forma NAND bisognerebbe ridisegnare le linee che collegano l'output delle porte AND all'input della porta OR utilizzando due cerchietti d'inversione, come illustra la Figura 3.8(c); infine, utilizzando

l'equivalenza mostrata nella Figura 3.7(a) si ottiene il circuito della Figura 3.8(d). Le variabili  $\bar{A}$  e  $\bar{B}$  possono essere generate da  $A$  e  $B$  utilizzando le porte NAND o NOR con i loro input legati insieme. Si noti che i cerchietti d'inversione possono essere spostati a piacere lungo una linea, muovendoli per esempio dagli output delle porte di input della Figura 3.8(d) agli input della porta di output.

Come ultima osservazione sull'equivalenza di circuiti dimostreremo che, in modo sorprendente, una stessa porta logica reale può calcolare diverse funzioni a seconda delle convenzioni adottate. La Figura 3.9(a) mostra l'output di una certa porta,  $F$ , al variare della combinazione degli input (input e output sono indicati in volt). Se adottiamo la convenzione, chiamata logica positiva, secondo la quale 0 volt è il valore logico 0 e 1,5 volt è il valore logico 1, otteniamo la tabella di verità della Figura 3.9(b), corrispondente alla funzione AND. Se al contrario adottiamo la logica negativa, per la quale 0 volt rappresenta il valore logico 1 e 1,5 volt il valore logico 0, otteniamo la tabella di verità mostrata nella Figura 3.9(c), corrispondente alla funzione OR.

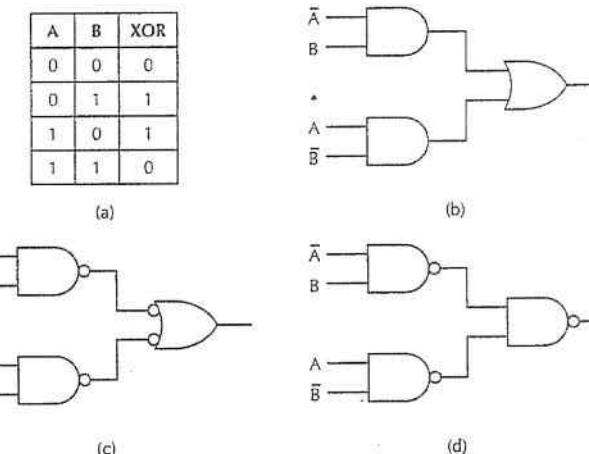


Figura 3.8 (a) Tabella di verità della funzione XOR. (b)-(d) Tre circuiti per calcolarla.

A	B	F
0	0	0
0	5	0
5	0	0
5	5	5

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

Figura 3.9 (a) Caratteristiche elettriche di un dispositivo. (b) Logica positiva. (c) Logica negativa.

La convenzione scelta per assegnare le tensioni ai valori logici riveste quindi un'importanza cruciale. Se non specificato diversamente, d'ora in poi utilizzeremo la logica positiva; i termini valore logico 1, vero e alto saranno quindi fra loro sinonimi, così come lo saranno i termini valore logico 0, falso e basso.

## 3.2 Circuiti logici digitali elementari

Nei paragrafi precedenti abbiamo visto come implementare tabelle di verità e altri semplici circuiti utilizzando singole porte logiche. Anche se una volta era comune farlo, attualmente solo pochi circuiti sono realmente costruiti “porta per porta”. Al giorno d'oggi nella costruzione di circuiti i componenti elementari sono generalmente rappresentati da moduli contenenti un certo numero di porte. Nei paragrafi successivi analizzeremo più da vicino questi blocchi elementari e vedremo come vengono utilizzati e come sono costruiti a partire da singole porte logiche.

### 3.2.1 Circuiti integrati

Le porte logiche non sono prodotte o vendute individualmente, ma in unità chiamate circuiti integrati, alle quali spesso ci si riferisce con i termini IC o chip.

Un circuito integrato è un pezzo rettangolare di silicio di varie dimensioni a seconda di quante porte sono necessarie per implementare i componenti del chip. I circuiti piccoli misurano circa  $2\text{ mm} \times 2\text{ mm}$ , mentre i grandi stampi possono arrivare a  $18\text{ mm} \times 18\text{ mm}$ . I circuiti integrati sono montati in supporti di plastica o di ceramica che possono essere molto più grandi dei circuiti ospitati nei casi in cui siano necessari molti piedini (o pin “contatto”) per collegare il chip al mondo esterno. Ogni piedino si collega all’ingresso o all’uscita di una porta del chip, oppure alla potenza o alla terra.

La Figura 3.10 mostra alcuni comuni supporti per circuiti integrati utilizzati per i chip attuali. I chip più piccoli, come per esempio quelli utilizzati per microcontrollori domestici o per le RAM, utilizzano un supporto di tipo DIP (*Dual Inline Package*, “contenitore a due file di piedini”). Il DIP è un supporto con due file di pin che viene posizionato in un alloggiamento (detto anche *socket*, “zoccolo, attacco o presa”) corrispondente sulla scheda madre. I supporti DIP più comuni hanno 14, 16, 18, 20, 22, 24, 28, 40, 64 o 68 pin. Per i grandi circuiti integrati sono spesso utilizzati supporti quadrati con perni sui quattro lati o sul fondo. Due comuni supporti per i circuiti integrati più grandi sono PGA (*Pin Grid Arrays*) e LGA (*Land Grid Arrays*). I PGA hanno i pin sul fondo del supporto che si inseriscono in un alloggiamento corrispondente sulla scheda madre. I socket PGA utilizzano spesso un meccanismo che permette l’inserimento del circuito senza sforzo; viene in seguito azionata una levetta per applicare una pressione laterale a tutti i pin del PGA, in modo che il circuito sia tenuto saldamente ancorato al suo alloggiamento. I supporti LGA hanno invece piccoli pad piatti sul fondo del chip e il socket LGA ha una copertura che si inserisce sul supporto applicando una forza rivolta verso il basso, assicurando così che tutti i pad LGA siano in contatto con i pad del suo alloggiamento.

Poiché molti supporti per circuiti integrati hanno una forma simmetrica, può essere un problema capire l’orientamento per una corretta installazione del chip. I DIP hanno tipicamente una marcatura in un angolo che va fatta coincidere con un segno corrispon-

dente sul socket DIP. I PGA hanno in genere un pin mancante. Se si tenta quindi di inserire il PGA nel suo alloggiamento in modo non corretto, il PGA non entra. Poiché i chip LGA non hanno piedini, per la corretta installazione si applica una tacca su uno o due lati del chip, che corrisponde a una tacca sul socket LGA. Il chip LGA non entrerà nel suo alloggiamento se le due tacche non corrispondono.

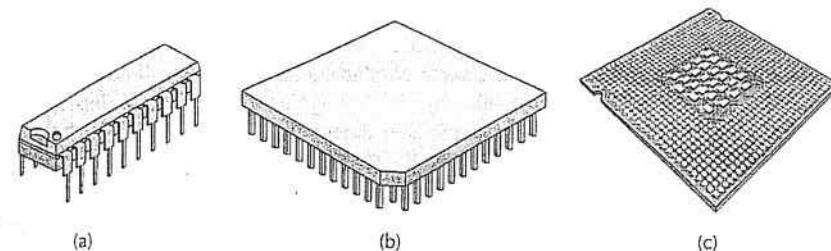


Figura 3.10 Tipologie di supporti per circuiti integrati: DIP (a), PGA (b), LGA (c).

Per i nostri scopi tutte le porte logiche considerate sono ideali, nel senso che l’output appare non appena viene applicato il voltaggio in input. In realtà i chip hanno un ritardo temporale finito, chiamato **ritardo della porta**, che comprende sia il tempo dovuto alla propagazione del segnale attraverso il chip, sia il tempo di commutazione. Generalmente questi ritardi sono compresi tra centinaia di picosecondi e pochi nanosecondi.

Lo stato dell’arte attuale della tecnologia permette di inserire più di un miliardo di transistor all’interno di un chip. Dato che ogni circuito può essere costruito mediante porte NAND, si potrebbe pensare che un produttore potrebbe realizzare un chip estremamente generalizzato contenente 500 milioni di porte NAND. Sfortunatamente però un chip di questo tipo richiederebbe 1.500.000.000 pin; considerando che la distanza standard tra i pin è di 1 mm, un chip LGA dovrebbe avere un lato di 38 m per far posto a tutti i pin, il che potrebbe porre dei limiti alla sua commercializzazione. È chiaro che l’unico modo per trarre vantaggio dallo stato attuale della tecnologia consiste nel progettare circuiti con un elevato rapporto porte/pin. Nei paragrafi successivi esamineremo alcuni semplici MSI che, per implementare una determinata funzione, combinano un buon numero di porte interne pur richiedendo un limitato numero di connessioni esterne.

### 3.2.2 Reti combinatorie

Molte applicazioni della logica digitale richiedono un circuito, chiamato **rete combinatoria**, con più input e più output, in cui gli output sono unicamente determinati dagli input. Non tutti i circuiti hanno questa proprietà; un circuito contenente elementi di memoria può per esempio generare valori che dipendono non solo dalle variabili d’ingresso, ma anche dai valori memorizzati. Un circuito che implementa una tabella di verità, come quella della Figura 3.3(a), è un tipico esempio di rete combinatoria. In questo paragrafo esamineremo le reti combinatorie utilizzate più di frequente.

### Multiplexer

In logica digitale un **multiplexer** è un circuito con  $2^n$  dati di input, un valore di output e  $n$  input di controllo; gli input di controllo permettono di selezionare uno dei dati di input, che viene “instradato” verso l’output. La Figura 3.11 mostra un diagramma di un multiplexer con otto input. Le tre linee di controllo,  $A$ ,  $B$  e  $C$ , codificano un numero a 3 bit che specifica quale delle otto linee di input deve essere instradata verso la porta OR e quindi verso l’output. Indipendentemente dal valore definito dalle linee di controllo, sette delle porte AND genereranno sempre il valore 0, mentre quella rimanente produrrà in output 0 oppure 1, a seconda del valore della linea d’ingresso selezionata. Ciascuna porta AND può essere abilitata da una diversa combinazione degli input di controllo. Il circuito del multiplexer è mostrato nella Figura 3.11; aggiungendo anche la tensione e la terra può essere confezionato in un contenitore dotato di 14 pin.

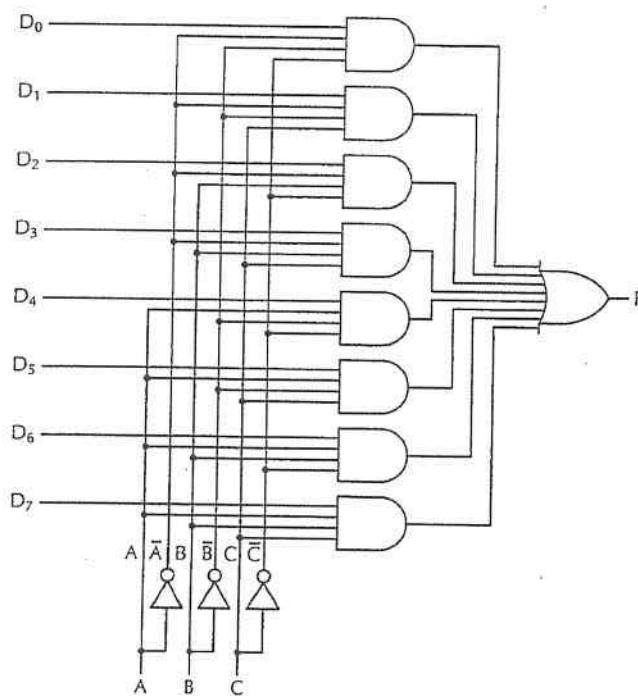


Figura 3.11 Multiplexer a otto vie.

La Figura 3.12(b) mostra come possiamo utilizzare il multiplexer per implementare la funzione di maggioranza della Figura 3.3(a). In corrispondenza di ciascuna combinazione di  $A$ ,  $B$  e  $C$  viene selezionata una delle linee dei dati di input e ciascuna di loro è

collegata a  $V_{cc}$  (valore logico 1) oppure alla terra (valore logico 0). L’algoritmo che permette di collegare gli input in modo corretto è semplice: l’input  $D_i$  è lo stesso valore presente nella riga  $i$  della tabella di verità. Nella Figura 3.3(a) le righe 0, 1, 2 e 4 valgono 0 e quindi gli input corrispondenti sono collegati alla terra; le righe rimanenti valgono invece 1 e sono quindi collegate al valore logico 1. Seguendo questa strategia è possibile implementare una qualsiasi tabella di verità mediante il chip della Figura 3.12(a).

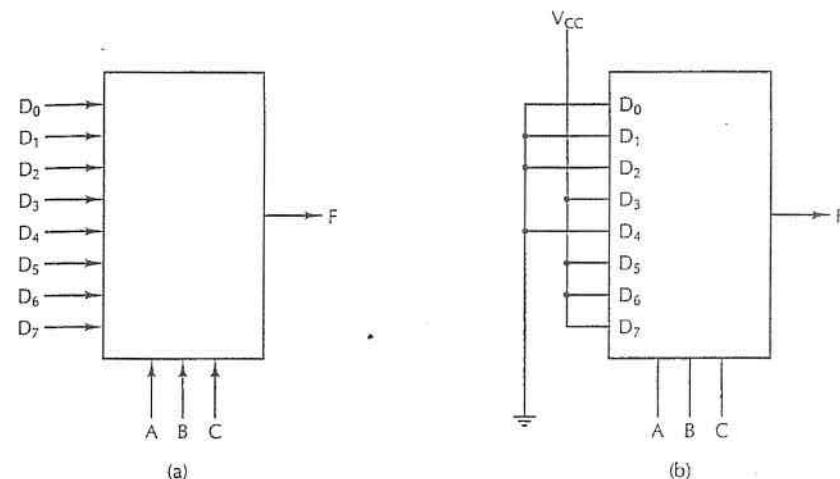


Figura 3.12 (a) Multiplexer MSI. (b) Lo stesso multiplexer per calcolare la funzione di maggioranza.

Finora abbiamo visto come utilizzare un multiplexer per selezionare un input fra alcuni valori d’ingresso e per implementare una tabella di verità. Un’altra delle possibili applicazioni è la conversione di dati da parallelo a seriale. Se si immettono 8 bit di dati nelle linee di input e se si fa variare il valore (binario) delle linee di controllo da 000 a 111, si ottengono sulla linea di output gli 8 bit in serie. Un utilizzo molto comune della conversione da parallelo a seriale lo si riscontra nelle tastiere: la pressione di ogni tasto definisce implicitamente un numero a 7 o 8 bit che deve essere inviato su un collegamento seriale, per esempio USB.

L’inverso del multiplexer è il **demultiplexer**, che redirige il suo segnale di input verso uno dei  $2^n$  output in base ai valori delle linee di controllo; se il valore binario definito dalle linee di controllo è  $k$ , viene selezionato l’output  $k$ .

### Decodificatori

Come secondo esempio analizzeremo ora un circuito, chiamato **decodificatore (decoder)**, che accetta come input un numero a  $n$  bit e lo utilizza per impostare a 1 una sola delle  $2^n$  linee di output. La Figura 3.13 mostra questo circuito nel caso  $n = 3$ .

Per capire in quali situazioni può essere utile questo circuito immaginiamo una piccola memoria di otto chip, da 256 MB ciascuno. Gli indirizzi del chip 0 variano da 0 a 256 MB, quelli del chip 1 da 256 MB a 512 MB e così via. Quando si fornisce alla memoria un indirizzo, si utilizzano i suoi 3 bit più significativi per selezionare uno degli otto chip. In riferimento al circuito della Figura 3.13 questi 3 bit corrispondono ai tre input,  $A$ ,  $B$  e  $C$ ; a seconda del loro valore solo una delle linee di output, assume il valore 1, mentre le altre rimangono a 0. Ciascuna linea di output permette poi di abilitare uno degli otto chip della memoria; dato che solo una linea di output viene impostata al valore 1, solo uno dei chip viene abilitato.

Il funzionamento del circuito della Figura 3.13 è semplice. Ciascuna porta AND ha tre input, il primo dei quali è  $A$  o  $\bar{A}$ , il secondo  $B$  o  $\bar{B}$  e il terzo  $C$  o  $\bar{C}$ , e viene abilitata da una diversa combinazione dei valori di input:  $D_0$  da  $\bar{A} \bar{B} \bar{C}$ ,  $D_1$  da  $\bar{A} \bar{B} C$  e così via.

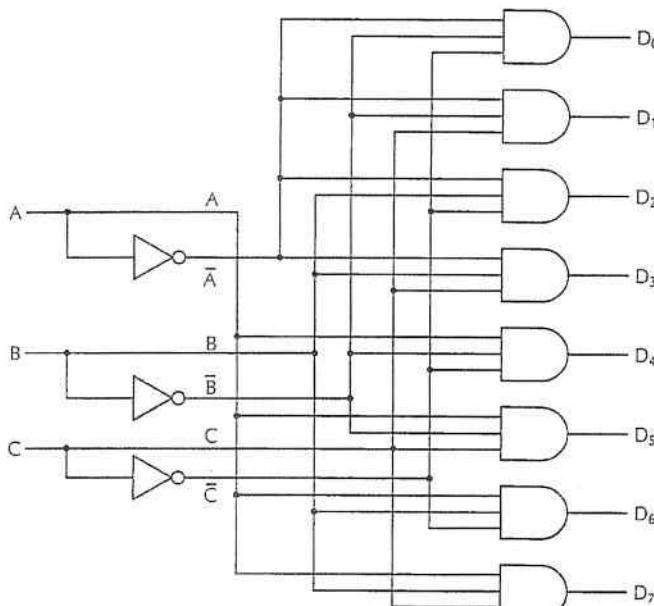


Figura 3.13 Decodificatore da 3 a 8.

### Comparatori

Un altro circuito particolarmente utile è il **comparatore**, che permette di confrontare due stringhe di bit. Il semplice comparatore mostrato nella Figura 3.14 accetta due input,  $A$  e  $B$ , ciascuno lungo 4 bit, e genera 1 se sono uguali, mentre 0 se sono diversi. Il circuito è basato sulla porta logica XOR (EXCLUSIVE OR), che produce in output un

valore 0 se i suoi input sono uguali e un valore 1 se sono diversi. Se due stringhe in ingresso sono uguali, tutte e quattro le porte XOR devono generare come risultato 0. Questi quattro segnali possono poi essere connessi a una stessa porta logica OR in modo da produrre un valore 0 quando gli input sono uguali e un valore 1 nel caso contrario. Nel nostro esempio abbiamo utilizzato una porta NOR nell'ultimo stadio del circuito in modo da invertire il risultato del test: 1 significa uguale, mentre 0 significa diverso.

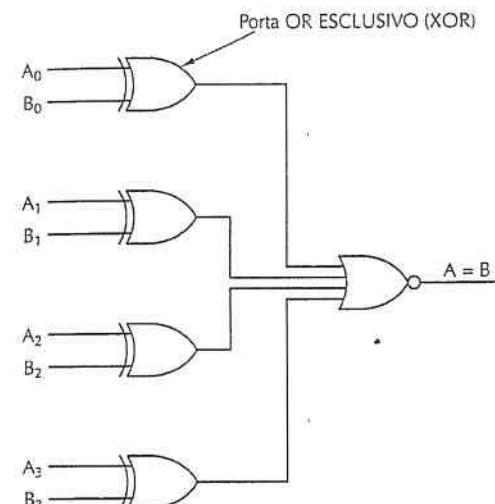


Figura 3.14 Semplice comparatore a 4 bit.

### Array logici programmabili

Un chip molto generale che permette di calcolare somme di prodotti è l'**array logico programmabile** o PLA (*Programmable Logic Array*), di cui la Figura 3.15 mostra un semplice esempio. Questo chip ha 12 ingressi e al suo interno questi valori vengono invertiti; quindi il numero totale di segnali di input diventa 24. Il cuore del circuito è costituito da una schiera di 50 porte AND che possono avere come input un qualsiasi sottosinsieme dei 24 segnali di input. Una matrice di  $24 \times 50$  bit fornita dall'utente determina le connessioni desiderate tra i segnali di input e le 50 porte AND. Ogni linea di input connessa alle 50 porte AND contiene un fusibile. Al momento della fabbricazione del chip i 1200 fusibili sono intatti; successivamente l'utente può programmare la matrice bruciando i fusibili con l'applicazione di un'alta tensione.

L'uscita del circuito consiste in 6 porte OR, che possono avere fino a 50 input, corrispondenti agli output delle porte AND. Anche in questo caso l'utente deve fornire una matrice ( $50 \times 6$ ) per specificare quali connessioni instaurare. Il chip ha 20 pin in tutto, 12 per gli input, 6 per gli output, e 2 per la tensione e la terra.

Come esempio di utilizzo di un circuito PLA consideriamo nuovamente il circuito mostrato nella Figura 3.3(b), che ha tre input, quattro porte AND, una porta OR e tre inverteri. Il nostro PLA, dopo aver impostato inizialmente le connessioni in modo appropriato, può calcolare la stessa funzione usando tre dei suoi 12 input, quattro delle 50 porte AND e una delle sue 6 porte OR. Le quattro porte AND dovrebbero calcolare rispettivamente  $ABC$ ,  $ABC$ ,  $ABC$  e  $ABC$ , e la porta OR dovrebbe accettare in input questi quattro prodotti. In realtà lo stesso PLA potrebbe essere utilizzato per calcolare quattro funzioni di analoga complessità.

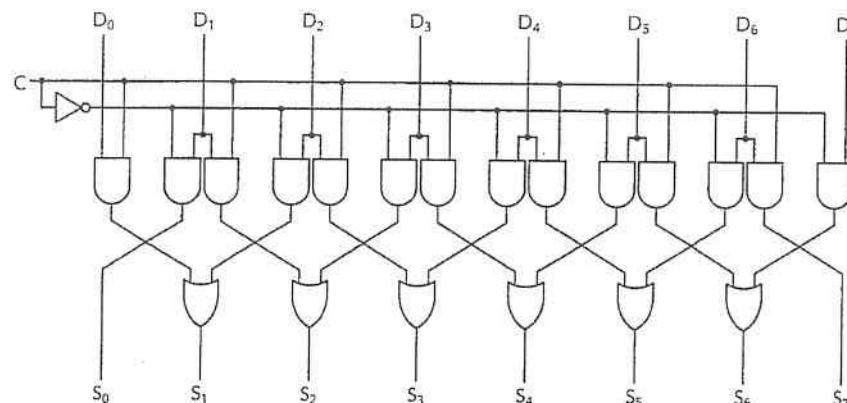


Figura 3.15 Array logico programmabile a 12 input e 6 output. I quadratini rappresentano i fusibili che possono essere bruciati per determinare la funzione da calcolare. I fusibili sono disposti in due matrici: quella superiore per le porte AND, quella inferiore per le porte OR.

Nel caso di funzioni semplici il numero di variabili rappresenta un fattore vincolante, mentre per funzioni più complesse il limite potrebbe essere rappresentato dal numero di porte AND o OR.

Anche se i PLA *programmabili sul campo*, come quello appena descritto, vengono ancora utilizzati, oggi, in molte applicazioni, si preferisce impiegare dei PLA costruiti ad hoc. Questi ultimi sono più economici di quelli programmabili e, in caso di grandi volumi, vengono progettati sulla base delle specifiche del cliente.

Alla luce della presentazione dei tre modi diversi per implementare la tabella di verità della Figura 3.3(a) possiamo passare a un confronto. Utilizzando componenti SSI sono necessari 4 chip; una soluzione alternativa, mostrata nella Figura 3.12(b), prevede invece di utilizzare un unico multiplexer di tipo MSI. Infine l'ultima possibilità consiste nell'usare un quarto di un chip PLA. Se occorre implementare più funzioni il chip PLA è ovviamente più efficiente rispetto agli altri due metodi; nel caso invece di circuiti semplici, i chip SSI e MSI sono preferibili in quanto più economici.

### 3.2.3 Circuiti per l'aritmetica

È arrivato ora il momento di passare dai circuiti generici alle reti combinatorie. Ricordiamoci che le reti combinatorie hanno uscite che sono funzioni dei loro ingressi, ma i circuiti utilizzati per fare calcoli aritmetici non hanno questa proprietà. Inizieremo con un semplice registro a scorrimento (*shifter*) a 8 bit, continueremo guardando com'è costruito un sommatore e infine esamineremo le unità aritmetico-logiche che svolgono un ruolo centrale all'interno di tutti i calcolatori.

#### Registri a scorrimento

Il primo circuito aritmetico MSI che analizziamo ha otto input e otto output (vedi la Figura 3.16). Gli input sono collegati alle linee  $D_0, \dots, D_7$ , mentre l'output, corrispondente all'input traslato di un bit, risulta disponibile sulle linee  $S_0, \dots, S_7$ . La linea di controllo,  $C$ , ha valore 0 se lo spostamento deve avvenire verso sinistra, e 1 in caso contrario. Nel caso di uno spostamento a sinistra si inserisce uno 0 nel bit 7, e nel caso di uno shift a destra si inserisce uno 0 nel bit 0.

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

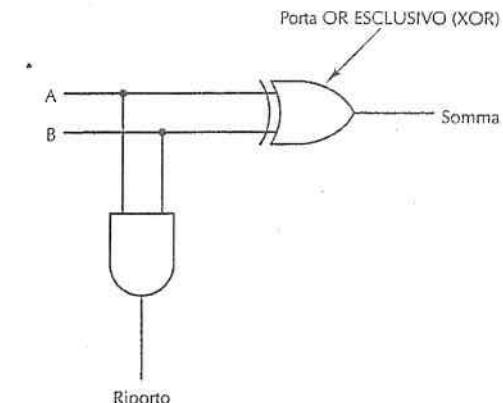


Figura 3.16 Registro a scorrimento a sinistra di 1 bit.

Per comprendere il funzionamento del circuito si noti che per tutti i bit c'è una coppia di porte AND, fatta eccezione per le porte che si trovano alle estremità. Quando  $C = 1$  la porta che si trova a destra in ciascuna coppia viene abilitata, lasciando passare il bit corrispondente verso l'output. Dato che la porta AND è collegata all'input della porta OR alla sua destra, si ottiene uno scorrimento verso destra. Quando  $C = 0$  è la porta AND di sinistra in ciascuna coppia a essere abilitata, producendo uno spostamento verso sinistra.

#### Sommatori

È quasi impossibile immaginare un calcolatore che non sia in grado di eseguire le somme. Per questo motivo un circuito che effettui l'addizione è una parte di essenziale

importanza per qualsiasi CPU. La Figura 3.17(a) mostra la tabella di verità per la somma di interi a 1 bit, in cui sono presenti due output: la somma dei due input,  $A$  e  $B$ , e il riporto da sommare alla successiva posizione (a sinistra). La Figura 3.17(b) rappresenta un circuito, chiamato **half adder** (cioè *semisommatore*, non tenendo conto del riporto in ingresso), capace di calcolare il bit della somma e il bit del riporto.

Questo circuito è in grado di sommare correttamente i bit meno significativi di due stringhe binarie, ma non è in grado di eseguire correttamente la somma degli altri bit, dato che non riesce a gestire il riporto che arriva dalle posizioni precedenti. Per far ciò è necessario un **sommatore** come quello mostrato nella Figura 3.18. Osservando attentamente il circuito dovrebbe apparire chiaro che un sommatore è costruito a partire da due semisommatori. La linea di output *Sum* vale 1 se una o tre delle linee  $A$ ,  $B$  e  $Carry$  in valgono 1. *Carry out* vale 1 se sia  $A$  sia  $B$  valgono 1 (input di sinistra della porta OR) oppure se uno dei due vale 1 e anche *Carry in* vale 1. L'unione dei due semisommatori permette di generare in output sia il bit della somma sia il bit del riporto.

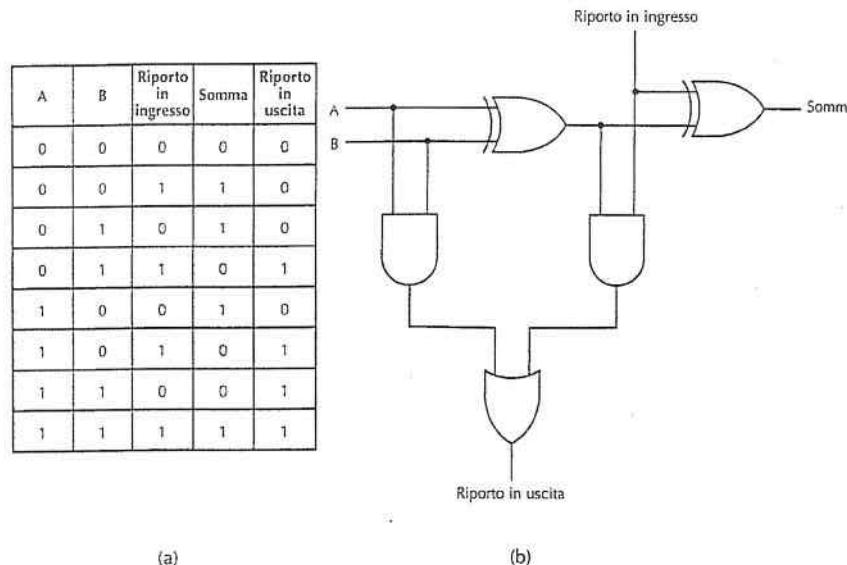


Figura 3.17 (a) Tabella di verità di un sommatore. (b) Circuito di un sommatore.

Per esempio, per generare un sommatore di due parole a 16 bit occorre replicare il circuito della Figura 3.18(b) per 16 volte. Per un dato bit il riporto in uscita viene utilizzato come riporto in entrata per il bit alla sua sinistra. Il riporto in entrata del bit all'estremità destra è collegato al valore 0. Questo tipo di sommatore è chiamato **sommatore a propagazione di riporto**, dato che nel caso peggiore, sommando 1 a 111...111 (bina-

rio), la somma non può essere completata finché il riporto non si sia propagato lungo tutta la parola, dal bit all'estremità destra fino a quello all'estremità sinistra. Esistono anche altri tipi di sommatori che non hanno questo ritardo e risultano dunque più veloci; per questo motivo vengono generalmente preferiti rispetto a quelli a propagazione di riporto.

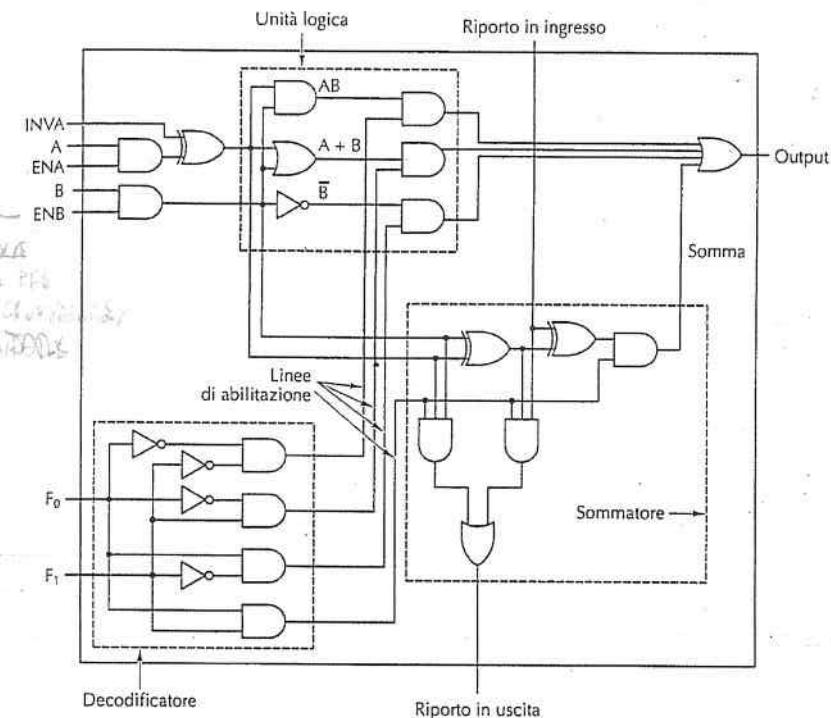


Figura 3.18 ALU a 1 bit.

Per fare un esempio di sommatore più veloce immaginiamo di dividere un sommatore a 32 bit in uno per i 16 bit meno significativi e in un altro per i 16 bit più significativi. Nel momento in cui comincia l'addizione quest'ultimo non può ancora iniziare a lavorare poiché è obbligato ad attendere il valore del riporto per un tempo pari a 16 somme di singoli bit. Consideriamo però la seguente modifica. Invece di avere un singolo sommatore per i bit più significativi ne creiamo due identici in parallelo duplicando l'hardware corrispondente. Il circuito così modificato consiste in tre sommatori a 16 bit: uno per i bit meno significativi e due per quelli più significativi,  $U_0$  e  $U_1$ , funzionanti in parallelo. Nel sommatore  $U_0$  viene collegato il valore 0 come riporto, mentre nel sommatore

$U_1$  viene collegato il valore 1. Tutti e tre i sommatori possono ora cominciare nello stesso momento, anche se soltanto uno dei due sommatori  $U_0$  e  $U_1$  sarà corretto. Dopo aver atteso il completamento delle prime 16 somme di singoli bit si conoscerà il valore del riporto da aggiungere al sommatore dei 16 bit più significativi. Grazie a questo valore è possibile selezionare il sommatore corretto fra  $U_0$  e  $U_1$ . Questo trucco permette di dimezzare il tempo richiesto dall'addizione; un simile sommatore viene chiamato a **selezione del riporto**. Lo stesso trucco può essere ripetuto per costruire i sommatori a 16 bit come più sommatori a 8 bit e così via.

### Unità aritmetico logiche

La maggior parte dei calcolatori contiene un unico circuito in grado di effettuare le operazioni AND, OR e somma di due parole. Di solito un circuito di questo tipo funzionante su parole a  $n$  bit è costruito a partire da  $n$  identici circuiti per le singole posizioni dei bit. La Figura 3.19 è un esempio di tale circuito, chiamato **unità aritmetico logica o ALU (Arithmetic Logic Unit)**; esso può calcolare una qualsiasi delle quattro funzioni,  $A \text{ AND } B$ ,  $A \text{ OR } B$ ,  $\bar{B}$  oppure  $A + B$ , in base al valore binario (00, 01, 10 oppure 11) definito dalle linee  $F_0$  e  $F_1$  preposte alla selezione della funzione aritmetica. Si noti che in questo caso con  $A + B$  si intende la somma aritmetica di  $A$  e  $B$  e non l'operazione booleana OR.

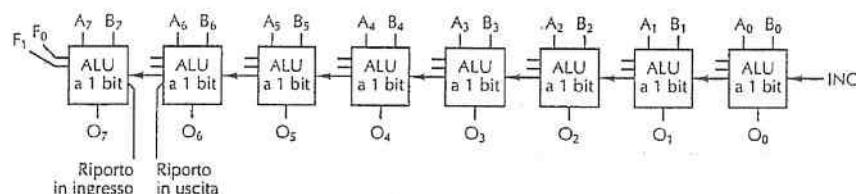


Figura 3.19 Otto ALU a 1 bit connesse per comporre una ALU a 8 bit. Per semplificare non sono mostrati segnali di abilitazione e segnali di inversione.

Il settore in basso a sinistra della nostra ALU contiene un decodificatore a 2 bit che permette di generare, in base ai segnali di controllo  $F_0$  ed  $F_1$ , i segnali di attivazione delle quattro operazioni. In base ai valori di  $F_0$  e  $F_1$  viene selezionata una sola delle quattro linee di attivazione, permettendo all'output della funzione selezionata di passare attraverso la porta logica OR che genera l'output finale.

Il settore in alto a sinistra contiene le porte logiche necessarie per calcolare  $A \text{ AND } B$ ,  $A \text{ OR } B$  e  $\bar{B}$ ; in base alle linee di attivazione che escono dal decodificatore uno solo di questi risultati viene passato alla porta logica finale OR. Dato che solo uno degli output del decodificatore varrà 1, soltanto una delle quattro porte AND che forniscono i valori d'ingresso alla porta OR verrà attivata; l'output delle altre sarà invece 0, indipendentemente dai valori di  $A$  e  $B$ . Oltre a poter usare  $A$  e  $B$  come input per le operazioni logiche o aritmetiche, è anche possibile forzare uno dei due valori a 0 negando rispettivamente ENA oppure ENB. È inoltre possibile ottenere il valore di  $\bar{A}$  abilitando il segnale INVA. Nel

Capitolo 4 vedremo alcuni utilizzi di INVA, ENA e ENB. In condizioni normali entrambi gli input sono abilitati; ENA e ENB valgono quindi 1, mentre INVA è impostato a 0. In questa situazione A e B sono immessi all'interno dell'unità logica senza alcuna modifica.

Il settore in basso a destra della ALU contiene un sommatore per calcolare la somma di  $A$  e  $B$ , e gestire allo stesso tempo i riporti; la gestione dei riporti è necessaria dato che con ogni probabilità vari circuiti dello stesso tipo potrebbero essere collegati fra loro per permettere operazioni su intere parole. Circuiti come quello mostrato nella Figura 3.19 sono attualmente disponibili e vengono chiamati **bit slices** ("suddivisioni di un bit"), che permettono ai progettisti di calcolatori di costruire ALU di larghezza arbitraria. La Figura 3.20 mostra una ALU a 8 bit costruita unendo otto ALU a 1 bit. Il segnale INC è utilizzato solo per le operazioni di addizione; quando è attivo permette di incrementare il risultato (cioè sommargli 1), rendendo così possibile il calcolo di somme come  $A + 1$  e  $A + B + 1$ .

Anni fa, un bit slice era un vero e proprio chip che si poteva comprare. Oggi un bit slice è una libreria che un progettista può replicare in un programma CAD per produrre un file per le macchine di produzione di chip. L'idea resta comunque essenzialmente la stessa.

### 3.2.4 Clock

In molti circuiti digitali l'ordine secondo il quale si verificano gli eventi è cruciale. In alcuni casi un evento deve precederne un altro, mentre in altre situazioni due eventi devono avvenire simultaneamente. Molti circuiti digitali utilizzano dei clock per gestire la sincronizzazione e permettere ai progettisti di ottenere le relazioni temporali desiderate. In questo contesto un **clock** è un circuito che emette una serie di impulsi di larghezza definita e a intervalli temporali costanti. L'intervallo temporale compreso tra le estremità di due impulsi consecutivi è detto **ciclo di clock**. La frequenza degli impulsi è generalmente compresa tra 100 MHz e 4 GHz, corrispondenti a cicli di clock compresi tra 10 ns e 250 ps. Per ottenere un'accuratezza elevata di solito la frequenza di clock è controllata da un oscillatore a cristalli.

In un calcolatore possono verificarsi più eventi durante uno stesso ciclo di clock. Se però è necessario che si verifichino in uno specifico ordine occorre dividere il ciclo di clock in sottocicli. Una tecnica che viene spesso utilizzata per ottenere una risoluzione più fine rispetto a quella del clock principale consiste nell'intercettare la linea del clock principale e inserirla in un circuito di cui si conosce il ritardo; in questo modo è possibile generare un secondo segnale di clock la cui fase è traslata rispetto a quella del clock principale, come mostra la Figura 3.20(a). Il diagramma di temporizzazione della Figura 3.20(b) fornisce quattro diversi riferimenti temporali utilizzabili per sincronizzare eventi discreti.

1. Fronte di salita di C1.
2. Fronte di discesa di C1.
3. Fronte di salita di C2.
4. Fronte di discesa di C2.

Associando diversi eventi ai quattro fronti è possibile stabilire per loro una desiderata sequenza. Se, all'interno di ogni ciclo di clock, sono necessari più di quattro riferimenti

ti temporali occorre collegare al clock principale altre linee secondarie e utilizzare circuiti con ritardi diversi.

In alcuni circuiti si è interessati maggiormente agli intervalli temporali piuttosto che agli istanti di tempo. A un evento potrebbe per esempio essere consentito di verificarsi in qualsiasi istante durante il quale  $C_1$  è alto e non per forza esattamente in corrispondenza del fronte di salita. Un altro evento potrebbe verificarsi solo quando  $C_2$  è alto. Se sono necessari più di due diversi intervalli temporali si possono utilizzare più linee di clock oppure si può fare in modo che gli intervalli in cui i segnali dei due clock sono alti si sovrappongano parzialmente. In quest'ultimo caso si possono distinguere quattro diversi intervalli:  $\overline{C_1}$  AND  $\overline{C_2}$ ,  $\overline{C_1}$  AND  $C_2$ ,  $C_1$  AND  $\overline{C_2}$  e  $C_1$  AND  $C_2$ .

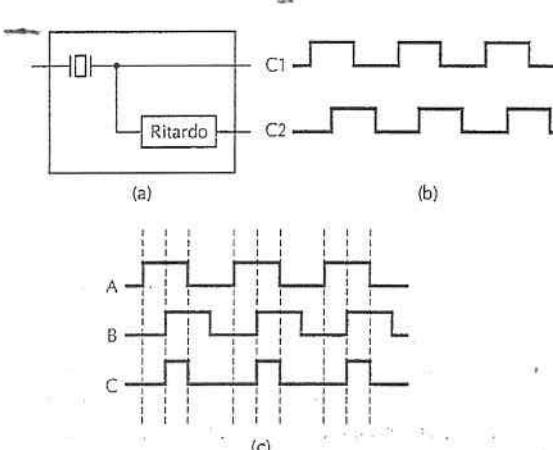


Figura 3.20 (a) Clock. (b) Diagramma di temporizzazione del clock. (c) Generazione di un clock asimmetrico.

Per essere più precisi i clock sono simmetrici nel senso che il tempo speso nello stato in cui il segnale è alto è uguale al tempo speso quando il segnale è basso, come mostra la Figura 3.20(b). Per generare una sequenza di impulsi asimmetrici il clock principale viene sfasato utilizzando un circuito ritardante e ne viene calcolato l'AND logico rispetto al segnale originale, come mostra il segnale C della Figura 3.20(c).

### 3.3 Memoria

Una componente essenziale di ogni calcolatore è la memoria; se non ci fosse non potrebbe esistere nessun calcolatore, almeno nella forma in cui lo conosciamo. La memoria è utilizzata per conservare sia le istruzioni da eseguire sia i dati. Nei paragrafi successivi esamineremo i componenti base di un sistema di memoria; partiremo dal livello delle porte logiche per comprenderne il funzionamento e il modo in cui sono combinate per formare memorie più capaci.

#### 3.3.1 Latch

Per creare una memoria a 1 bit è necessario disporre di un circuito che in qualche modo "ricordi" i precedenti valori di input. La Figura 3.21(a) mostra come sia possibile costruire un circuito di questo tipo utilizzando due porte NOR. Anche se è possibile costruire circuiti analoghi con porte NAND, nel seguito non li menzioneremo più, dato che dal punto di vista concettuale sono identici alle versioni basate su porte NOR.

Il circuito della Figura 3.21(a) è chiamato **latch SR** e ha due input:  $S$ , per impostare (*Setting*) il valore del latch e  $R$  per azzerarlo (*Resetting*). Il circuito ha anche due output,  $Q$  e  $\bar{Q}$ , che, come vedremo tra poco, sono complementari l'uno rispetto all'altro. Diversamente dalle reti combinatorie l'output di un latch non è determinato unicamente dai valori di input correnti.

Per vedere come ciò avviene assumiamo, come si verifica nella maggior parte dei casi, che sia  $S$  sia  $R$  valgano 0. Ai fini della spiegazione assumiamo inoltre che  $Q = 0$ . Dato che  $Q$  viene reinserito come input della porta NOR visualizzata in alto, entrambi gli input di questa porta valgono 0 generando come output il valore 1. Il valore 1 è riutilizzato come input della porta visualizzata in basso, i cui input sono 1 e 0 e producono come output  $Q = 0$ . Questo stato, mostrato nella Figura 3.21(a), è stabile.

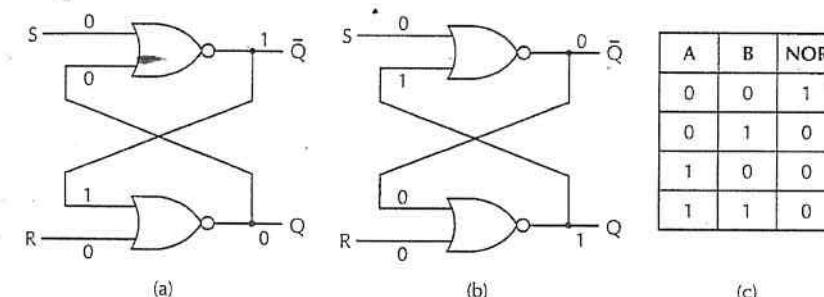


Figura 3.21 (a) Latch di tipo NOR nello stato 0. (b) Latch di tipo NOR nello stato 1. (c) Tabella di verità del NOR.

Immaginiamo ora che  $Q$  non sia 0, ma 1, mentre  $R$  e  $S$  continuano a valere 0. La porta logica visualizzata in alto ha come input 0 e 1; il suo output  $\bar{Q}$  vale 0 ed è riutilizzato come input della porta visualizzata in basso. Anche questo stato è stabile ed è mostrato nella Figura 3.21(b). Uno stato in cui entrambi gli output valgono 0 è invece instabile, in quanto obbliga entrambe le porte ad avere due 0 come input, il che, se fosse vero, produrrebbe come risultato 1 e non 0. Analogamente è impossibile che entrambi gli output valgano 1, dato che ciò forzerebbe gli input a 0 e 1, il cui risultato dovrebbe essere 0 e non 1. La conclusione è semplice: per  $R = S = 0$  il latch ha due stati stabili che identificheremo con 0 e 1 in base al valore di  $Q$ .

Esaminiamo ora quali effetti producono sullo stato del latch i valori dell'input. Supponiamo che  $S$  diventi 1, mentre  $Q = 0$ . Gli input del NOR superiore sono quindi 1 e 0 e

forzano l'output  $\bar{Q}$  a valere 0. Questo cambiamento porta entrambi gli input della porta inferiore a valere 0, producendo in output il valore 1. Settando quindi  $S$  (cioè impostandone il valore a 1) si può far passare lo stato da 0 a 1. Settando  $R$  a 1 quando il latch si trova nello stato 0 non si produce invece alcun effetto, dato che l'output della porta NOR in basso è 0 sia che i suoi input valgano 10 sia che valgano 11.

Applicando un ragionamento simile si può verificare facilmente che impostare  $S$  a 1 quando il latch è nello stato  $Q = 1$  non produce alcun effetto, mentre settando  $R$  si modifica lo stato del latch portandolo nello stato  $Q = 0$ . Riassumendo si può dire che quando  $S$  è impostato temporaneamente a 1 lo stato del latch diventa  $Q = 1$ , indipendentemente dallo stato in cui si trovava precedentemente. Allo stesso modo quando si imposta temporaneamente  $R$  a 1 si forza il latch a passare nello stato  $Q = 0$ . Il circuito "ricorda" quindi quale valore, se  $S$  oppure  $R$ , è stato settato per ultimo; utilizzando questa proprietà è possibile costruire le memorie dei calcolatori.

#### Latch SR temporizzato

Spesso è preferibile impedire che un latch cambi di stato se non in specifici momenti. Un circuito che gode di questa caratteristica è detto **latch SR temporizzato**; per costruirlo occorre modificare leggermente il circuito visto precedentemente. Il circuito della Figura 3.22 ha un input aggiuntivo, il *clock*, il cui valore è generalmente 0. Quando il *clock* vale 0 entrambe le porte AND generano in output il valore 0, indipendentemente dai valori di  $S$  e  $R$ , impedendo quindi al latch di cambiare di stato. Quando il *clock* vale 1 le porte AND non bloccano più i segnali  $S$  e  $R$  che possono dunque tornare a pilotare lo stato del latch. Nonostante il suo nome non è obbligatorio che il segnale collegato all'input aggiuntivo debba per forza essere comandato da un *clock*. I termini *enable* e *strobe* sono largamente utilizzati per indicare che l'input clock vale 1, ovvero che il circuito è dipendente dallo stato di  $S$  e da quello di  $R$ .

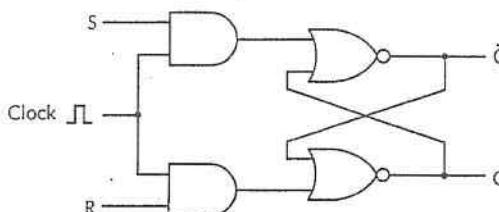


Figura 3.22 Latch SR temporizzato.

Finora abbiamo intenzionalmente evitato di spiegare che cosa succede se sia  $S$  sia  $R$  valgono 1: il circuito diventa non deterministico finché sia  $R$  sia  $S$  non tornino ad assumere il valore 0. L'unico stato consistente per  $S = R = 1$  è quando  $Q = \bar{Q} = 0$ ; non appena entrambi gli input ritornano al valore 0 il latch deve tuttavia passare istantaneamente in uno dei suoi due stati stabili. Se uno dei due input torna a 0 prima dell'altro, prevale quello che rimane al valore 1 più a lungo, dato che quando uno solo degli input vale 1

esso determina lo stato del latch. Se invece entrambi gli input ritornano a 0 nello stesso istante (cosa che può verificarsi molto raramente) il latch passa in uno dei due stati stabili in modo del tutto casuale.

#### Latch D temporizzato

Un buon modo per risolvere l'ambiguità del latch SR (causata dalla situazione  $S = R = 1$ ) è evitare che si verifichi. La Figura 3.23 mostra un circuito che ha un solo input,  $D$ . Dato che l'input della porta AND rappresentata in basso è sempre il complemento dell'input di quella superiore, non può mai accadere che entrambi gli input valgono 1. Quando  $D = 1$  e il *clock* vale 1, il latch viene portato nello stato  $Q = 1$ , mentre, quando  $D = 0$  e il *clock* vale 1, il latch passa nello stato  $Q = 0$ . In altre parole quando il *clock* vale 1 il valore corrente di  $D$  viene campionato e memorizzato nel latch. Questo circuito, chiamato **latch D temporizzato**, è una vera e propria memoria a 1 bit, in cui il valore memorizzato è sempre disponibile sulla linea  $Q$ . Per caricare in memoria il valore corrente di  $D$  occorre spedire un impulso positivo sulla linea del *clock*.

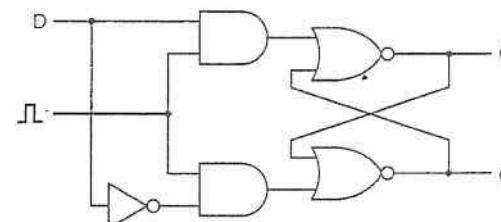


Figura 3.23 Latch D temporizzato.

Il circuito appena descritto richiede 11 transistori; esistono tuttavia circuiti più sofisticati (ma con un'implementazione meno ovvia) che possono memorizzare 1 bit utilizzando solo sei transistori. Nelle implementazioni reali si adottano queste soluzioni più sofisticate, che impiegano meno transistori. Il circuito descritto resta sempre stabile se vi è corrente (la corrente non è mostrata). Più avanti vedremo circuiti di memoria che dimenticano rapidamente il loro stato se non sono in qualche modo "richiamati".

#### 3.3.2 Flip-flop

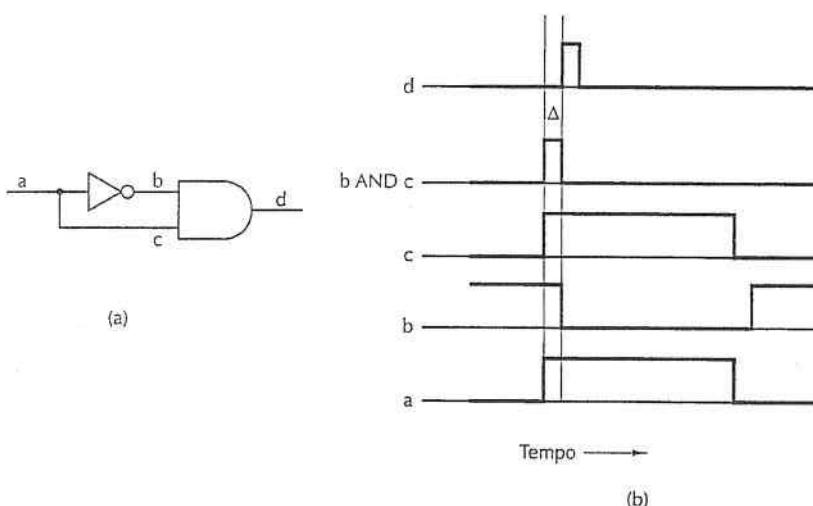
In molti circuiti è necessario campionare il valore di una certa linea in un particolare istante e memorizzarlo. In questi circuiti, chiamati **flip-flop**, la transizione di stato non si verifica quando il *clock* vale 1, ma durante la transizione del *clock* da 0 a 1 (fronte di salita) oppure da 1 a 0 (fronte di discesa). In questa situazione la lunghezza dell'impulso del *clock* non ha alcuna importanza, purché le transizioni si verifichino con sufficiente velocità.

Per sottolinearlo ulteriormente ripetiamo la differenza che c'è tra un flip-flop e un latch: un flip-flop è una commutazione sul fronte, mentre un latch è una commutazione a

livello. Occorre però prestare attenzione al fatto che in letteratura spesso questi termini vengono confusi; molti autori utilizzano il termine "flip-flop" per indicare un latch e viceversa.

Esistono vari approcci per progettare un flip-flop. Se per esempio esistesse un metodo per generare un impulso di lunghezza estremamente breve sul fronte di salita, si potrebbe immettere tale impulso in un latch D. Il circuito che implementa questa soluzione è mostrato nella Figura 3.24(a).

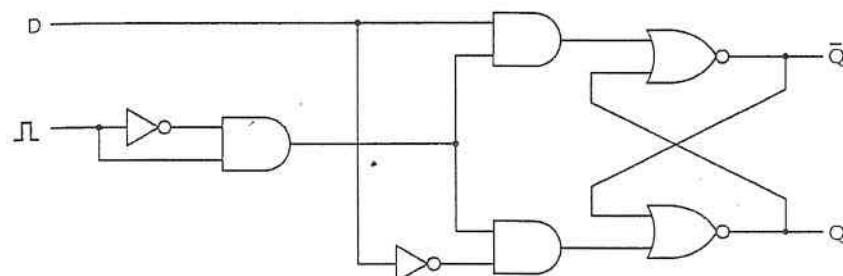
A prima vista potrebbe sembrare che l'output della porta AND debba essere sempre zero, dato che l'AND tra un qualsiasi segnale e il suo inverso vale sempre zero; in realtà la situazione, anche se in modo sottile, è diversa. L'invertitore induce un piccolo, ma non nullo, ritardo di propagazione che permette al circuito di funzionare in modo corretto. Supponiamo di misurare la tensione in quattro punti diversi:  $a$ ,  $b$ ,  $c$  e  $d$ . Il segnale di input, misurato in  $a$ , è un lungo impulso di clock, come mostra, in basso, la Figura 3.24(b); il segnale in  $b$  è invece rappresentato dal grafico che nella figura si trova sopra quello precedente. Si noti che questo segnale, oltre a essere stato invertito, risulta anche sfasato di un ritardo che è di pochi nanosecondi (a seconda del tipo di invertitore utilizzato).



**Figura 3.24** (a) Generatore d'impulsi. (b) Diagrammi temporali

Anche il segnale  $c$  subisce un ritardo, che però è determinato soltanto dal tempo di propagazione del segnale (a due terzi della velocità della luce). Se la distanza tra  $a$  e  $c$  è, per esempio, di 20  $\mu\text{m}$ , il ritardo di propagazione è di un decimo di picosecondo. Tale valore è certamente trascurabile rispetto al tempo di commutazione dell'invertitore. Di conseguenza per qualsiasi intento o scopo del circuito, il segnale  $c$  può essere considerato identico al segnale  $a$ .

Quando gli input  $b$  e  $c$  vengono processati dalla porta AND si ottiene come risultato un impulso di breve durata, come mostra la Figura 3.25(b); la larghezza  $\Delta$  dell'impulso è uguale al ritardo dell'invertitore, generalmente inferiore a 5 ns. Com'è illustrato nella Figura 3.24(b) l'output della porta AND corrisponde semplicemente a questo impulso sfasato del ritardo interno alla porta logica AND. Questo sfasamento temporale significa che il latch D verrà attivato con un ritardo fisso rispetto al fronte di salita del clock; tuttavia ciò non ha alcun effetto sulla durata dell'impulso. Un impulso di 1 ns che segnala quando campionare la linea D può essere considerato sufficientemente corto per una memoria con un ciclo della durata di 10 ns; in un caso simile il circuito completo potrebbe essere quello della Figura 3.25. Occorre sottolineare che il vantaggio dell'architettura di questo flip-flop è di essere di facile comprensione; tuttavia nella realtà si ricorre di solito ad architetture più sofisticate.



**Figura 3.25** Flip-flop D

La Figura 3.26 mostra i simboli comunemente usati per i latch e i flip-flop. La Figura 3.26(a) indica un latch il cui stato viene caricato quando il clock,  $CK$ , vale 1; al contrario lo stato del latch della Figura 3.26(b) è generalmente 1, ma passa temporaneamente al valore 0 per assumere lo stato dalla linea  $D$ . Le Figure 3.26(c) e (d) rappresentano due flip-flop invece che due latch; ciò è indicato dal simbolo triangolare vicino all'input del clock. La Figura 3.26(c) cambia stato in corrispondenza del fronte di salita dell'impulso del clock (transizione da 0 a 1), mentre la Figura 3.26(d) cambia stato in corrispondenza del fronte di discesa (transizione da 1 a 0). Molti latch e flip-flop, seppur non tutti, hanno anche l'output  $\bar{Q}$  e alcuni sono dotati di due input aggiuntivi, *Set* o *Preset* (che forzano lo stato a  $Q = 1$ ) e *Reset* o *Clear* (che forzano lo stato a  $Q = 0$ ).

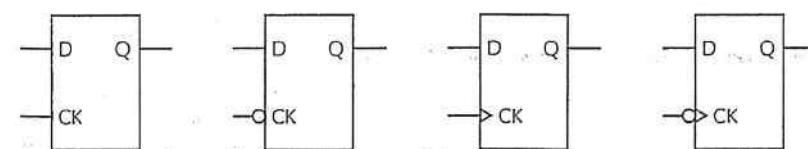


Figura 3.26 Flip-flop D temporizzato

### 3.3.3 Registri

I Flip-Flop possono essere combinati per creare dei registri che memorizzano dati composti da più bit. La figura 3.27 mostra come otto flip-flop possano essere raggruppati per formare un registro da 8 bit. Il registro riceve in ingresso un valore di 8 bit (da  $I_0$  a  $I_7$ ) quando vi è una transizione del clock CK. Per implementare un registro tutte le linee di clock sono collegate allo stesso segnale in ingresso CK, in modo che quando si ha una transizione di clock il registro riceve dal canale di ingresso il nuovo dato di 8 bit.

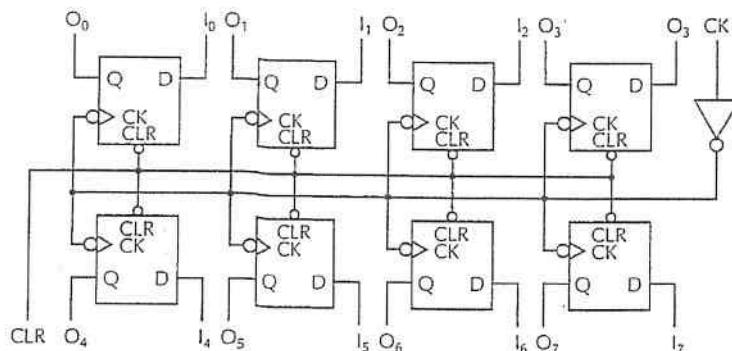


Figura 3.27 Un registro a 8 bit costruito a partire da flip-flop a 1 bit.

I singoli flip-flop sono del tipo mostrato nella Figura 3.25(d), anche se il cerchietto d'inversione non è presente per via dell'invertitore collegato al segnale di clock CK; per questa differenza i flip-flop vengono caricati in corrispondenza della transizione di salita. Anche tutti e otto i canali di cancellazione sono collegati fra loro, di modo che quando CLR va a 0 tutti i flip-flop siano forzati ad assumere lo stato 0. Ci si potrebbe chiedere per quale motivo il clock CK è invertito nella linea di input per poi essere invertito nuovamente in corrispondenza di ciascun flip-flop. Il motivo è che un segnale potrebbe non avere sufficiente corrente per pilotare tutti i flip-flop; l'invertitore svolge in realtà il ruolo di amplificatore.

Una volta progettato un registro a 8 bit, lo si può utilizzare per costruire registri più grandi. Per esempio, un registro a 32 bit può essere costruito combinando due registri a 16 bit, unendo i loro segnali di clock CK e di cancellazione CLR. Nel corso del Capitolo 4 analizzeremo più da vicino i registri e il loro impiego.

### 3.3.4 Organizzazione della memoria

Nei paragrafi precedenti siamo partiti dalla semplice memoria a 1 bit della Figura 3.23 per arrivare alla memoria a 8 bit della Figura 3.27. Per realizzare memorie di dimensione maggiore è però necessaria un'organizzazione di tipo diverso, nella quale sia possibile indirizzare singole parole. La Figura 3.28 mostra un'organizzazione della memoria molto

comune che soddisfa questo requisito. L'esempio illustra una memoria con quattro parole a 3 bit in cui ciascuna operazione legge o scrive un'intera parola. Anche se la memoria ha una capacità totale (12 bit) decisamente maggiore rispetto a quella del flip-flop ottale visto precedentemente, essa richiede un numero inferiore di pin. Una caratteristica ancora più rilevante è che questa organizzazione è facilmente estendibile a memorie di dimensione maggiore. Si noti che il numero di parole è sempre una potenza di 2.

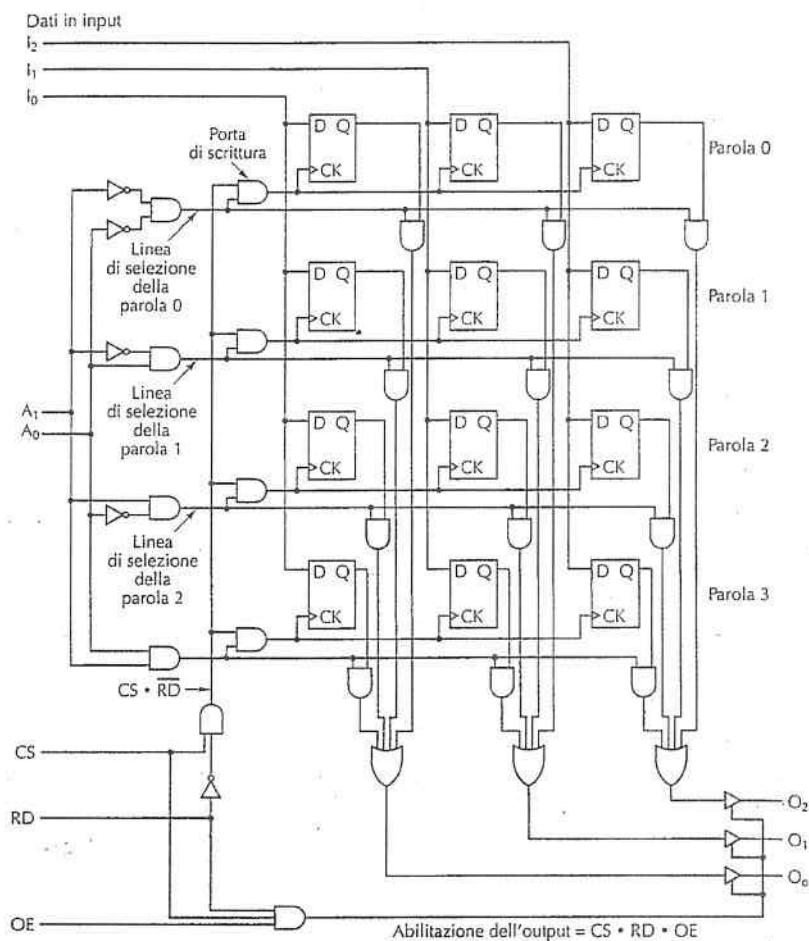


Figura 3.28 Diagramma logico di una memoria  $4 \times 3$ . Ogni riga è una delle quattro parole a 3 bit. Lettura e scrittura riguardano sempre parole complete.

A prima vista la memoria della Figura 3.28 potrebbe sembrare complessa, ma in realtà, se si considera la regolarità della struttura, si nota che è piuttosto semplice. Delle otto linee di input tre sono per i dati,  $I_0$ ,  $I_1$  e  $I_2$ , due per l'indirizzo,  $A_0$  e  $A_1$ , e tre per i controlli, CS per la selezione del chip, RD per distinguere tra lettura e scrittura e OE per l'abilitazione dell'output. Le tre linee di output,  $O_0$ ,  $O_1$  e  $O_2$ , sono invece dedicate ai dati. È interessante osservare che questa memoria da 12 bit necessita di meno segnali rispetto ai registri a 8 bit visti in precedenza. I registri a 8 bit utilizzano 20 pin, includendo corrente e terra, mentre le memorie da 12 bit ne richiedono 13, perché, a differenza dei registri, condividono un segnale di output. Nel nostro esempio, 4 bit di memoria condividono lo stesso segnale di output. Il valore delle linee di indirizzo determina a quale dei 4 bit di memoria è permesso di ricevere o restituire un valore.

Per utilizzare questo chip è necessario che una componente logica esterna imponga, in caso di lettura, sia CS sia RD al valore alto (valore logico 1), mentre, in caso di scrittura, li imposta al valore basso (valore logico 0). Le due linee dell'indirizzo devono essere impostate in modo da indicare quale delle quattro parole a 3 bit deve essere letta o scritta. In lettura le linee di input non vengono utilizzate e la parola selezionata viene resa disponibile sulle linee di output. In scrittura i bit presenti sulle linee di input dei dati vengono caricati nella parola di memoria selezionata, mentre le linee di output non vengono utilizzate.

Per comprendere il funzionamento della memoria osserviamo ora in modo più dettagliato la Figura 3.28. Le quattro porte AND situate a sinistra della memoria formano un decodificatore e servono a selezionare la parola. I quattro invertitori in input sono stati collocati per far sì che ciascuna porta logica sia abilitata (l'output è alto) da un indirizzo diverso. Ciascuna porta è collegata a una linea per la selezione di una delle parole, indicate dall'alto verso il basso con i numeri 0, 1, 2 e 3. Quando il chip viene selezionato per un'operazione di scrittura la linea verticale etichettata con  $CS \cdot RD$  assume valore alto, abilitando quindi una delle quattro porte di scrittura; la porta abilitata dipende da quale linea per la selezione della parola ha assunto valore alto. L'output della porta di scrittura guida tutti i segnali CK relativi alla parola selezionata, caricando i dati di input nei flip-flop che costituiscono la parola stessa. La scrittura è possibile soltanto quando CS è alto e RD è basso; l'unica parola a essere scritta è quella selezionata dai segnali  $A_0$  e  $A_1$ , mentre le altre non vengono modificate.

L'operazione di lettura è simile: la decodifica dell'indirizzo, per esempio, si svolge esattamente nello stesso modo, la linea  $CS \cdot RD$  assume però valore basso, disabilitando di conseguenza tutte le porte di scrittura e impedendo la modifica dei flip-flop. La linea per la selezione della parola scelta abilita le porte AND cui sono collegati i bit Q della parola selezionata. La parola selezionata spedisce quindi i propri dati alle porte OR mostrate nella parte bassa della figura. Dato che le altre parole generano in output valori 0, il risultato delle porte OR è identico al valore memorizzato nella parola selezionata. Le tre parole non selezionate non forniscono quindi alcun contributo all'output finale.

Anche se si potrebbe progettare un circuito nel quale le tre porte OR siano collegate direttamente alle tre linee di output dei dati, in alcuni casi ciò potrebbe causare dei problemi. Nell'illustrazione abbiamo distinto le linee di input da quelle di output, ma nelle memorie reali si utilizzano le stesse linee. Se avessimo collegato le porte OR alle linee

di output il chip avrebbe cercato di spedire in output i dati anche durante un'operazione di scrittura, forzando ciascuna linea ad assumere un particolare valore e interferendo quindi con i dati di input. Per questo motivo è preferibile avere la possibilità di connettere le porte OR alle linee di output durante le letture e di disconnetterle completamente durante le scritture. Quello di cui abbiamo bisogno è un interruttore elettronico che possa instaurare o interrompere una connessione in una frazione di nanosecondo.

Un interruttore del genere è chiamato **buffer non invertente** e la Figura 3.29(a) mostra il simbolo che lo rappresenta. Esso ha un dato di input, un dato di output e un input di controllo. Quando l'input di controllo è alto il buffer funge da collegamento, come mostra la Figura 3.29(b). Quando invece l'input di controllo è basso il buffer si comporta come un circuito aperto, come mostra la Figura 3.29(c): è come se qualcuno avesse scollegato il dato di output dal resto del circuito con una pinza tagliafili. Tuttavia, diversamente dall'utilizzo di una pinza tagliafili, la connessione può essere ripristinata in una frazione di nanosecondo semplicemente reimpostando il segnale di controllo al valore alto.

La Figura 3.29(d) mostra un **buffer invertente** che si comporta come un normale invertitore quando il controllo ha valore alto e disconnette l'output dal circuito quando il controllo ha valore basso. I due buffer sono **dispositivi a-tre stati**, in quanto possono generare in output i valori 0 e 1, oppure nessuno dei due (circuito aperto). I buffer inoltre amplificano il segnale e possono quindi guidare più input allo stesso tempo; talvolta vengono impiegati all'interno dei circuiti proprio per questa ragione, anche quando non sono richieste le loro proprietà d'inversione.

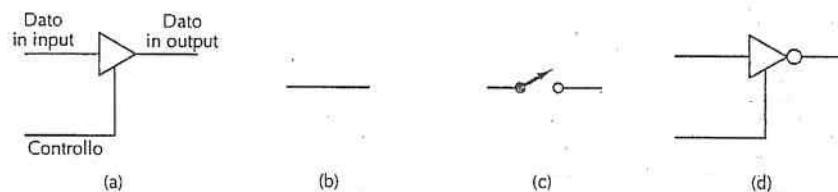


Figura 3.29 (a) Buffer non invertente. (b) Risultato di (a) quando il controllo è alto. (c) Risultato di (a) quando il controllo è basso. (d) Buffer invertente.

Tornando al circuito della memoria dovrebbe essere chiaro quale sia il ruolo dei buffer non invertenti che si trovano sulle linee di output dei dati. Quando CS, RD e OE hanno tutti il valore alto anche il segnale per l'abilitazione dell'output è alto; ciò permette di attivare i buffer e di spedire una parola sulle linee di output. Al contrario, quando uno qualsiasi dei segnali CS, RD e OE è basso, l'output è scollegato dal resto del circuito.

### 3.3.5 Chip di memoria

Una proprietà interessante della memoria della Figura 3.28 è che può essere facilmente ampliata. Quella che abbiamo descritto è una memoria  $4 \times 3$ , costituita cioè da quattro

parole di 3 bit ciascuna. Per estenderla alla dimensione  $4 \times 8$  occorre semplicemente aggiungere cinque colonne composte da quattro flip-flop ciascuna, così come cinque linee di input aggiuntive e altrettante linee di output. Per passare dalla dimensione  $4 \times 3$  a quella  $8 \times 3$  si devono aggiungere quattro nuove righe di tre flip-flop ciascuna, oltre a una linea aggiuntiva,  $A_2$ , per l'indirizzo. Una memoria con questa struttura dovrebbe avere un numero di parole esprimibile come una potenza di 2 al fine di massimizzare l'efficienza; il numero di bit di una parola può invece assumere qualsiasi valore.

Dato che la tecnologia dei circuiti integrati è particolarmente adatta a realizzare chip la cui struttura interna abbia uno schema bidimensionale ripetuto, i chip di memoria ne sono un'applicazione ideale. Con l'avanzamento della tecnologia continua ad aumentare il numero di bit che si può inserire in un chip. Tale valore raddoppia all'incirca ogni 18 mesi (legge di Moore). I chip di dimensioni maggiori non sempre rendono obsoleti quelli più piccoli, dato che spesso occorre trovare dei compromessi tra diversi fattori, quali la capacità, la velocità, l'alimentazione, il prezzo e la comodità di interfacciamento. Di solito i chip di dimensioni maggiori sono venduti come prodotti di qualità superiore e quindi a un prezzo per bit più elevato rispetto a quelli più vecchi e più piccoli.

Fissata una dimensione della memoria esistono diversi modi per organizzare il chip. La Figura 3.30 mostra due possibili organizzazioni per un vecchio chip di memoria della dimensione di 4 Mbit:  $512K \times 8$  e  $4096K \times 1$ . Per inciso, le dimensioni dei chip di memoria sono generalmente indicate in bit, e non in byte, e anche noi ci atterremo a questa convenzione. Nella Figura 3.30(a) sono necessarie 19 linee per indirizzare uno dei  $2^{19}$  byte e otto linee di output per caricare e memorizzare i byte selezionati.

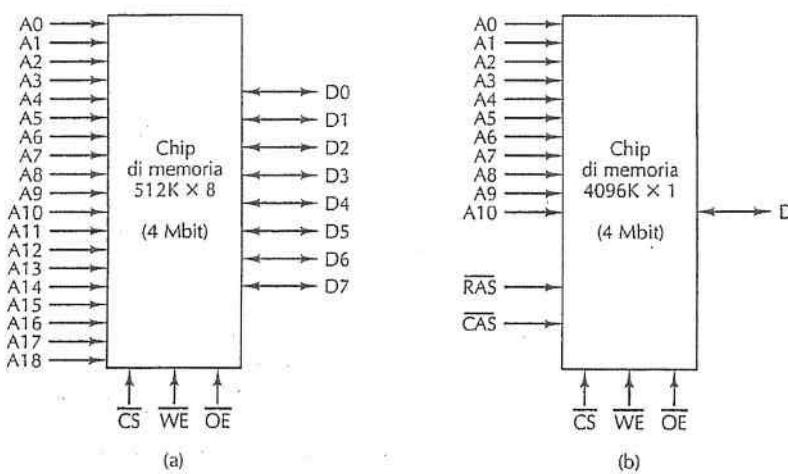


Figura 3.30 Due modi di organizzare un chip di memoria a 4 Mbit.

Occorre sottolineare un aspetto della terminologia utilizzata. Su alcuni pin l'applicazione di un'alta tensione genera una qualche azione, mentre su altri le azioni sono innescate da bassi valori di tensione. Per evitare confusione diremo che un segnale è asserito (piuttosto che dire che assume valore alto o basso) per indicare che è impostato in modo da generare una qualche azione. Alcuni pin sono asseriti con valore alto, mentre altri con valore basso. I pin asseriti con valore basso sono identificati da una linea sopra il loro nome. Un segnale chiamato  $CS$  (*Chip Select*) viene quindi asserito con il valore alto, mentre uno chiamato  $\overline{CS}$  è asserito con il valore basso. L'opposto di asserito è negato; quando non succede nulla di particolare i pin sono negati.

Torniamo ora al nostro chip di memoria. Dato che un calcolatore ha di solito vari chip di memoria è necessario disporre di un segnale che selezioni il chip richiesto in un dato momento, di modo che solo esso risponda al segnale, mentre tutti gli altri lo ignorino. Per questo scopo si utilizza il segnale  $\overline{CS}$ , asserito quando si intende abilitare il chip. Inoltre è necessario un metodo per distinguere tra le operazioni di lettura e quelle di scrittura; ciò viene fatto dal segnale  $WE$  (*Write Enable*), utilizzato per indicare che i dati devono essere scritti piuttosto che letti. Infine il segnale  $OE$  (*Output Enable*) viene asserito per guidare i segnali di output; quando non è asserito, l'output del chip è sconnesso dai circuiti.

Nella Figura 3.30(b) si utilizza un diverso schema per l'indirizzamento. Al suo interno questo chip è organizzato come una matrice di  $2048 \times 2048$  celle a 1 bit, che forniscono una capacità totale di 4 Mbit. Per indirizzare il chip si seleziona inizialmente una riga immettendo un numero a 11 bit sui pin dell'indirizzo e asserendo il segnale  $RAS$  (*Row Address Strobe*, "strobe dell'indirizzo di riga"). Successivamente si immette sui pin dell'indirizzo un numero di colonna e si asserisce il segnale  $CAS$  (*Column Address Strobe*, "strobe dell'indirizzo di colonna"). Il chip risponde accettando, oppure generando in output, un dato da 1 bit.

I chip di memoria di grandi dimensioni sono spesso costruiti come matrici di  $n \times n$  indirizzate da numeri di riga e colonna. Questo tipo di architettura riduce il numero di pin necessari, ma rende allo stesso tempo più lento il chip, in quanto sono necessari due cicli di indirizzamento, uno per la riga e uno per la colonna. Per riguadagnare parte della velocità persa a causa dell'architettura, in alcuni chip è possibile specificare un indirizzo di riga seguito da una sequenza di indirizzi di colonna in modo da poter accedere a bit consecutivi all'interno di una stessa riga.

Anni fa i chip di memoria più grandi erano spesso organizzati come nella Figura 3.30(b). Quando però la dimensione delle parole di memoria è passata da 8 bit a 32 bit, i chip larghi 1 bit sono diventati poco convenienti. Per costruire una memoria con parole a 32 bit mediante chip  $4096K \times 1$  è infatti necessario utilizzarne 32 in parallelo. Questi 32 chip forniscono una capacità totale di 16 MB, mentre l'utilizzo di chip  $512K \times 8$  richiede solo quattro chip in parallelo e fornisce memorie della dimensione di 2 MB. Per evitare di avere 32 chip per la memoria, quasi tutti i produttori offrono attualmente famiglie di chip con larghezze da 4, 8 e 16 bit. Ovviamente se si considerano parole a 64 bit la situazione è ancora più critica.

La Figura 3.31 mostra due esempi di moderni chip a 512 Mbit. I chip hanno quattro banchi di memoria di 128 Mbit ciascuno e richiedono quindi due linee per la selezione

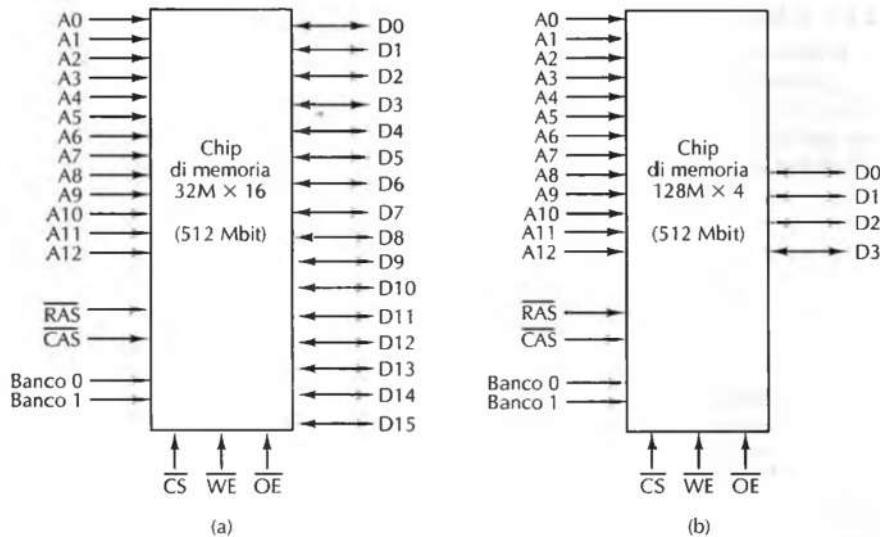


Figura 3.32 Due modi di organizzare un chip di memoria a 512 Mbit.

Questi 32 chip forniscono una capacità totale di 16 MB, mentre l'utilizzo di chip  $512K \times 8$  richiede solo quattro chip in parallelo e fornisce memorie della dimensione di 2 MB. Per evitare di avere 32 chip per la memoria, quasi tutti i produttori offrono attualmente famiglie di chip con larghezze da 4, 8 e 16 bit. Ovviamente se si considerano parole a 64 bit la situazione è ancora più critica.

La Figura 3.32 mostra due esempi di moderni chip di memoria a 512 Mbit. I chip hanno quattro banchi di memoria di 128 Mbit ciascuno e richiedono quindi due linee per la selezione del banco desiderato. La Figura 3.32(a) mostra un progetto  $32M \times 16$ , con 13 linee per il segnale  $\overline{RAS}$ , 10 linee per il segnale  $\overline{CAS}$  e 2 linee per la selezione del banco. L'insieme di questi 25 segnali permette di indirizzare ciascuna delle  $2^{25}$  celle interne, tutte a 16 bit. La Figura 3.32(b) mostra invece un progetto  $128M \times 4$ , con 13 linee per il segnale  $\overline{RAS}$ , 12 linee per il segnale  $\overline{CAS}$  e 2 linee per la selezione del banco. In questo caso i 27 segnali permettono di selezionare una qualsiasi delle  $2^{27}$  celle interne a 4 bit. La scelta del numero di righe e colonne di un chip dipende da scelte ingegneristiche; in ogni caso non è necessario che la matrice sia quadrata.

Questi esempi mettono in evidenza due distinti problemi, non collegati tra loro, che riguardano la progettazione dei chip di memoria. Il primo concerne la larghezza dell'output (in bit): il chip invia 1, 4, 8, 16 o un altro numero di bit in una volta sola? Il secondo corrisponde a chiedersi se tutti gli indirizzi vengono forniti allo stesso tempo su pin distinti oppure si forniscono prima i valori di riga e poi quelli di colonna come negli esempi della Figura 3.32. Un ingegnere che deve costruire un chip di memoria, prima di cominciare a progettare il proprio chip, deve dare una risposta a queste domande.

### 3.3.6 RAM e ROM

Tutte le memorie studiate finora possono essere sia lette sia scritte. Le memorie di questo tipo sono chiamate **RAM** (*Random Access Memory*, "memoria ad accesso casuale"); il termine è fuorviante in quanto tutti i chip di memoria sono accessibili in modo casuale, ma ormai è talmente radicato che è impossibile modificarlo. Esistono due tipi di RAM: statiche e dinamiche. Le **RAM statiche** (SRAM) sono costruite utilizzando circuiti simili ai flip-flop D e hanno la proprietà di mantenere il proprio contenuto fintanto che vi è alimentazione: per secondi, minuti, ore o anche giorni. Le RAM statiche sono molto veloci e i loro tempi di accesso sono usualmente dell'ordine dei nanosecondi. Per questa ragione sono molto diffuse come memorie cache di secondo livello.

Al contrario le **RAM dinamiche** (DRAM) non usano flip-flop, ma sono composte da un array di celle, ciascuna delle quali contiene un transistor e un piccolo condensatore. Il condensatore può essere caricato o scaricato per memorizzare i valori 0 oppure 1. Dato che la carica elettrica tende a disperdersi occorre effettuare, a intervalli temporali di pochi millisecondi, un *refresh* (ricarica) di ciascun bit della RAM dinamica per evitare che i dati vadano persi. Le RAM dinamiche richiedono un'interfaccia più complessa rispetto a quelle statiche per via dei componenti logici esterni che devono occuparsi del refresh; in molte applicazioni questo svantaggio è compensato dalla maggior capacità che le RAM dinamiche possono offrire.

Le RAM dinamiche hanno un'elevata densità (molti bit per chip) dato che richiedono soltanto un transistor e un condensatore per bit (rispetto ai sei transistor per bit della migliore RAM statica). Per questa ragione le memorie centrali sono quasi sempre costruite utilizzando RAM dinamiche. Questa grande capacità ha però un prezzo: le RAM dinamiche sono lente (decine di nanosecondi). Per cercare di trarre vantaggio dalle proprietà di entrambi i tipi di memoria spesso si ricorre a un uso combinato, realizzando una cache con RAM statica e la memoria centrale con RAM dinamica.

Esistono vari tipi di chip di RAM dinamica. Il più datato è il DRAM FPM (*Fast Page Mode*), ancora utilizzato nei calcolatori più vecchi. Internamente è organizzato a matrice di bit, e per funzionare occorre che l'hardware fornisca prima l'indirizzo di riga e poi quello di colonna, allo stesso modo di quanto avveniva con i segnali  $\overline{RAS}$  e  $\overline{CAS}$  nel contesto della Figura 3.31. Inoltre si utilizzano dei segnali esplicativi per indicare alla memoria quando rispondere; dunque il funzionamento della memoria è asincrono rispetto al clock principale del sistema.

La DRAM FPM è stata sostituita dalla DRAM EDO (*Extended Data Output*) in cui un riferimento alla memoria può avere inizio ancor prima che sia completato il precedente. Questa semplice strategia a pipeline non accelera il singolo riferimento, ma aumenta la larghezza di banda della memoria.

I chip FPM e EDO lavoravano in modo accettabile quando i chip di memoria avevano dei cicli di 12 ns, o erano addirittura più lenti. Quando i processori sono diventati così veloci da richiedere memorie più rapide, i chip FPM e EDO sono stati sostituiti dalle **SDRAM** (*Synchronous DRAM*, DRAM sincrona); la SDRAM è una RAM ibrida, in parte statica e in parte dinamica, ed è guidata dal clock principale del sistema. Il maggior vantaggio di queste RAM è che il clock elimina la necessità dei segnali di controllo per specificare al chip quando deve rispondere. Al contrario la CPU comunica alla memoria per quanti cicli deve funzionare e poi fa partire l'esecuzione. Ad ogni ciclo la memoria manda in output 4, 8 o 16 bit, a seconda del numero delle sue linee di output. L'eliminazione dei segnali di controllo aumenta il tasso di trasferimento dati tra CPU e memoria.

Dopo la SDRAM il passaggio successivo è stata la SDRAM DDR (*Double Data Rate*, "memoria a duplice tasso di trasferimento"). In queste memorie il chip produce un output sul fronte di salita del segnale di clock e uno sul fronte di discesa, raddoppiando così il tasso di trasferimento dati. Un chip DDR largo 8 bit e funzionante a 200 MHz genera in output, per 200 milioni di volte al secondo (ovviamente per un breve intervallo di tempo), due valori a 8 bit; questi dati corrispondono a un *burst rate* (cioè una frequenza di picco) teorico di 3,2 Gbps. Le memorie DDR2 e DDR3 offrono prestazioni migliori delle DDR grazie all'incremento delle velocità del bus a 533 MHz e 1067 MHz rispettivamente. Quando questo libro è andato in stampa la prima volta, il chip DDR3 più veloce poteva trasferire dati alla velocità di 17.067 GB/s.

#### Chip di memoria non volatile

Le RAM non sono l'unico tipo di chip di memoria. In molte applicazioni, come i giocattoli, gli elettrodomestici e le automobili, il programma e alcuni dati devono rimanere memorizzati anche quando viene tolta l'alimentazione. Inoltre non si richiede mai la modifica del programma né dei dati installati. Questi requisiti hanno portato allo sviluppo delle **ROM** (*Read-Only Memory*, "memoria di sola lettura") che non possono essere modificate o cancellate, né intenzionalmente né accidentalmente. I dati sono inseriti durante la sua fabbricazione. Ciò avviene esponendo alla luce un materiale fotosensibile attraverso una maschera contenente il pattern di bit desiderato e incidendo quindi la superficie esposta (o non esposta). L'unico modo per cambiare il programma consiste nella sostituzione dell'intero chip.

In grandi volumi le ROM sono molto più economiche delle RAM dato che il costo per la realizzazione della maschera viene ammortizzato dal gran numero di esemplari. Le memorie ROM non sono però flessibili, dato che non possono essere modificate dopo la fabbricazione; inoltre il tempo che passa tra l'ordine e la ricezione delle ROM può essere di settimane. Per far sì che le aziende potessero sviluppare più agevolmente nuovi prodotti basati su ROM sono state inventate le **PROM** (*Programmable ROM*, ROM programmabile). Una PROM differisce da una ROM per il fatto che può essere programmata (una volta) sul posto, evitando quindi i tempi per il completamento dell'ordine da parte del produttore. Molte PROM contengono un array di piccoli fusibili, che possono essere bruciati: ciò viene fatto selezionando righe e colonne e applicando un'alta tensione a un particolare pin del chip.

Il passo successivo in questa linea di prodotti è stata la **EPROM** (*Erasable PROM*, PROM cancellabile), i cui campi non solo possono essere programmati, ma anche cancellati. Quando si espone la piccola lente al quarzo che si trova nella EPROM a un'intensa luce ultravioletta per 15 minuti, tutti i bit assumono il valore 1. Se ci si aspetta che la produzione di un chip richiederà un gran numero di fasi di test le EPROM risultano molto più economiche delle PROM, dato che possono essere riutilizzate. In genere le EPROM hanno la stessa organizzazione delle RAM statiche. La EPROM 27C040 a 4 Mbit utilizza per esempio la stessa organizzazione della Figura 3.31(a), generalmente impiegata per una RAM statica. È interessante notare che chip vecchi come questo non sono completamente morti, ma sono diventati più economici e hanno trovato un nuovo

utilizzo in prodotti di bassa fascia particolarmente sensibili ai costi. Un chip 27C040 oggi costa meno di 3\$ e si può spendere ancora meno se lo si compra in grandi quantità.

Meglio ancora delle EPROM sono la **EEPROM** (la prima E sta per *elettricamente*) che possono essere cancellate applicando impulsi elettrici invece di dover inserire il chip in una camera speciale per l'esposizione alla luce ultravioletta. Inoltre queste memorie sono riprogrammabili senza doverle rimuovere dal circuito, mentre una EPROM deve essere inserita nello speciale dispositivo che ne permette la programmazione. Le EEPROM però non possono essere più grandi di 1/64 delle comuni EPROM e la loro velocità è solo la metà. Le EEPROM non possono competere con le DRAM e le SRAM dato che sono 10 volte più lente, hanno una capacità 100 volte minore e sono molto più costose; esse sono utilizzate solo quando la loro proprietà di non volatilità è cruciale.

Un tipo più recente di EEPROM è la **memoria flash**. Diversamente dalla EPROM, che è cancellabile per esposizione alla luce ultravioletta, e dalla EEPROM, nella quale ogni singolo byte è cancellabile, la memoria flash è cancellabile a blocchi e riscrivibile. Come la EEPROM anche la memoria flash può essere cancellata senza doverla rimuovere dal circuito. Varie aziende producono piccole schede con circuiti stampati che contengono anche 64 GB di memoria flash utilizzate, oltre che per altri scopi, come "pellicole" per memorizzare le foto delle macchine fotografiche digitali. Come detto nel Capitolo 2, la memoria flash sta iniziando a sostituire i dischi magnetici. Utilizzata come disco, la memoria flash offre minori tempi d'accesso e minori consumi, ma con un costo per bit molto più alto. La Figura 3.32 mostra un riepilogo dei vari tipi di memorie.

Tipo	Categoria	Cancellazione	Byte modificabili	Volatile	Tipico utilizzo
SRAM	Read/write	Elettrica	Sì	Sì	Cache di secondo livello
DRAM	Read/write	Elettrica	Sì	Sì	Memoria centrale (vecchia)
SDRAM	Read/write	Elettrica	Sì	Sì	Memoria centrale (recente)
ROM	Read-only	Impossibile	No	No	Elettrodomestici (prodotti in grandi volumi)
PROM	Read-only	Impossibile	No	No	Dispositivi (prodotti in piccoli volumi)
EPROM	Read-mostly	Raggi UV	No	No	Prototipazione di dispositivi
EEPROM	Read-mostly	Elettrica	Sì	No	Prototipazione di dispositivi
Flash	Read/write	Elettrica	No	No	"Pellicola" per macchine fotografiche digitali

Figura 3.32 Confronto tra vari tipi di memoria.

## FPGA

Come abbiamo visto nel Capitolo 1, gli FPGA (*field-programmable gate array*) sono circuiti integrati che contengono una logica programmabile e permettono di formare un circuito logico arbitrario semplicemente caricando l’FPGA con i dati di configurazione appropriati. Il vantaggio principale di questi circuiti è la possibilità di costruire nuovi circuiti hardware in poche ore, piuttosto che attendere i mesi necessari per fabbricare circuiti integrati. Nonostante ciò, questi ultimi non sono sul viale del tramonto, essendo ancora in possesso di un significativo vantaggio di costo rispetto agli FPGA sui grandi volumi; sono inoltre più veloci e consumano meno energia. A causa dei loro vantaggi in fase di progettazione, tuttavia, gli FPGA vengono spesso utilizzati per la prototipazione e la progettazione di applicazioni su piccola scala.

Guardiamo ora all’interno di un FPGA e cerchiamo di capire come può essere utilizzato per implementare una vasta gamma di circuiti logici. Il chip FPGA contiene due componenti principali che vengono replicati più volte: le LUT (*Look-Up Table*, tabelle di ricerca) e le interconnessioni programmabili. Esaminiamo il modo in cui questi componenti vengono utilizzati.

Una LUT, mostrata nella Figura 3.33(a), è una piccola memoria programmabile che produce un segnale in uscita che viene poi trasmesso alla interconnessione programmabile ed eventualmente a un registro. La memoria programmabile viene usata per creare una funzione logica arbitraria. La LUT nella figura ha una memoria  $16 \times 4$  che può emulare qualsiasi circuito logico con 4 bit di ingresso e 4 bit di uscita. La programmazione della LUT richiede il caricamento nella memoria delle risposte appropriate della logica combinatoria che deve essere emulata. In altre parole, se la logica combinatoria produce il valore di Y per un dato ingresso X, il valore Y va scritto nella LUT all’indice X.

L’esempio nella Figura 3.33(b) mostra come una singola LUT a 4 ingressi potrebbe implementare un contatore a 3 bit con reset. Il contatore dell’esempio continua a contare aggiungendo uno (modulo 4) al valore corrente del contatore, a meno che il segnale di reset CLR venga asserito, nel qual caso il contatore riporta il suo valore a zero.

Per implementare il contatore dell’esempio, le quattro entry superiori della LUT sono poste a zero. Queste voci restituiscono il valore zero quando il contatore viene resettato. Quindi, il bit più significativo della LUT ( $I_{0..3}$ ) rappresenta l’ingresso di reset (CLR), che è asserito con un valore logico 1. Nelle rimanenti voci della LUT il valore di indice  $I_{0..3}$  contiene  $(I + 1)$  modulo 4. Per completare il progetto, i segnali in uscita  $O_{0..3}$  devono essere collegati, tramite l’interconnessione programmabile, ai segnali interni di ingresso  $I_{0..3}$ .

Per capire meglio il contatore con reset realizzato con un FPGA, consideriamo le operazioni che esegue. Se, per esempio, lo stato corrente del contatore è 2 e il segnale di reset non è asserito, l’indirizzo di input della LUT sarà 2 e l’output verso i flip-flop sarà di conseguenza 3. Se, nello stesso stato, il segnale di reset (CLR) viene asserito, l’input della LUT sarà 6 e quindi lo stato successivo sarà 0.

Può sembrare, tutto sommato, un modo arcano per costruire un contatore con reset e in effetti un progetto completamente personalizzato con un circuito contatore e segnali di ripristino al flip-flop sarebbe più piccolo, più veloce, e consumerebbe meno energia.

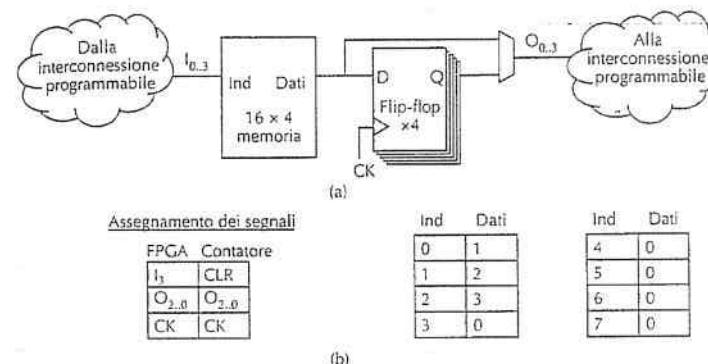


Figura 3.33 (a) La LUT di un FPGA. (b) La configurazione della LUT per la realizzazione di un contatore a 3 bit con reset.

Il vantaggio principale del progetto basato su FPGA è che si può costruire artigianalmente a casa in un’ora, mentre il più efficiente design completamente personalizzato deve essere fabbricato a partire dal silicio e l’operazione potrebbe richiedere anche più di un mese.

Per utilizzare un FPGA il progetto deve essere descritto con una rappresentazione del circuito o con un linguaggio di descrizione hardware (cioè un linguaggio di programmazione usato per descrivere strutture hardware). Il progetto viene poi elaborato da un sintetizzatore, che mappa il circuito su un’architettura FPGA specifica. Succede spesso che il progetto che si desidera realizzare non possa essere mappato sull’FPGA. Gli FPGA sono realizzati con un numero variabile di LUT e quelli che ne hanno di più costano di più. In generale, se il vostro progetto non si adatta all’FPGA, è necessario semplificarlo o rinunciare ad alcune funzionalità, oppure acquistare un FPGA più grande (e più costoso). Progetti molto grandi potrebbero non trovare spazio nemmeno nel più grande FPGA. In questo caso il progettista dovrà mappare il progetto in più FPGA; questo compito è sicuramente più difficile, ma è ancora una passeggiata rispetto alla progettazione di un circuito integrato personalizzato.

## 3.4 Chip della CPU e bus

Forti delle conoscenze acquisite sui circuiti integrati, sui clock e sui circuiti di memoria possiamo ora cominciare a mettere insieme i vari pezzi per affrontare il sistema nel suo complesso. In questo paragrafo analizzeremo inizialmente alcuni aspetti generali delle CPU dal punto di vista del livello logico digitale, contatti compresi (significato dei segnali sui singoli pin), e forniremo anche un’introduzione all’architettura dei bus, dato che le CPU dipendono strettamente dal modo in cui questi sono progettati. Nei paragrafi successivi daremo esempi più dettagliati di CPU, bus, e loro interfacce.

## FPGA

Come abbiamo visto nel Capitolo 1, gli **FPGA** (*field-programmable gate array*) sono circuiti integrati che contengono una logica programmabile e permettono di formare un circuito logico arbitrario semplicemente caricando l’FPGA con i dati di configurazione appropriati. Il vantaggio principale di questi circuiti è la possibilità di costruire nuovi circuiti hardware in poche ore, piuttosto che attendere i mesi necessari per fabbricare circuiti integrati. Nonostante ciò, questi ultimi non sono sul viale del tramonto, essendo ancora in possesso di un significativo vantaggio di costo rispetto agli FPGA sui grandi volumi; sono inoltre più veloci e consumano meno energia. A causa dei loro vantaggi in fase di progettazione, tuttavia, gli FPGA vengono spesso utilizzati per la prototipazione e la progettazione di applicazioni su piccola scala.

Guardiamo ora all’interno di un FPGA e cerchiamo di capire come può essere utilizzato per implementare una vasta gamma di circuiti logici. Il chip FPGA contiene due componenti principali che vengono replicati più volte: le **LUT** (*Look-Up Table*, tabelle di ricerca) e le **interconnessioni programmabili**. Esaminiamo il modo in cui questi componenti vengono utilizzati.

Una LUT, mostrata nella Figura 3.33(a), è una piccola memoria programmabile che produce un segnale in uscita che viene poi trasmesso alla interconnessione programmabile ed eventualmente a un registro. La memoria programmabile viene usata per creare una funzione logica arbitraria. La LUT nella figura ha una memoria  $16 \times 4$  che può emulare qualsiasi circuito logico con 4 bit di ingresso e 4 bit di uscita. La programmazione della LUT richiede il caricamento nella memoria delle risposte appropriate della logica combinatoria che deve essere emulata. In altre parole, se la logica combinatoria produce il valore di Y per un dato ingresso X, il valore Y va scritto nella LUT all’indice X.

L’esempio nella Figura 3.33(b) mostra come una singola LUT a 4 ingressi potrebbe implementare un contatore a 3 bit con reset. Il contatore dell’esempio continua a contare aggiungendo uno (modulo 4) al valore corrente del contatore, a meno che il segnale di reset CLR venga asserito, nel qual caso il contatore riporta il suo valore a zero.

Per implementare il contatore dell’esempio, le quattro entry superiori della LUT sono poste a zero. Queste voci restituiscono il valore zero quando il contatore viene resettato. Quindi, il bit più significativo della LUT ( $I_{3,0}$ ) rappresenta l’ingresso di reset (CLR), che è asserito con un valore logico 1. Nelle rimanenti voci della LUT il valore di indice  $I_{0,3}$  contiene  $(I + 1)$  modulo 4. Per completare il progetto, i segnali in uscita  $O_{0,3}$  devono essere collegati, tramite l’interconnessione programmabile, ai segnali interni di ingresso  $I_{0,3}$ .

Per capire meglio il contatore con reset realizzato con un FPGA, consideriamo le operazioni che esegue. Se, per esempio, lo stato corrente del contatore è 2 e il segnale di reset non è asserito, l’indirizzo di input della LUT sarà 2 e l’output verso i flip-flop sarà di conseguenza 3. Se, nello stesso stato, il segnale di reset (CLR) viene asserito, l’input della LUT sarà 6 e quindi lo stato successivo sarà 0.

Può sembrare, tutto sommato, un modo arcano per costruire un contatore con reset e in effetti un progetto completamente personalizzato con un circuito contatore e segnali di ripristino al flip-flop sarebbe più piccolo, più veloce, e consumerebbe meno energia.

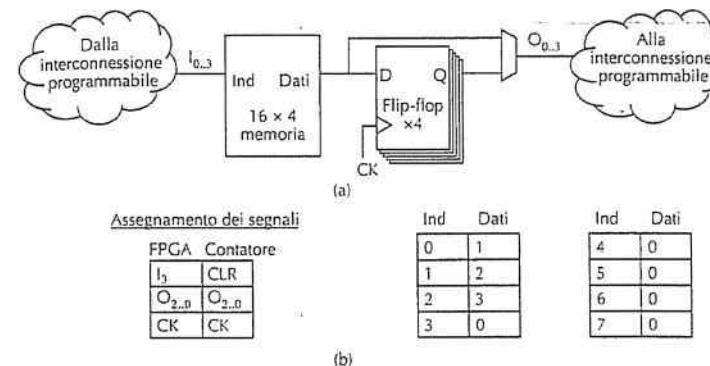


Figura 3.33 (a) La LUT di un FPGA. (b) La configurazione della LUT per la realizzazione di un contatore a 3 bit con reset.

Il vantaggio principale del progetto basato su FPGA è che si può costruire artigianalmente a casa in un’ora, mentre il più efficiente design completamente personalizzato deve essere fabbricato a partire dal silicio e l’operazione potrebbe richiedere anche più di un mese.

Per utilizzare un FPGA il progetto deve essere descritto con una rappresentazione del circuito o con un linguaggio di descrizione hardware (cioè un linguaggio di programmazione usato per descrivere strutture hardware). Il progetto viene poi elaborato da un sintetizzatore, che mappa il circuito su un’architettura FPGA specifica. Succede spesso che il progetto che si desidera realizzare non possa essere mappato sull’FPGA. Gli FPGA sono realizzati con un numero variabile di LUT e quelli che ne hanno di più costano di più. In generale, se il vostro progetto non si adatta all’FPGA, è necessario semplificarlo o rinunciare ad alcune funzionalità, oppure acquistare un FPGA più grande (e più costoso). Progetti molto grandi potrebbero non trovare spazio nemmeno nel più grande FPGA. In questo caso il progettista dovrà mappare il progetto in più FPGA; questo compito è sicuramente più difficile, ma è ancora una passeggiata rispetto alla progettazione di un circuito integrato personalizzato.

## 3.4 Chip della CPU e bus

Forti delle conoscenze acquisite sui circuiti integrati, sui clock e sui circuiti di memoria possiamo ora cominciare a mettere insieme i vari pezzi per affrontare il sistema nel suo complesso. In questo paragrafo analizzeremo inizialmente alcuni aspetti generali delle CPU dal punto di vista del livello logico digitale, **contatti** compresi (significato dei segnali sui singoli pin), e forniremo anche un’introduzione all’architettura dei bus, dato che le CPU dipendono strettamente dal modo in cui questi sono progettati. Nei paragrafi successivi daremo esempi più dettagliati di CPU, bus, e loro interfacce.

### 3.4.1 Chip della CPU

Tutte le CPU moderne sono contenute in un unico chip, rendendo in questo modo ben definita l'interazione con il resto del sistema. Ogni chip di CPU ha un insieme di pin attraverso il quale passano tutte le relative comunicazioni verso il mondo esterno. Alcuni pin spediscono i segnali della CPU, altri ricevono quelli del mondo esterno e altri ancora possono fare entrambe le cose. Conoscendo la funzione di tutti i pin possiamo capire in quale modo la CPU interagisce con la memoria e i dispositivi di I/O nel livello logico digitale.

I pin di una CPU possono essere divisi in tre tipi: indirizzi, dati e controlli. Questi pin sono collegati ad analoghi pin presenti sulla memoria, o sui chip di I/O, mediante un insieme di cavi paralleli chiamato bus. La CPU, per prelevare un'istruzione, ne impone l'indirizzo sui suoi pin di indirizzamento, e poi asserisce una o più linee di controllo per informare la memoria che vuole leggere una parola. La memoria risponde spedendo la parola richiesta sui pin della CPU dedicati ai dati e asserendo un segnale che notifica che l'operazione è stata completata. Quando la CPU vede questo segnale accetta la parola ed esegue l'istruzione.

Se l'istruzione richiede la lettura o la scrittura di certi dati, l'intero processo viene ripetuto per ogni parola di dati. In seguito entreremo nei dettagli, ma per il momento è importante capire che la CPU comunica con la memoria e i dispositivi di I/O inviando e accettando segnali sui propri pin. Non esiste altro tipo di comunicazione.

Due dei principali parametri che determinano le prestazioni di una CPU sono il numero di pin d'indirizzo e il numero di pin di dati. Un chip con  $m$  pin d'indirizzo può indirizzare fino a  $2^m$  locazioni di memoria; valori comuni di  $m$  sono 16, 20, 32 e 64. Analogamente un chip con  $n$  pin di dati può leggere o scrivere in un'unica operazione una parola a  $n$  bit (valori comuni di  $n$  sono 8, 32 e 64). Una CPU con 8 pin di dati dovrà eseguire quattro operazioni per leggere una parola a 32 bit, mentre una CPU con 32 pin di dati può compiere lo stesso lavoro in una sola operazione. Il chip con 32 pin di dati è quindi molto più veloce, ma molto più costoso.

Oltre ai pin d'indirizzo e dei dati, le CPU sono dotate anche di alcuni pin di controllo. Questi regolano il flusso e la temporizzazione dei dati da e verso la CPU, e possono essere utilizzati anche in vari altri modi. Tutte le CPU hanno pin per l'alimentazione (generalmente da +1,2 a +1,5 volt) e per la terra, oltre a un segnale di clock (un'onda quadra con una particolare frequenza); gli altri pin variano invece in modo considerevole da chip a chip. Ciononostante i pin di controllo possono essere approssimativamente raggruppati nelle seguenti categorie principali:

1. controllo del bus;
2. interrupt;
3. arbitraggio del bus;
4. comunicazione con il coprocessore;
5. stato;
6. altro.

In seguito descriveremo brevemente queste categorie e forniremo maggiori dettagli quando studieremo i chip Intel Core i7, TI OMAP4430 e Atmel ATmega168. La Figura 3.34 mostra un generico chip di una CPU che usa questi gruppi di segnali.

I pin per il controllo del bus mandano principalmente sul bus dei segnali dalla CPU (destinati alla memoria e ai chip di I/O) per notificare quando la CPU vuole leggere dalla o scrivere in memoria oppure compiere altre azioni. La CPU utilizza questi pin per controllare il resto del sistema e comunicargli quali operazioni intende compiere.

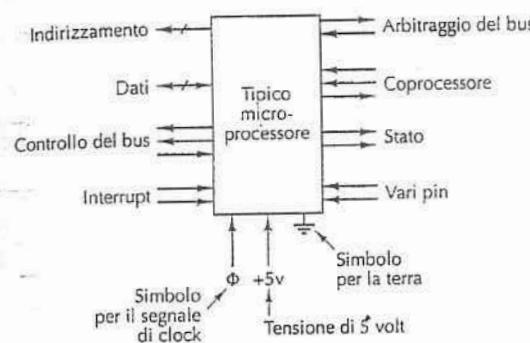


Figura 3.34 Una generica CPU. Le frecce indicano i segnali di input e di output. I trattini diagonali indicano pin multipli; per una specifica CPU un valore ne indica la quantità.

I pin di interrupt sono degli input che giungono alla CPU dalle periferiche. Nella maggior parte dei sistemi la CPU può comunicare a un dispositivo di I/O l'inizio di un'operazione e, mentre questo porta avanti il proprio lavoro, passa a eseguire qualche altra operazione. Non appena il dispositivo ha completato la propria operazione asserisce un segnale su uno di questi pin in modo da interrompere la CPU e permetterle di utilizzare il dispositivo, per esempio per controllare se si siano verificati degli errori di I/O. Alcune CPU hanno un pin di output per spedire una conferma in risposta a un segnale di interrupt.

I pin per l'arbitraggio del bus sono necessari per regolare il traffico sul bus, allo scopo di evitare che due dispositivi cerchino di usarlo nello stesso momento. Ai fini dell'arbitraggio la CPU conta quanto un altro dispositivo e anch'essa deve fare esplicita richiesta di utilizzo del bus.

Alcune CPU sono progettate per funzionare con coprocessori come i chip in virgola mobile oppure, in alcuni casi, i chip grafici o di altro tipo. Per facilitare la comunicazione tra CPU e coprocessore sono disponibili pin speciali per ricevere e soddisfare vari tipi di richieste.

Oltre a questi segnali alcune CPU possono o per resettare il calcolatore o effettuare operazioni di debugging, avere altri pin per altri scopi, per esempio per fornire o ricevere informazioni di stato, o ancora per garantire compatibilità con chip di I/O più vecchi.

### 3.4.2 Bus del calcolatore

Un bus è un collegamento elettrico che unisce diversi dispositivi. I bus possono essere classificati in base alla loro funzione; alcuni di loro sono impiegati internamente alla CPU per trasferire dati da e verso la ALU, mentre altri sono esterni alla CPU e servono a connetterla con la memoria o con altri dispositivi di I/O. Ciascun tipo di bus soddisfa certi requisiti e gode di proprietà specifiche. In questo paragrafo e nei successivi ci concentreremo sui bus che connettono la CPU alla memoria e ai dispositivi di I/O, mentre nel prossimo capitolo esamineremo in modo più dettagliato i bus interni alla CPU.

I primi PC avevano un unico bus esterno chiamato anche *bus di sistema*. Esso era composto da 50 a 100 fili paralleli di rame che si inserivano nella scheda madre e i cui connettori erano distanziati a intervalli regolari per permettere l'inserimento di memorie e schede di I/O. Generalmente i personal computer moderni hanno invece un bus specifico tra la CPU e la memoria e (almeno) un altro bus per le periferiche. La Figura 3.35 mostra un sistema minimale, composto da un bus di memoria e un bus di I/O.

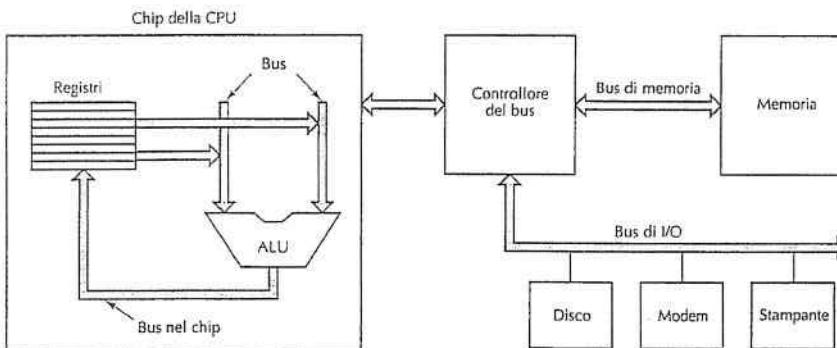


Figura 3.35 Sistema di un calcolatore con più bus.

Come spesso avviene in letteratura i bus sono stati disegnati nella figura come frecce "larghe". Vi è una sottile differenza tra una linea spessa e una linea intersecata da un trattino diagonale vicino al quale si legge un numero (di bit). Quando tutti i bit sono dello stesso tipo, cioè sono, per esempio, bit d'indirizzo oppure bit di dati, si usa comunemente il trattino diagonale. Quando invece sono coinvolte linee d'indirizzo, di dati e di controllo si utilizza con più frequenza la freccia spessa.

Mentre i progettisti della CPU sono liberi di utilizzare all'interno del chip il tipo di bus che preferiscono, per i bus esterni bisogna invece definire delle precise regole di funzionamento, che devono essere rispettate dai dispositivi a loro collegati. La cosa consente che anche schede progettate da altri produttori possano collegarsi correttamente al bus. L'insieme di queste regole è detto **protocollo del bus**. Inoltre devono essere definite le specifiche meccaniche ed elettriche in modo che le schede prodotte da terze

parti abbiano le dimensioni corrette e i loro connettori siano meccanicamente compatibili con quelli della scheda madre in termini di tensione, temporizzazione e così via. Altri bus non dispongono di specifiche meccaniche, perché sono progettati per essere utilizzati esclusivamente all'interno di un circuito integrato, per esempio per connettere tra loro le componenti all'interno di un SoC (*system-on-a-chip*).

Nel mondo dei calcolatori esiste un gran numero di bus ampiamente utilizzati. Alcuni fra i più conosciuti, sia attuali sia d'importanza storica, sono i seguenti (tra parentesi sono indicati i calcolatori che li hanno adottati): l'Omnibus (PDP-8), l'Unibus (PDP-11), il Multibus (8086), il bus VME (apparecchiature per i laboratori di fisica), il bus PC IBM (PC/XT), il bus ISA (PC/AT), il bus EISA (80386), il Microchannel (PS/2), il Nubus (Macintosh), il bus PCI (molti PC), il bus SCSI (molti PC e workstation), l'Universal Serial Bus (PC moderni) e il FireWire (elettronica di consumo). Il mondo sarebbe probabilmente migliore se tutti, tranne uno, sparissero di colpo dalla faccia della terra (d'accordo, è un po' troppo, ma che cosa dire se ne scomparissero tutti tranne due?). Sfortunatamente sembra molto difficile che avvenga una standardizzazione in quest'area, in quanto sono stati già fatti troppi investimenti in tutti questi sistemi, fra loro incompatibili.

Master	Slave	Esempio
CPU	Memoria	Prelievo delle istruzioni e dei dati
CPU	Dispositivo di I/O	Inizio del trasferimento dei dati
CPU	Coprocessore	Passaggio dell'istruzione al coprocessore da parte della CPU
I/O	Memoria	DMA (Direct Memory Access)
Coprocessore	CPU	Prelievo degli operandi dalla CPU da parte del coprocessore

Figura 3.36 Esempi di master e slave del bus.

Cominciamo il nostro studio partendo dal funzionamento dei bus. Alcune periferiche che si collegano al bus sono attive e possono iniziare un trasferimento dati, mentre altre sono passive e restano in attesa di una richiesta. Quelle attive sono chiamate **master**, e quelle passive **slave**. Quando la CPU ordina al controllore di un disco di leggere o scrivere un blocco, svolge il ruolo di master, e il controllore del disco quello di slave. Successivamente però il controllore del disco fungerà da master nel momento in cui ordina alla memoria di accettare le parole che sta leggendo dal disco. La Figura 3.36 elenca alcune tipiche combinazioni di master e slave. La memoria non può mai fungere da master.

Molto spesso i segnali digitali generati dalle periferiche sono troppo deboli per alimentare un bus, soprattutto se è relativamente lungo o se è collegato a molti dispositivi. Per questo motivo molti master sono connessi al bus mediante un chip chiamato **driver del bus**, che funge essenzialmente da amplificatore digitale; in modo analogo la maggior parte degli slave sono connessi al bus attraverso un **ricevitore del bus**. Per le periferiche che possono svolgere sia il ruolo di master sia quello di slave si utilizza un chip

chiamato **trasmettitore-ricevitore del bus**. Spesso questi chip d'interfaccia sono dei dispositivi a tre stati per permettere loro di essere liberi (sconnessi) quando non sono necessari; un'altra possibilità, che permette di ottenere lo stesso effetto, consiste nell'aganciare la periferica al bus tramite un **collettore aperto**. Quando due o più dispositivi su una linea a collettore aperto asseriscono la linea nello stesso istante, il risultato è un OR di tutti i segnali; spesso questa organizzazione è chiamata **OR-cablata**. Nella maggior parte dei bus alcune linee sono a tre stati mentre altre, che richiedono la proprietà di essere OR-cablata, sono a collettore aperto.

Allo stesso modo della CPU anche il bus ha i propri indirizzi, i propri dati e le proprie linee di controllo. Ciononostante non è necessario che vi sia una corrispondenza uno-a-uno tra i pin della CPU e i segnali del bus. Alcune CPU hanno per esempio tre pin che codificano se si sta effettuando una lettura o una scrittura da memoria, una lettura o una scrittura di I/O oppure un'altra operazione. Un normale bus potrebbe avere una linea per la lettura da memoria, una seconda per la scrittura in memoria, una terza per la lettura da periferica, una quarta per la scrittura su periferica e così via. In questo caso si renderebbe necessario un decodificatore tra la CPU e il bus per far corrispondere le due estremità, cioè per convertire il segnale codificato in 3 bit in segnali distinti che possono essere spediti sulle varie linee del bus.

Il progetto e il funzionamento dei bus sono argomenti piuttosto complessi, sono trattati in vari libri (Anderson et al., 2004; Solari e Wills, 2004). Le principali decisioni da prendere nella progettazione di un bus riguardano l'ampiezza, la temporizzazione, l'arbitraggio e le operazioni che consente. Ciascuna di queste scelte ha un impatto significativo sulla velocità e sulla larghezza di banda del bus.

### 3.4.3 Ampiezza del bus

Nella progettazione dei bus il parametro più scontato da considerare è la sua ampiezza. Maggiore è il numero di linee d'indirizzo di un bus, maggiore sarà la quantità di memoria che la CPU potrà indirizzare direttamente. Se un bus ha  $n$  linee d'indirizzo, una CPU può indirizzare  $2^n$  diverse locazioni di memoria. La cosa sembra piuttosto semplice.

Il problema è che bus più larghi richiedono un numero maggiore di fili rispetto a quelli più stretti. Questo significa anche una maggiore occupazione di spazio (per esempio sulla scheda madre) e la necessità di utilizzare connettori più grandi. Dato che tutti questi fattori rendono il bus più costoso occorre trovare un compromesso tra la dimensione massima di memoria e il costo del sistema. Un sistema con 64 linee d'indirizzo e  $2^{32}$  byte di memoria costerà più di uno con 32 linee d'indirizzo e  $2^{32}$  byte di memoria, ma nel secondo caso non è possibile espandere la memoria in un secondo momento.

Il risultato di questa osservazione è che molti progettisti di sistemi tendono a essere poco lungimiranti, con conseguenze negative. Il PC IBM originario conteneva una CPU 8088 e un bus con indirizzi a 20 bit, come mostrato nella Figura 3.37(a); con 20 bit a disposizione il PC poteva indirizzare soltanto 1 MB di memoria.

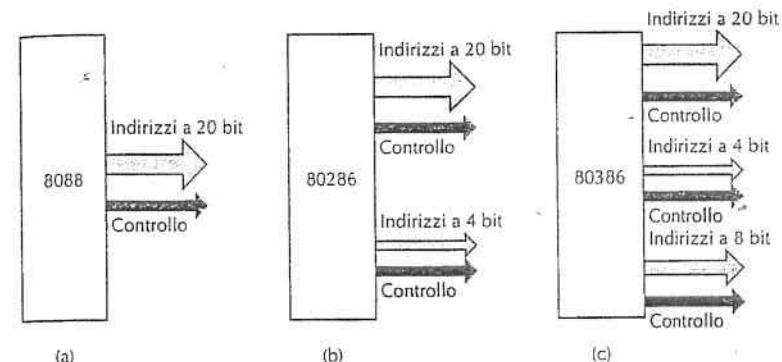


Figura 3.37 Crescita nel tempo degli indirizzi del bus.

Con la CPU successiva, l'80286, Intel decise di aumentare lo spazio degli indirizzi a 16 MB; questo aumento richiese l'aggiunta di quattro linee di bus (senza modificare le 20 precedenti per ragioni di retrocompatibilità), come mostra la Figura 3.37(b). Purtroppo fu necessario aggiungere anche alcune nuove linee di controllo per gestire le nuove linee d'indirizzo. Quando apparve l'80386 furono aggiunte altre otto linee d'indirizzo oltre alle linee di controllo aggiuntive, come mostra la Figura 3.37(c). Il progetto risultante (il bus EISA) è molto meno ordinato di come sarebbe potuto essere se si fosse partiti da zero avendo a disposizione fin dall'inizio tutte e 32 le linee.

Con il tempo, non solo tende a crescere il numero delle linee d'indirizzo, ma anche il numero delle linee di dati, seppur per una ragione diversa. Esistono due modi per aumentare la larghezza di banda dei dati su un bus: diminuire il periodo di clock del bus (più trasferimenti/s) oppure aumentare la larghezza dei dati del bus. Un'altra possibilità è quella di aumentare la velocità del bus, anche se ciò pone alcune difficoltà; i segnali su linee distinte viaggiano infatti a velocità leggermente diverse. Questo problema è conosciuto come **disallineamento del bus**, e più il bus è veloce più questo problema diventa marcato.

Un altro problema che sorge nel caso in cui si intenda velocizzare il bus è la perdita della retrocompatibilità. Schede progettate per bus più lenti non funzioneranno con il nuovo bus. Rendere inutilizzabili le vecchie schede non rende di certo felici né i proprietari di queste schede né i loro produttori. Per questa ragione l'approccio più utilizzato è quello di aggiungere nuove linee di dati, come mostrato nella Figura 3.37. In ogni caso, come ci si potrebbe aspettare, questa crescita incrementale non genera come risultato un'architettura semplice e ordinata. I PC IBM e i loro successori, per fare un esempio, sono passati da otto linee di dati a 16 e, in seguito, a 32 su un bus che è rimasto essenzialmente dello stesso tipo.

Per aggirare il problema di bus troppo ampi a volte i progettisti optano per un **bus multiplexato**. In questa architettura invece di tenere separate le linee d'indirizzo e quelle dei dati, si utilizza un certo numero di linee, per esempio 32, per entrambi. All'inizio

è molto meno ordinato di come sarebbe potuto essere se si fosse partiti da zero avendo a disposizione fin dall'inizio tutte e 32 le linee.

Con il tempo, non solo tende a crescere il numero delle linee d'indirizzo, ma anche il numero delle linee di dati, seppur per una ragione diversa. Esistono due modi per aumentare la larghezza di banda dei dati su un bus: diminuire il periodo di clock del bus (più trasferimenti/s) oppure aumentare la larghezza dei dati del bus. Un'altra possibilità è quella di aumentare la velocità del bus, anche se ciò pone alcune difficoltà; i segnali su linee distinte viaggiano infatti a velocità leggermente diverse. Questo problema è conosciuto come **disallineamento del bus**, e più il bus è veloce più questo problema diventa marcato.

Un altro problema che sorge nel caso in cui si intenda velocizzare il bus è la perdita della retrocompatibilità. Schede progettate per bus più lenti non funzioneranno con il nuovo bus. Rendere inutilizzabili le vecchie schede non rende di certo felici né i proprietari di queste schede né i loro produttori. Per questa ragione l'approccio più utilizzato è quello di aggiungere nuove linee di dati, come mostrato nella Figura 3.37. In ogni caso, come ci si potrebbe aspettare, questa crescita incrementale non genera come risultato un'architettura semplice e ordinata. I PC IBM e i loro successori, per fare un esempio, sono passati da otto linee di dati a 16 e, in seguito, a 32 su un bus che è rimasto essenzialmente dello stesso tipo.

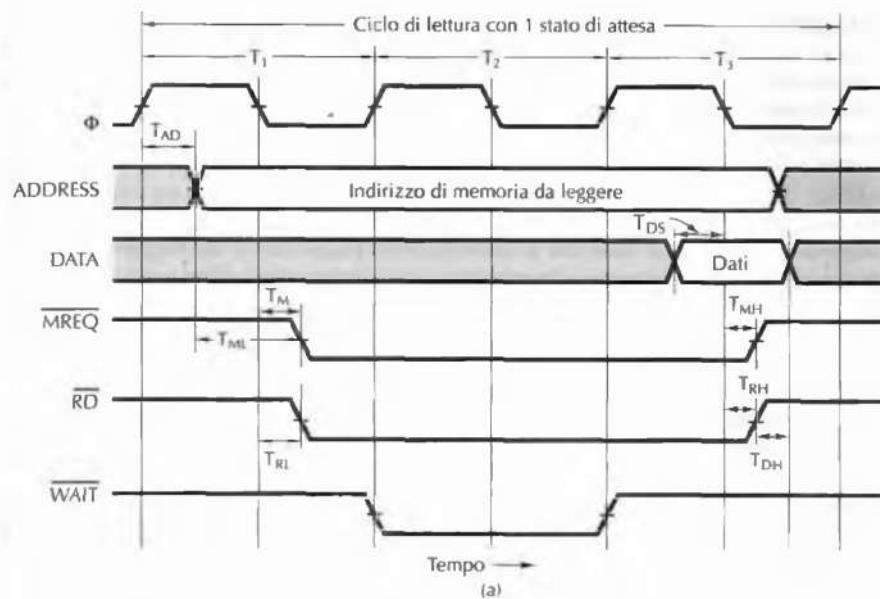
Per aggirare il problema di bus troppo ampi a volte i progettisti optano per un **bus multiplexato**. In questa architettura invece di tenere separate le linee d'indirizzo e quelle dei dati, si utilizza un certo numero di linee, per esempio 32, per entrambi. All'inizio di un'operazione sul bus le linee sono utilizzate per gli indirizzi, mentre in seguito vengono impiegate per i dati. Per esempio nel caso di una scrittura in memoria questo significa che le linee d'indirizzo devono essere impostate ai valori corretti e propagate fino alla memoria prima di spedire i dati sul bus. Quando si utilizzano linee separate è invece possibile spedire gli indirizzi e i dati allo stesso tempo. L'uso del multiplexing riduce l'ampiezza del bus (e quindi il costo), ma rende il sistema più lento. Quando devono prendere le proprie decisioni, i progettisti sono obbligati a valutare attentamente tutte queste possibilità.

#### 3.4.4 Temporizzazione del bus

I bus possono essere separati in due categorie distinte in base alla loro temporizzazione. Un bus sincrono ha una linea pilotata da un oscillatore a cristalli: su questa linea un segnale consiste in un'onda quadra con frequenza generalmente compresa tra 5 e 100 MHz. Tutte le operazioni sul bus richiedono un numero intero di questi cicli, chiamati **cicli di bus**. L'altro tipo di bus, il bus asincrono, non ha invece un orologio principale; i cicli di bus possono avere una qualsiasi lunghezza e non devono essere necessariamente uguali nella comunicazione tra dispositivi.

##### Bus sincroni

Consideriamo la temporizzazione mostrata nella Figura 3.38(a) come esempio del funzionamento di un bus sincrono e riferiamoci a un clock a 100 MHz, che fornisce un ciclo di bus di 10 ns. Anche se questo valore potrebbe sembrare un po' lento rispetto alle velocità delle CPU, superiori a 3 GHz, occorre considerare che solo pochi bus dei PC esistenti sono molto più veloci. Per fare un esempio il bus PCI, molto diffuso, funziona generalmente a



(a)

Simbolo	Parametro	Min	Max	Unità
$T_{AD}$	Ritardo dell'output dell'indirizzo		4	nsec
$T_{ML}$	Indirizzo stabile prima di $\overline{MREQ}$		2	nsec
$T_M$	Ritardo di $MREQ$ rispetto al fronte di discesa di $\Phi$ in $T_1$	3		nsec
$T_{RI}$	Ritardo di $RD$ rispetto al fronte di discesa di $\Phi$ in $T_1$	3		nsec
$T_{DS}$	Tempo di impostaz. dei dati prima del fronte di discesa di $\Phi$	2		nsec
$T_{MH}$	Ritardo di $MREQ$ rispetto al fronte di discesa di $\Phi$ in $T_3$		3	nsec
$T_{RH}$	Ritardo di $RD$ rispetto al fronte di discesa di $\Phi$ in $T_3$	3		nsec
$T_{DH}$	Tempo di mantenimento dei dati dopo la negazione di $RD$	0		nsec

(b)

Figura 3.38 (a) Temporizzazione di una lettura su un bus sincrono. (b) Specifiche di alcuni tempi critici.

33 MHz oppure a 66 MHz. Le ragioni per le quali i bus attuali sono lenti sono già state sottolineate (disallineamento, retrocompatibilità, ecc).

Nel nostro esempio assumeremo inoltre che la lettura dalla memoria richieda 15 ns a partire dal momento in cui l'indirizzo è stabile. Con questi parametri, come vedremo a breve, saranno necessari tre cicli di bus per leggere una parola. Come mostra la figura, il primo ciclo inizia in corrispondenza del fronte di salita di  $T_1$  e il terzo termina nel fronte

di salita di  $T_4$ . Si noti che i fronti di salita e di discesa non sono stati disegnati verticalmente in quanto nessun segnale elettrico può cambiare il proprio valore in un tempo nullo. In questo esempio assumeremo che il segnale impieghi 1 ns per cambiare valore. Il clock e le linee ADDRESS, DATA,  $\overline{MREQ}$ ,  $\overline{RD}$  e  $\overline{WAIT}$  sono tutti mostrati usando la stessa scala temporale.

L'inizio di  $T_1$  è definito dal fronte di salita del clock. A partire da  $T_1$  la CPU fornisce l'indirizzo della parola sulle linee d'indirizzo. Dato che l'indirizzo non è un singolo valore, come invece il clock, non possiamo mostrarlo nella figura come una singola linea; viene quindi rappresentato con due linee che si incrociano nel momento in cui l'indirizzo cambia. L'ombreggiatura prima del punto d'intersezione indica inoltre che il valore non è importante. Adottando la stessa convenzione vediamo che il contenuto delle linee dei dati non è significativo fino a  $T_3$ .

Dopo che le linee d'indirizzo si sono stabilizzate sui nuovi valori, vengono asserite  $\overline{MREQ}$  e  $\overline{RD}$ . La prima indica che si sta per accedere alla memoria (invece che a un dispositivo di I/O), mentre la seconda è asserita per le letture e negata per le scritture. Dato che la memoria impiega 15 ns dopo che l'indirizzo è stabile (inizialmente durante il primo ciclo di clock) non può fornire i dati richiesti durante  $T_2$ . La memoria asserisce la linea  $\overline{WAIT}$  all'inizio di  $T_2$  per segnalare alla CPU di non aspettarla.

Questa azione inserisce alcuni **stati di attesa** (cicli di bus addizionali) finché la memoria non completa l'operazione e neghi  $\overline{WAIT}$ . Nell'esempio è stato inserito uno stato di attesa ( $T_2$ ), dato che la memoria è troppo lenta. All'inizio di  $T_3$  la memoria nega  $\overline{WAIT}$ , in quanto è sicuro che i dati saranno disponibili nel corso del ciclo corrente.

Durante la prima metà di  $T_3$ , la memoria mette i dati sulle linee appropriate. Nel fronte di discesa di  $T_3$  la CPU consulta (cioè legge) le linee dei dati, memorizzando il valore in un suo registro. Dovendo leggere i dati la CPU nega  $\overline{MREQ}$  e  $\overline{RD}$ . Se necessario può cominciare un altro ciclo di memoria nel successivo fronte di salita del clock. Questa sequenza può essere ripetuta indefinitamente.

Chiariamo ora il significato degli otto simboli presenti nel diagramma della Figura 3.38(b) che elenca le specifiche di temporizzazione. Per esempio,  $T_{AD}$  è l'intervallo di tempo tra il fronte di salita del ciclo di clock  $T_1$  e il momento in cui vengono impostate le linee d'indirizzo. Secondo le specifiche di temporizzazione,  $T_{AD} \leq 4$  ns. Questo significa che il produttore della CPU garantisce che durante ogni ciclo di lettura la CPU manderà in output l'indirizzo da leggere entro 11 ns dal punto medio del fronte di salita di  $T_1$ .

Le specifiche di temporizzazione richiedono inoltre che i dati siano disponibili sulle linee almeno  $T_{DS}$  ns (cioè 2 ns) prima del fronte di discesa di  $T_3$ . La cosa è necessaria per dare il tempo alla linea di stabilizzarsi prima che la CPU legga. La combinazione dei vincoli su  $T_{AD}$  e  $T_{DS}$  fa sì che, nel caso peggiore, la memoria abbia a disposizione soltanto  $25 - 4 - 2 = 19$  ns tra il momento in cui appare l'indirizzo e quello in cui deve fornire i dati. Dato che sono sufficienti 10 ns, anche nel caso peggiore una memoria a 10 ns è sempre in grado di fornire la risposta durante il ciclo  $T_3$ . Al contrario una memoria a 20 ns potrebbe non farcela in tempo e in tal caso dovrebbe inserire un secondo stato di attesa e rispondere durante  $T_4$ .

Le specifiche di temporizzazione garantiscono inoltre che l'indirizzo venga impostato almeno 2 ns prima che  $\overline{MREQ}$  sia asserito. Questo intervallo temporale può essere significativo nel caso in cui  $\overline{MREQ}$  piloti la selezione del chip sulla memoria; questo poiché alcune memorie richiedono un certo tempo per l'impostazione dell'indirizzo prima di selezionare il chip. È

Questa azione inserisce alcuni stati di attesa (cicli di bus addizionali) finché la memoria non completa l'operazione e neghi WAIT. Nell'esempio è stato inserito uno stato di attesa ( $T_2$ ), dato che la memoria è troppo lenta. All'inizio di  $T_3$  la memoria nega WAIT, in quanto è sicuro che i dati saranno disponibili nel corso del ciclo corrente.

Durante la prima metà di  $T_3$ , la memoria mette i dati sulle linee apposite. Nel fronte di discesa di  $T_3$  la CPU consulta (cioè legge) le linee dei dati, memorizzando il valore in un suo registro. Dovendo leggere i dati la CPU nega MREQ e RD. Se necessario può cominciare un altro ciclo di memoria nel successivo fronte di salita del clock. Questa sequenza può essere ripetuta indefinitamente.

Chiariamo ora il significato degli otto simboli presenti nel diagramma della Figura 3.38(b) che elenca le specifiche di temporizzazione. Per esempio,  $T_{AD}$  è l'intervallo di tempo tra il fronte di salita del ciclo di clock  $T_1$  e il momento in cui vengono impostate le linee d'indirizzo. Secondo le specifiche di temporizzazione,  $T_{AD} \leq 4$  ns. Questo significa che il produttore della CPU garantisce che durante ogni ciclo di lettura la CPU manderà in output l'indirizzo da leggere entro 4 ns dal punto medio del fronte di salita di  $T_1$ .

Le specifiche di temporizzazione richiedono inoltre che i dati siano disponibili sulle linee almeno  $T_{DS}$  ns (cioè 2 ns) prima del fronte di discesa di  $T_3$ . La cosa è necessaria per dare il tempo alla linea di stabilizzarsi prima che la CPU legga. La combinazione dei vincoli su  $T_{AD}$  e  $T_{DS}$  fa sì che, nel caso peggiore, la memoria abbia a disposizione soltanto  $25 - 4 - 2 = 19$  ns tra il momento in cui appare l'indirizzo e quello in cui deve fornire i dati. Dato che sono sufficienti 15 ns, anche nel caso peggiore una memoria a 15 ns è sempre in grado di fornire la risposta durante il ciclo  $T_3$ . Al contrario una memoria a 20 ns potrebbe non farcela in tempo e in tal caso dovrebbe inserire un secondo stato di attesa e rispondere durante  $T_4$ .

Le specifiche di temporizzazione garantiscono inoltre che l'indirizzo venga impostato almeno 2 ns prima che MREQ sia asserito. Questo intervallo temporale può essere significativo nel caso in cui MREQ piloti la selezione del chip sulla memoria; questo poiché alcune memorie richiedono un certo tempo per l'impostazione dell'indirizzo prima di selezionare il chip. È chiaro che un progettista di CPU non dovrebbe scegliere un chip di memoria che richiede un tempo di setup di 3 ns.

I vincoli su  $T_M$  e  $T_{RL}$  fanno sì che MREQ e RD vengano asseriti entro 3 ns dalla discesa del clock  $T_1$ . Nel caso peggiore il chip di memoria, per mettere i propri dati sul bus, avrà a disposizione solo  $10 + 10 - 3 - 2 = 15$  ns dopo che MREQ e RD sono stati asseriti. Questo vincolo si aggiunge (e ne è indipendente) all'attesa di 15 ns necessaria dopo che l'indirizzo è stabile.

$T_{MH}$  e  $T_{RH}$  indicano quanto tempo impiegano MREQ e RD per essere negati dopo che i dati sono stati generati. Infine  $T_{DH}$  specifica per quanto tempo la memoria deve mantenere i dati sul bus dopo che RD è stato negato. Per quanto riguarda il nostro esempio di CPU, la memoria può rimuovere i dati dal bus non appena viene negato RD; tuttavia su alcune CPU reali i dati devono essere mantenuti stabili per un tempo leggermente più lungo.

È importante sottolineare che la Figura 3.38 è una versione decisamente semplificata dei reali vincoli di temporizzazione. Anche se nella realtà vengono specificati molti

altri tempi critici, l'esempio riesce tuttavia a dare l'idea del funzionamento di un bus sincrono.

Un altro punto che vale la pena precisare è che i segnali di controllo possono essere asseriti con valore alto o con valore basso. Tocca al progettista del bus determinare quale dei due sia più conveniente, e la scelta è essenzialmente arbitraria. Questa decisione potrebbe essere vista come l'equivalente, in hardware, della scelta che un programmatore compie quando deve decidere se rappresentare un settore vuoto del disco mediante valori 0 o valori 1.

### Bus asincroni

Lavorare con i bus sincroni è facile per via dei loro intervalli temporali discreti; ciononostante presentano alcuni problemi. Per fare un esempio, qualsiasi operazione sul bus si svolge in tempi multipli del clock del bus; se una CPU e una memoria sono in grado di completare un trasferimento in 3,1 cicli, sono tuttavia obbligate ad allungare il tempo necessario a 4 cicli, dato che non sono consentite frazioni di cicli.

Inoltre, una volta scelto un ciclo di bus e appositamente costruite le memorie e le schede di I/O, è difficile trarre vantaggio dai successivi sviluppi della tecnologia. Supponiamo per esempio che alcuni anni dopo la realizzazione del sistema della Figura 3.38 si rendano disponibili nuove memorie con tempi di accesso di 8 ns invece che di 15 ns. Con la nuova memoria si eliminerebbe il tempo di attesa, accelerando di conseguenza la macchina. Supponiamo ora che diventino disponibili memorie a 4 ns; ciò non porterebbe ad alcun ulteriore guadagno in termini di prestazioni, poiché con questa architettura il tempo minimo per una lettura è di due cicli. In altri termini, se un bus sincrono ha un insieme eterogeneo di dispositivi, alcuni veloci, altri più lenti, il bus deve essere regolato alla velocità della periferica più lenta, e le altre, pur essendo più veloci, non possono sfruttare tutto il loro potenziale.

È possibile gestire tecnologie caratterizzate da prestazioni diverse mediante l'utilizzo di un bus asincrono, cioè di un bus che non possiede un clock principale, come mostra la Figura 3.39. Il master del bus, invece di legare ogni operazione al clock, dopo aver asserito l'indirizzo, MREQ, RD e tutto ciò che gli necessita, asserisce uno speciale segnale che chiameremo MSYN (*Master SYNchronization*). Quando lo slave vede questo segnale esegue il lavoro richiesto alla massima velocità possibile; una volta terminato il proprio compito asserisce SSYN (*Slave SYNchration*).

Quando il master vede che SSYN è stato asserito, sa che i dati sono disponibili e può quindi memorizzarli; successivamente nega le linee d'indirizzo, MREQ, RD e MSYN. Quando lo slave vede la negazione di MSYN, sa che il ciclo è stato completato e quindi nega SSYN; a questo punto si è tornati alla situazione iniziale, nella quale tutti i segnali sono negati e si attende il master successivo.

Come mostra la Figura 3.39 i diagrammi di temporizzazione dei bus asincroni (e talvolta anche dei bus sincroni) usano frecce per indicare cause ed effetti. L'asserzione di MSYN ha come effetto l'asserzione delle linee dei dati e il fatto che lo slave asserisce SSYN. L'asserzione di SSYN causa, a sua volta, l'asserzione delle linee d'indirizzo, di MREQ, di RD e di MSYN. La negazione di MSYN provoca infine la negazione di SSYN, che termina la lettura e riporta il sistema al suo stato originario.

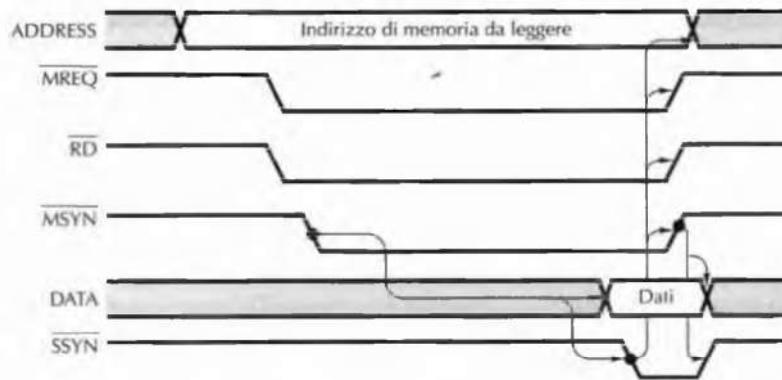


Figura 3.39 Funzionamento di un bus asincrono.

Quando il master vede che  $\overline{SSYN}$  è stato asserito, sa che i dati sono disponibili e può quindi memorizzarli; successivamente nega le linee d'indirizzo,  $\overline{MREQ}$ ,  $\overline{RD}$  e  $\overline{MSYN}$ . Quando lo slave vede la negazione di  $\overline{MSYN}$ , sa che il ciclo è stato completato e quindi nega  $\overline{SSYN}$ ; a questo punto si è tornati alla situazione iniziale, nella quale tutti i segnali sono negati e si attende il master successivo.

Come mostra la Figura 3.39 i diagrammi di temporizzazione dei bus asincroni (e talvolta anche dei bus sincroni) usano frecce per indicare cause ed effetti. L'assezione di  $\overline{MSYN}$  ha come effetto l'assezione delle linee dei dati e il fatto che lo slave asserisce  $\overline{SSYN}$ . L'assezione di  $\overline{SSYN}$  causa, a sua volta, l'assezione delle linee d'indirizzo, di  $\overline{MREQ}$ , di  $\overline{RD}$  e di  $\overline{MSYN}$ . La negazione di  $\overline{MSYN}$  provoca infine la negazione di  $\overline{SSYN}$ , che termina la lettura e riporta il sistema al suo stato originario.

Un insieme di segnali che coordina in questo modo due dispositivi con lo scopo di non farli interferire è chiamato **full handshake** ("stretta di mano completa"); esso consiste essenzialmente di quattro eventi:

1.  $\overline{MSYN}$  è asserito
2.  $\overline{SSYN}$  è asserito in risposta a  $\overline{MSYN}$
3.  $\overline{MSYN}$  è negato in risposta a  $\overline{SSYN}$
4.  $\overline{SSYN}$  è negato in risposta alla negazione di  $\overline{MSYN}$ .

Dovrebbe essere chiaro che i full handshake non dipendono dalla temporizzazione. Ogni evento è causato da un evento precedente e non da un impulso del clock. Se una particolare coppia master-slave è lenta essa non influisce in alcun modo su una successiva coppia master-slave la cui velocità potrebbe essere molto più elevata.

Il vantaggio di un bus asincrono dovrebbe ora apparire evidente. Il problema è che in realtà la maggior parte dei bus è sincrona, in quanto più facile da realizzare; semplicemente la CPU asserisce i propri segnali e la memoria reagisce di conseguenza. Non è presente alcun feedback (causa e effetto), ma, nel caso in cui i componenti siano stati scelti

in modo appropriato, tutto funzionerà correttamente senza dover ricorrere all'handshake. Inoltre occorre dire che gli investimenti effettuati sulla tecnologia dei bus sincroni sono enormi.

### 3.4.5 Arbitraggio del bus

Finora abbiamo assunto tacitamente che la CPU sia l'unico master del bus. In realtà anche i chip di I/O devono diventare master per poter leggere e scrivere in memoria e provocare interrupt, e anche i coprocessori potrebbero avere la necessità di fungere da master. La domanda che sorge è: "Che cosa succede se due o più dispositivi vogliono diventare master del bus nello stesso momento?". La risposta è che si rende necessario qualche forma di **arbitraggio del bus** per evitare confusione.

Il meccanismo di arbitraggio può essere centralizzato o decentralizzato. Consideriamo inizialmente l'arbitraggio centralizzato, di cui una forma particolarmente semplice è mostrata nella Figura 3.40(a). In questo schema un singolo arbitro del bus determina chi sarà il prossimo master. In molte CPU l'arbitro è integrato nel chip stesso della CPU, mentre in altre è necessario utilizzare un chip separato. Il bus contiene un'unica linea di richiesta OR-cablatata che in qualsiasi momento può essere asserita da uno o più dispositivi. L'arbitro non ha alcun modo per sapere quanti dispositivi hanno richiesto il bus; esso può solo distinguere tra "qualche richiesta" e "nessuna richiesta".

Quando l'arbitro vede una richiesta di utilizzo del bus, lo concede asserendo la linea per la concessione del bus. A questa linea sono collegati in serie tutti i dispositivi di I/O, come un semplice filo di lucine natalizie. Quando il dispositivo fisicamente più vicino all'arbitro vede la concessione, effettua un controllo per verificare se ne ha fatto richiesta. In caso affermativo si impossessa del bus, senza però propagare la concessione lungo il resto della linea. Se invece non ha fatto richiesta, propaga la concessione sulla linea in direzione del prossimo dispositivo che si comporterà allo stesso modo, e così pure i successivi, finché un dispositivo accetterà la concessione e si impossesserà del bus. Questo schema è chiamato **collegamento a festone** (*daisy chaining*) ed ha la proprietà di assegnare ai dispositivi diverse priorità in base alla loro vicinanza all'arbitro: il dispositivo più vicino vince.

Molti bus, per aggirare il fatto che le priorità sono implicitamente determinate dalla distanza rispetto all'arbitro, definiscono dei livelli di priorità. Per ciascun livello esiste una linea per effettuare la richiesta e una per segnalare la concessione. Il bus della Figura 3.40(b) ha due livelli, 1 e 2 (i bus reali possono averne 4, 8 o 16). A ciascun dispositivo viene assegnato un livello per la richiesta del bus; a quelli per cui la temporizzazione è più critica vengono assegnate priorità più elevate. Nella Figura 3.40(b) i dispositivi 1, 2 e 4 usano la priorità 1, mentre i dispositivi 3 e 5 usano la priorità 2.

Se in uno stesso momento ci sono richieste da livelli di priorità diversi l'arbitro concede il bus a un dispositivo di priorità più elevata; fra dispositivi della stessa priorità si utilizza invece la strategia del collegamento a festone. Nella Figura 3.40(b) in caso di conflitti il dispositivo 2 ha priorità sul dispositivo 4, che ce l'ha a sua volta sul 3. Il dispositivo 5 ha invece la priorità più bassa fra tutti, in quanto si trova alla fine del collegamento a festone a priorità inferiore.

Aprendo una parentesi occorre precisare che non è tecnicamente necessario collegare serialmente i dispositivi 1 e 2 alla linea di concessione del bus di livello 2, dato che non possono effettuare delle richieste su tale linea. Tuttavia per ragioni d'implementazione è più facile

una linea per effettuare la richiesta e una per segnalare la concessione. Il bus della Figura 3.40(b) ha due livelli, 1 e 2 (i bus reali possono averne 4, 8 o 16). A ciascun dispositivo viene assegnato un livello per la richiesta del bus; a quelli per cui la temporizzazione è più critica vengono assegnate priorità più elevate. Nella Figura 3.40(b) i dispositivi 1, 2 e 4 usano la priorità 1, mentre i dispositivi 3 e 5 usano la priorità 2.

Se in uno stesso momento ci sono richieste da livelli di priorità diversi l'arbitro concede il bus a un dispositivo di priorità più elevata; fra dispositivi della stessa priorità si utilizza invece la strategia del collegamento a festone. Nella Figura 3.40(b) in caso di conflitti il dispositivo 2 ha priorità sul dispositivo 4, che ce l'ha a sua volta sul 3. Il dispositivo 5 ha invece la priorità più bassa fra tutti, in quanto si trova alla fine del collegamento a festone a priorità inferiore.

Aprendo una parentesi occorre precisare che non è tecnicamente necessario collegare serialmente i dispositivi 1 e 2 alla linea di concessione del bus di livello 2, dato che non possono effettuare delle richieste su tale linea. Tuttavia per ragioni d'implementazione è più facile collegare tutte le linee di concessione a tutti i dispositivi, piuttosto che instaurare dei collegamenti speciali dipendenti dalla priorità associata a ciascun dispositivo.

Alcuni arbitri hanno una terza linea che viene asserita da un dispositivo nel momento in cui accetta una concessione e si impadronisce del bus. Subito dopo aver asserito questa linea di conferma, il dispositivo può negare le linee di richiesta e di concessione. Il risultato è che altri dispositivi possono richiedere il bus mentre il primo lo sta ancora utilizzando. Nel momento in cui termina il trasferimento in corso, il successivo master del bus sarà già stato selezionato. Esso può cominciare subito dopo che la linea di conferma è stata negata, cioè nel momento in cui può iniziare il nuovo turno determinato dall'arbitraggio. Questo schema richiede una linea aggiuntiva nel bus e una maggior quantità di componenti logici in ciascun dispositivo, ma permette un miglior utilizzo dei cicli del bus.

Nei sistemi in cui la memoria è collegata al bus principale, la CPU deve competere con tutti i dispositivi di I/O praticamente a ogni ciclo. Una soluzione molto comune per questa situazione consiste nell'assegnare alla CPU la priorità più bassa in modo che possa usare il bus soltanto quando nessun altro lo vuole. L'idea sottostante è che la CPU può sempre aspettare, mentre molto spesso i dispositivi di I/O sono obbligati ad acquisire il bus molto velocemente, pena la perdita dei dati in arrivo. I dischi che ruotano ad alte velocità, per esempio, non possono aspettare. In molti calcolatori questo problema viene risolto mettendo la memoria su un bus separato rispetto ai dispositivi di I/O, in modo che la CPU non debba competere con questi per l'accesso al bus.

Un'altra soluzione possibile è la decentralizzazione dell'arbitraggio del bus. Un calcolatore potrebbe per esempio avere 16 linee di richiesta, ciascuna con la propria priorità. Quando un dispositivo vuole utilizzare il bus asserisce la linea di richiesta. Tutti i dispositivi monitorano tutte le linee di richiesta in modo che alla fine di ciascun ciclo del bus ognuno di loro può sapere se era il richiedente con priorità più elevata e se quindi ha diritto a utilizzare il bus durante il ciclo successivo. Rispetto al metodo centralizzato questo schema di arbitraggio richiede un maggior numero di linee di bus, ma evita

il potenziale costo dell'arbitro. Un altro limite è che il numero di dispositivi non può superare il numero delle linee di richiesta.

Un altro tipo di arbitraggio decentralizzato del bus, mostrato nella Figura 3.41, utilizza solamente tre linee, indipendentemente da quanti dispositivi sono presenti. La prima linea del bus è una linea OR-cablagata per effettuare le richieste al bus. La seconda linea del bus è chiamata BUSY ed è asserita dal master corrente. La terza è invece utilizzata per arbitrare il bus ed è un collegamento a festone fra tutti i dispositivi; la testa di questa catena viene mantenuta asserita collegandola a un'alimentazione di 5 volt.

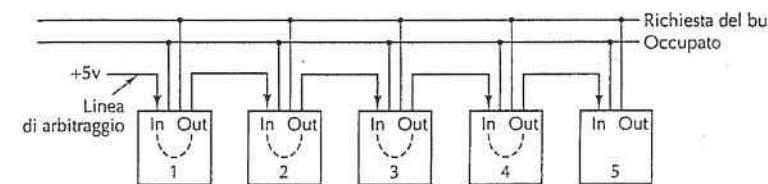


Figura 3.41 Arbitraggio decentralizzato del bus.

Quando nessun dispositivo richiede il bus la linea di arbitraggio, che è asserita, viene propagata attraverso tutti i dispositivi. Per acquisire il bus un dispositivo deve prima controllare se il bus è inattivo e se il segnale dell'arbitraggio che sta ricevendo, IN, è asserito. Se IN è negato, non può diventare il master e nega OUT. Se invece IN è asserito e il dispositivo vuole il bus, allora il dispositivo nega OUT in modo che il dispositivo successivo nel collegamento a festone veda IN negato e neghi anch'esso il proprio OUT. Tutti i dispositivi successivi vedranno pertanto IN negato e negheranno a loro volta OUT. Una volta tornata la calma solo un dispositivo rimarrà con IN asserito e OUT negato; esso diventerà il master del bus, asserirà BUSY e OUT e inizierà il proprio trasferimento.

Con un semplice ragionamento si deduce che fra i dispositivi che richiedono il bus lo ottiene quello che si trova più a sinistra. Questo schema è quindi simile all'originario arbitraggio con collegamento a festone, tranne il fatto che non c'è l'arbitro; questa differenza lo rende più economico, più veloce e non soggetto ai potenziali guasti dell'arbitro.

### 3.4.6 Operazioni del bus

Fino a questo momento abbiamo trattato solamente bus ordinari con un master (generalmente la CPU) che legge da uno slave (di solito la memoria) o scrive su di esso. Nei casi reali sono possibili anche altri tipi di bus che ora analizzeremo.

Di solito viene trasferita una parola alla volta. Tuttavia quando si utilizza la cache è preferibile prelevare in una volta sola un'intera linea di cache (per esempio 8 parole consecutive di 64 bit). Spesso i trasferimenti di blocchi possono essere effettuati in modo più efficiente rispetto a una sequenza di trasferimenti individuali. All'inizio della lettura di un blocco il master del bus comunica allo slave quante parole devono essere

trasferite inserendo, per esempio durante  $T_1$ , il conteggio delle parole sulle linee dei dati. Lo slave, invece di restituire un'unica parola, genera in output a ogni ciclo una nuova parola finché il contatore non si sia esaurito. La Figura 3.42 mostra una versione modificata della Figura 3.38(a) in cui un segnale aggiuntivo **BLOCK** viene asserito per indicare che è stato richiesto il trasferimento di un blocco. In questo esempio una lettura di un blocco di 4 parole richiede 6 cicli invece che 12.

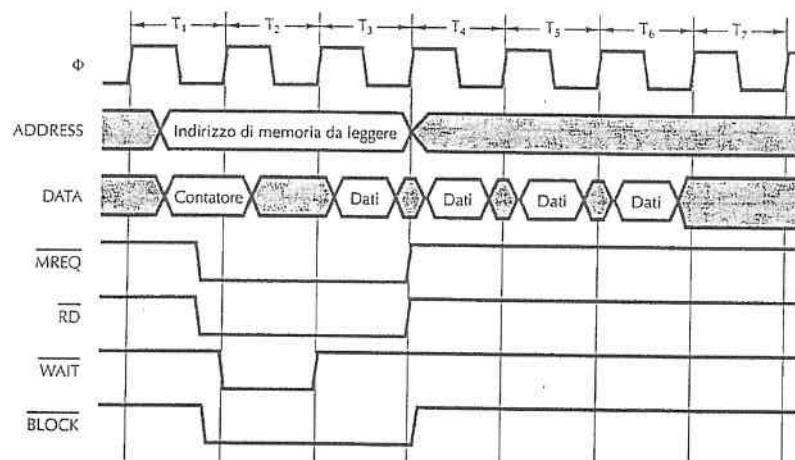


Figura 3.42 Trasferimento di un blocco.

Esistono anche altri tipi di cicli di bus. Per esempio su un sistema multiprocessore con due o più CPU sullo stesso bus è spesso necessario assicurarsi che, in un dato momento, soltanto una CPU utilizzi delle strutture dati critiche della memoria. Una tipica soluzione del problema consiste nell'avere in memoria una variabile che vale 0 quando nessuna CPU sta utilizzando la struttura dati e 1 quando invece è in uso. Se una CPU vuole acquisire l'accesso alla struttura dati deve leggere il valore della variabile e impostarla a 1, se vale 0. Il problema è che, con una buona dose di sfortuna, due CPU potrebbero leggere la variabile durante cicli di bus consecutivi. Se entrambe vedono che la variabile è 0 allora entrambe la imposteranno a 1 e penseranno di essere l'unica CPU che sta utilizzando la struttura dati. Questa sequenza di eventi porta al caos.

Per evitare questa situazione i sistemi multiprocessore hanno spesso uno speciale ciclo di bus leggi-modifica-scrivi che consente a una qualsiasi CPU di leggere una parola dalla memoria, analizzarla e riscriverla in memoria, tutto senza rilasciare il bus. Questo tipo di ciclo evita che altre CPU che intendono utilizzare il bus riescano a impadronirsiene interferendo con l'operazione della prima CPU.

Un altro importante tipo di ciclo di bus serve a gestire gli interrupt. Quando la CPU ordina a un dispositivo di I/O di effettuare qualche operazione si aspetta di solito un interrupt alla fine del lavoro; la segnalazione di questi interrupt richiede l'utilizzo del bus.

Dato che più dispositivi potrebbero voler generare un interrupt simultaneamente, si verificano gli stessi tipi di problemi di arbitraggio visti nel caso dei cicli di bus ordinari. La soluzione più usuale prevede di assegnare priorità ai dispositivi e di utilizzare un arbitro centralizzato per dare priorità al dispositivo per cui la temporizzazione è più critica. Esistono chip controlleri di interrupt di tipo standard che sono utilizzati in modo molto diffuso. Nei PC basati su processori Intel il chipset incorpora il controllore di interrupt 8259A, illustrato nella Figura 3.43.

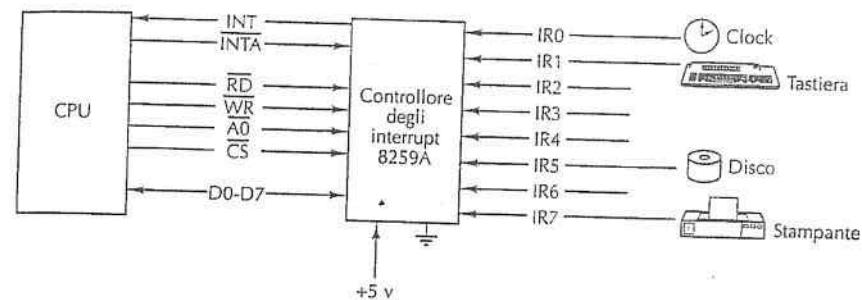


Figura 3.43 Utilizzo del controllore degli interrupt 8259A.

Il chip 8259A ha otto input, chiamati **IRx** (*Interrupt Request*, "richiesta di interrupt"), con i quali si possono collegare direttamente fino a otto controlleri di I/O. Quando uno di questi dispositivi vuole causare un interrupt asserisce la propria linea di input. Quando uno o più input sono asseriti il chip 8259A asserisce **INT** (*INTerrupt*), che pilota direttamente il pin di interrupt sulla CPU. Quando la CPU è in grado di gestire l'interrupt rispedisce all'8259A un impulso su **INTA** (*INTerrupt Acknowledge*, "conferma dell'interrupt"); a quel punto l'8259A deve specificare quale input ha causato l'interrupt generando in output il suo numero sul bus dei dati. Questa operazione richiede un ciclo di bus speciale. L'hardware della CPU utilizza quindi quel numero come indice all'interno di una tabella di puntatori, chiamata vettore di interrupt, per cercare l'indirizzo della procedura da eseguire per poter gestire l'interrupt.

L'8259A ha al suo interno vari registri che la CPU può leggere e scrivere utilizzando i cicli di bus ordinari e i pin **RD** (*Read*), **WR** (*WRite*), **CS** (*Chip Select*) e **A0**. Quando il software ha gestito l'interrupt ed è pronto per accettarne uno nuovo scrive uno speciale codice all'interno di uno dei registri; ciò fa sì che l'8259A neghi **INT**, a meno che non vi sia un altro interrupt pendente. In questi registri si può anche scrivere per portare l'8259A in uno dei vari modi disponibili, mascherando per esempio un insieme di interrupt oppure abilitando altre funzionalità.

Quando sono presenti più di otto dispositivi di I/O è possibile utilizzare in cascata più chip 8259A. Nel caso più estremo tutti gli otto input possono essere connessi agli output di ulteriori otto 8259A; è possibile così gestire fino a 64 dispositivi di I/O in una rete di interrupt a due livelli. Il chip Intel ICH10, uno dei chip del chipset Core i7, incorpora due controlleri di interrupt 8259A. ICH10 dispone dunque di 15 interrupt esterni, uno in meno rispetto ai 16 sui due chip 8259A, perché un interrupt viene utilizzato per connettere in cascata il secondo 8259A al primo. L'8259A possiede alcuni pin dedicati alla gestione di questa cascata, ma qui ci è sembrato opportuno ometterli. Oggi "8259A" è in realtà una parte di un altro chip.

Non abbiamo affatto esaurito il tema della progettazione dei bus; tuttavia il materiale presentato finora dovrebbe fornire sufficienti conoscenze di base per capire qual è in sostanza il funzionamento di un bus e come le CPU e i bus possono interagire. Ci spostiamo ora dal generale allo specifico e diamo uno sguardo ad alcuni esempi di CPU commerciali e ai loro bus.

### 3.5 Esempi di chip della CPU

In questo paragrafo esamineremo a livello hardware, in modo piuttosto dettagliato, i chip Intel Core i7, TI OMAP4430 e Atmel ATmega168.

#### 3.5.1 Intel Core i7

Il Core i7 è un diretto discendente della CPU 8088 utilizzata nel PC IBM originario. Il primo Core i7 fu introdotto nel novembre 2008, come una CPU a 4 processori, con 731 milioni di transistor, una frequenza di 3,2 GHz e una *larghezza di linea* di 45 nanometri. La larghezza di linea corrisponde alla grandezza dei collegamenti tra i transistor (ed è anche una misura della dimensione dei transistor stessi). Più ridotta è la larghezza di linea e più transistor possono essere inseriti sul chip. La legge di Moore riguarda fondamentalmente la capacità di continuare a ridurre le larghezze di linea. Larghezze di linea più piccole permettono velocità di clock più elevate. Per fare un confronto con le misure in gioco si pensi che i capelli hanno un diametro che varia tra 20.000 e i 100.000 nanometri, di cui i più fini sono quelli biondi, mentre quelli con diametro maggiore sono quelli neri.

La versione iniziale del Core i7 era basata sull'architettura "Nahalem", mentre le versioni successive si basano sull'architettura "Sandy Bridge". In questo contesto l'architettura rappresenta l'organizzazione interna della CPU, alla quale spesso è assegnato un nome in codice. I progettisti delle architetture di tanto in tanto si inventano qualche nome stravagante per il loro progetto. Un caso degno di nota è quello delle architetture della serie K di AMD, progettate per rompere l'apparente invulnerabilità di Intel nel mercato dei desktop. Il nome in codice della serie K era "Kryptonite", con evidente riferimento all'unica sostanza in grado di colpire Superman: un colpo intelligente alla dominante Intel.

I Core i7 basati su Sandy Bridge hanno portato il numero di transistor a 1,16 miliardi e lavorano alla frequenza di 3,5 GHz, con larghezza di linea di 32 nanometri. Il

Core i7, pur essendo decisamente più avanzato rispetto all'8088 con 29.000 transistor, è completamente compatibile con esso e può eseguire i suoi programmi senza alcuna modifica (lo stesso vale, ovviamente, anche per i programmi di tutti i processori intermedi).

Dal punto di vista del software il Core i7 è una macchina interamente a 64 bit. A livello utente ha tutte le funzionalità ISA dei modelli 80386, 80486, Pentium, Pentium II, Pentium Pro, Pentium III e Pentium 4, compresi gli stessi registri, le stesse istruzioni e la stessa implementazione dello standard in virgola mobile IEEE 754 interamente sul chip, inoltre ha alcune nuove istruzioni dedicate alla crittografia.

Il processore Core i7 è una CPU multicore, ovvero lo stampo di silicio contiene più processori. La CPU è venduta con un numero variabile di processori, per ora dai 2 ai 6, ma nei piani futuri questo numero crescerà. Quando un programmatore scrive un programma parallelo, utilizzando thread e lock, riesce ad avere significativi guadagni nella velocità d'esecuzione sfruttando il parallelismo dei processori multipli. In aggiunta, le singole CPU sono "hyperthreaded", cioè possono essere attivi simultaneamente diversi thread hardware sulla stessa CPU. L'*hyperthreading* (anche chiamato "multithreading simultaneo" dai progettisti) permette di tollerare latenze molto brevi, per esempio un fallimento d'accesso alla cache, con l'attivazione dei thread hardware. Al contrario il threading software può gestire soltanto latenze molto lunghe, per esempio un page fault, a causa dei numerosi cicli necessari per implementare il passaggio di thread via software.

Internamente, a livello di microarchitettura, il Core i7 è un progetto dalle molte potenzialità. Basato sulle architetture dei suoi predecessori, il Core 2 e il Core 2 Duo, il Core i7 può eseguire fino a 4 istruzioni per volta, il che fa di questo processore una macchina superscalare a 4 livelli. Esamineremo questa architettura nel Capitolo 4.

Il Core i7 ha 3 livelli di cache. Ogni processore di un Core i7 ha una cache dati di livello 1 (L1) da 32 KB, una cache istruzioni di livello 1 da 32 KB e una cache di livello 2 (L2) da 256 KB. La cache di secondo livello è unificata, ovvero contiene un mix di dati e istruzioni. Tutti i core condividono una cache unificata di livello 3 (L3) la cui dimensione varia dai 4 ai 15 MB a seconda del modello di processore. La presenza di 3 livelli di cache migliora significativamente le prestazioni del processore, ma ha un grande costo in termini di silicio, visto che il Core i7 può arrivare ad avere un totale di 17 MB di cache su un singolo stampo di silicio.

Siccome ogni chip Core i7 contiene più processori con una cache dati privata, sorgono dei problemi quando un processore modifica nella sua cache un dato presente anche nella cache di un altro processore. Se quest'ultimo prova a leggere quel dato dalla (sua) memoria otterrà un valore non aggiornato. Per garantire la coerenza della memoria in un multiprocessore ciascuna CPU effettua uno snooping ("ficcanso") sul bus della memoria per cercare eventuali riferimenti alle parole che ha memorizzato nella propria cache. Quando vede un tale riferimento si intromette nel bus fornendo il dato richiesto prima che lo possa fare la memoria. Nel Capitolo 8 studieremo lo snooping.

Nei sistemi Core i7 si utilizzano due principali bus esterni, entrambi sincroni: un bus di memoria DDR3 per l'accesso alla memoria centrale DRAM e un bus PCI Express per connettere il processore ai dispositivi di I/O. Le versioni di alta gamma del Core i7 hanno diversi bus di memoria e diversi bus PCI Express. Dispongono inoltre di una

porta QPI (*Quick Path Interconnect*) che connette il processore a un'interconnessione esterna al multiprocessore e permette di costruire sistemi con più di sei processori. La porta QPI invia e riceve richieste di coerenza della cache, oltre a vari altri messaggi per la gestione del multiprocessore, come gli interrupt interprocessore.

Un problema del Core i7, come di tutti gli altri processori moderni desktop, è il consumo energetico e il calore generato. Per prevenire eventuali danni al silicio il calore deve essere rimosso dal chip non appena prodotto. Il Core i7 consuma tra i 17 e i 150 watt, a seconda della frequenza e del modello. Pertanto Intel è costantemente alla ricerca di soluzioni per dissipare il calore prodotto dai suoi chip. Le tecnologie di raffreddamento e la conduzione termica del supporto sono vitali per proteggere il silicio.

Il Core i7 è realizzato in un supporto LGA quadrato di 37,5 mm di lato. Contiene 1.155 pad posti sul fondo, di cui 286 sono per l'alimentazione e 360 sono messi a terra per ridurre il rumore. I pad sono disposti più o meno come un quadrato  $40 \times 40$  mm con la parte centrale di  $17 \times 25$  mm mancante. Inoltre, mancano 20 pad sul perimetro, con una distribuzione asimmetrica, per evitare che il chip venga inserito in maniera scorretta nel suo zoccolo. La Figura 3.44 mostra la disposizione fisica dei pin.

Il chip è attrezzato in modo che sopra di esso sia possibile montare un dissipatore di calore e una ventola per raffreddarlo. Per avere un'idea dell'entità del problema energetico basta accendere una lampadina da 150 Watt, aspettare qualche minuto e sfiorarla (senza però toccarla). La stessa quantità di calore deve essere continuamente dissipata da un processore Core i7 di fascia alta. Quando il Core i7 smette di essere utile come CPU, lo si può quindi sempre riciclare come fornello da campeggio!

Secondo le leggi della fisica ogni fenomeno che produce una gran quantità di calore deve consumare molta energia. In un calcolatore portatile non è desiderabile utilizzare una gran quantità di energia in quanto la carica della batteria è limitata e tenderebbe a consumarsi molto rapidamente. Per risolvere questo problema Intel ha dotato le CPU di un modo per "riposare" quando sono inattive e di entrare in un "sonno profondo" quando è probabile che resteranno inattive per un certo periodo di tempo. Ci sono cinque stati possibili, che variano dalla piena esecuzione al sonno profondo. Negli stati intermedi alcune funzionalità (come lo snooping della cache e la gestione degli interrupt) sono disabilitate, mentre altre continuano a essere utilizzabili. Quando la CPU si trova nello stato di sonno profondo i registri sono preservati, mentre le cache vengono scaricate e spente. Per risvegliarla la CPU è necessario un segnale hardware. Non si sa se il Core i7 sia in grado di sognare quando dorme profondamente.

#### Disposizione logica dei contatti del Core i7

Dei 1155 pin del Core i7, ne vengono utilizzati 447 per i segnali, 286 per l'alimentazione (a diverse tensioni) e 360 per la terra, mentre gli altri 62 sono riservati a utilizzi futuri. Dato che alcuni segnali logici utilizzano due o più pin (come gli indirizzi di memoria richiesti), il numero di segnali distinti è solamente 131. La Figura 3.45 mostra una versione piuttosto semplificata della disposizione logica dei contatti. Sul lato sinistro della figura ci sono cinque gruppi principali di segnali per il bus di memoria, sul lato destro altri segnali di varia natura.

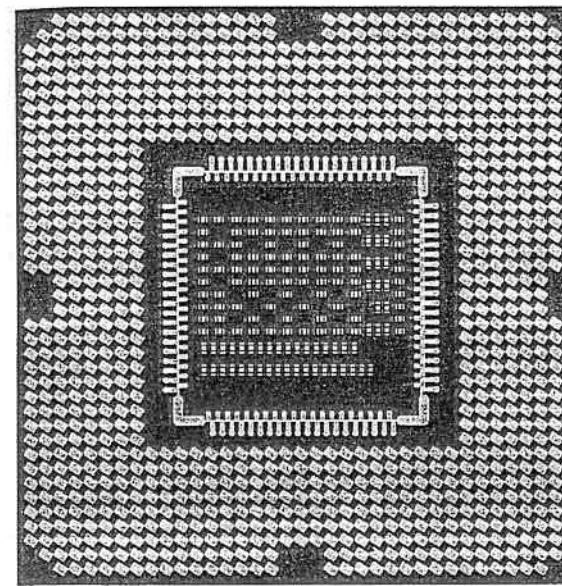


Figura 3.44 Disposizione fisica dei contatti del Core i7.

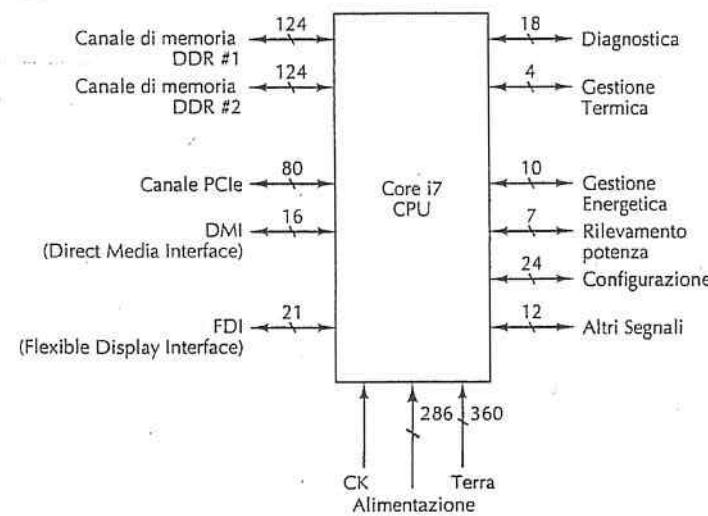


Figura 3.45 Disposizione logica dei contatti del Core i7.

Esaminiamo i segnali a partire da quelli del bus. I primi due segnali del bus sono utilizzati come interfacce verso le DRAM compatibili DDR3. Questo gruppo di segnali fornisce ai banchi DRAM l'indirizzo, i dati, il controllo e il clock. Il Core i7 supporta due canali DRAM DDR3 indipendenti che lavorano su un bus con frequenza di clock di 666 MHz in maniera bidirezionale e permettono 1333 transazioni al secondo. L'interfaccia DDR3 ha un'ampiezza di 64 bit, dunque le due interfacce DDR3 lavorando in tandem possono offrire ai programmi assetati di memoria fino a 20 GB di dati al secondo.

Il terzo gruppo di bus è costituito dall'interfaccia PCI Express ed è utilizzata per connettere le periferiche direttamente alla CPU. PCI Express è un'interfaccia seriale ad alta velocità, in cui ogni singolo collegamento seriale forma una corsia (lane) verso le periferiche. Il collegamento del Core i7 è un'interfaccia x16, il che significa che può gestire 16 corsie simultaneamente per arrivare così a una larghezza di banda aggregata di 16 GB/sec. Nonostante PCI Express sia un canale seriale, sui suoi collegamenti viaggia un ricco insieme di comandi, tra cui i read, i write e gli interrupt dei dispositivi e i comandi di configurazione.

Un'altra categoria di bus è formata dal DMI (*Direct Media Interface*), utilizzato per il collegamento della CPU al suo *chipset*. L'interfaccia DMI è simile a PCI Express, anche se lavora a una velocità dimezzata, dato che le sue 4 corsie possono offrire solamente una velocità di trasferimento dati di 2,5 GB/sec. Il chipset di una CPU contiene un ricco insieme di supporti per le interfacce di periferiche aggiuntive, richiesti tipicamente da sistemi di alto livello dotati di molte periferiche di I/O. Il chipset Core i7 utilizza i chip P67 e ICH10. Il P67 è il coltellino svizzero dei chip: offre infatti le interfacce SATA, USB, Audio, PCIe e Flash. Il chip ICH10 fornisce il supporto per interfacce più datate, tra cui l'interfaccia PCI e le funzionalità di controllo degli interrupt del chip 8259A. In aggiunta, ICH10 contiene vari altri circuiti, come controllori di clock real time, timer di eventi e DMA (*Direct Memory Access*). Grazie a chip come questi la costruzione di un PC completo risulta notevolmente semplificata.

È possibile configurare il Core i7 in modo che utilizzi gli interrupt allo stesso modo dell'8088 (per la retrocompatibilità) oppure secondo una nuova modalità che impiega un dispositivo chiamato APIC (*Advanced Programmable Interrupt Controller*).

Il Core i7 può funzionare a varie tensioni, ma deve sapere quale di queste viene utilizzata. I segnali per la gestione dell'alimentazione sono utilizzati per la selezione automatica della tensione, per comunicare alla CPU che la tensione è stabile e per altri aspetti riguardanti l'alimentazione. Sempre mediante questi segnali vengono gestiti i vari stati di riposo, dato che la loro scelta è fatta allo scopo di ridurre il consumo energetico.

Il Core i7, nonostante una gestione sofisticata dell'alimentazione, si riscalda notevolmente. Ogni chip Core i7 contiene diversi sensori di temperatura per la protezione dei circuiti in silicio, in grado di rilevare situazioni di surriscaldamento. Il gruppo monitoraggio termico si occupa della gestione termica e permette alla CPU di indicare alle sue componenti le situazioni di rischio di surriscaldamento. La CPU asserisce uno dei suoi pin quando la sua temperatura interna supera i 130° C (266° F). Se la CPU raggiunge questa temperatura, probabilmente inizia a sognare il pensionamento e una sua riconversione in fornello da campeggio!

Anche a queste temperature l'utente non si deve preoccupare della sicurezza del Core i7. Quando i sensori interni rilevano una situazione di possibile surriscaldamento viene avviata la limitazione termica (*thermal throttling*), una tecnica in grado di ridurre rapidamente il calore generato facendo in modo che la CPU lavori soltanto ogni N cicli di clock. Più grande è il valore di N, più il funzionamento del processore verrà limitato e più velocemente si raffredderà. Senza l'ideazione della limitazione termica le CPU, in caso di guasti al sistema di raffreddamento, sarebbero bruciate. Le prove di questi tempi bui della gestione termica delle CPU si possono recuperare cercando «exploding CPU» su YouTube. Questi video sono spesso fasulli, ma il problema è reale.

Il segnale Clock fornisce al processore il clock di sistema, usato internamente per generare una varietà di clock basati su suoi multipli o sue frazioni. È proprio così! È possibile generare anche dei multipli della frequenza di clock di sistema grazie a un dispositivo molto intelligente chiamato DLL (*Delay-Locked Loop*).

Il gruppo per la diagnostica contiene segnali per sistemi di test e debugging secondo lo standard IEEE 1149.1 dei test JTAG (*Joint Test Action Group*). L'ultimo gruppo è un calderone di vari altri segnali dedicati a utilizzi particolari.

#### Pipeline sul bus di memoria DDR3 del Core i7

Le moderne CPU come il Core i7 pretendono molto dalle memorie DRAM. I singoli processori possono produrre richieste di accesso più velocemente di quanto una lenta DRAM possa fornire i valori, e questo problema viene amplificato quando diversi processori effettuano richieste simultanee. Per evitare che la CPU rimanga in attesa a causa della mancanza di dati è fondamentale ottenere dalla memoria il massimo throughput possibile. Per questa ragione il bus di memoria DDR3 del Core i7 è strutturato a pipeline, in modo che su di esso possano avvenire fino a quattro transazioni contemporanee. Il concetto di pipeline è stato visto nel contesto delle CPU (Figura 2.4) nel corso del Capitolo 2, e anche le memorie possono avere un'architettura a pipeline.

Per consentire l'uso della pipeline, le richieste di memoria del Core i7 sono composte da tre fasi.

- Fase di ACTIVATE della memoria, che "apre" una riga della DRAM per prepararla a successivi accessi.
- Fase di READ o WRITE della memoria, in cui si possono effettuare accessi multipli a singole parole appartenenti alla riga correntemente aperta della DRAM, oppure accessi multipli a una sequenza di parole appartenenti alla riga corrente della DRAM con l'utilizzo del burst mode.
- La fase di PRECHARGE, che "chiude" la riga corrente della DRAM e prepara la memoria per il prossimo comando ACTIVATE.

Il segreto dell'accesso a memoria del Core i7 sta nell'organizzazione strutturata in più *banchi* della DRAM DDR3 sul suo chip. Un banco è un blocco di memoria DRAM a cui si può accedere in parallelo con altri banchi di memoria, anche se questi sono contenuti sullo stesso chip. Una DRAM DDR3 è tipicamente formata da 8 banchi di DRAM. Tuttavia, le specifiche dell'interfaccia DDR3 permettono fino a quattro accessi concorrenti soltanto su un singolo canale DDR3. Nel diagramma temporale della Figura

3.46 è mostrato come il Core i7 effettua 4 accessi di memoria a 3 distinti banchi DRAM. Gli accessi sono completamente sovrapposti, in modo che le letture sulla DRAM avvengono in parallelo sullo stesso chip. La figura mostra quali comandi portano a operazioni successive mediante l'uso di frecce nel diagramma temporale.

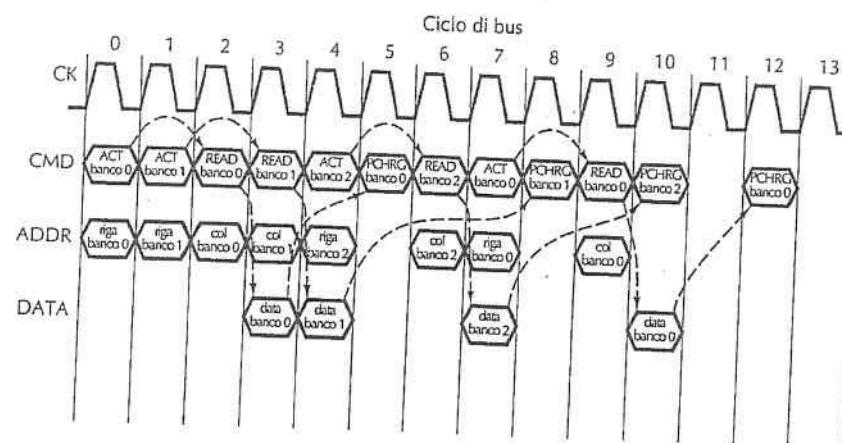


Figura 3.46 Richieste di memoria sull'interfaccia DDR3 del Core i7 gestite con pipeline.

Come mostrato nella figura 3.46, l'interfaccia di memoria DDR3 dispone di quattro segnali primari: il *clock* del bus (CK), il segnale di *comando* del bus (CMD), il segnale di *indirizzo* (ADDR) e quello di *dato* (DATA). Il segnale CK dirige tutte le attività del bus. Il segnale CMD indica quale attività si richiede alla DRAM connessa. Il comando ACTIVATE specifica l'indirizzo della riga DRAM da aprire per mezzo del segnale ADDR. Per eseguire una READ, viene fornito l'indirizzo della colonna DRAM per mezzo del segnale ADDR e la DRAM, dopo un intervallo di tempo fissato, mette il valore letto sul segnale DATA. Alla fine il comando PRECHARGE indica il banco da precaricare per mezzo del segnale ADDR. Ai fini di questo esempio, il comando ACTIVATE deve precedere il primo READ allo stesso banco di due cicli di bus DDR3 e i dati sono prodotti un ciclo di bus dopo il comando READ. Inoltre, l'operazione di PRECHARGE deve avvenire almeno due cicli di bus dopo l'ultima operazione di read allo stesso banco DRAM.

Il parallelismo nelle richieste alla memoria può essere osservato dalle richieste READ a distinti banchi di DRAM. I primi due accessi READ ai banchi 0 e 1 sono completamente sovrapposti e producono i loro risultati nei cicli di clock 3 e 4 rispettivamente. L'accesso al banco 2 si sovrappone parzialmente al primo accesso al banco 1 e, infine, la seconda READ al banco 0 si sovrappone parzialmente all'accesso al banco 2.

Ci si potrà chiedere come fa il Core i7 a sapere quando riceverà una risposta alle sue richieste di READ, e quando potrà effettuare nuove richieste. La risposta è che sa quan-

do riceve e quando iniziare le richieste perché possiede un modello completo delle attività interne di ogni DRAM DDR3 collegata. È quindi in grado di anticipare la restituzione dei dati nel ciclo corretto e saprà di dover evitare l'inizio di un'operazione PRECHARGE prima che siano passati due cicli dall'ultima READ. Il Core i7 può anticipare tutte queste attività perché l'interfaccia di memoria DDR3 è un'interfaccia sincrona di memoria. Ogni attività impiega quindi un ben noto numero di cicli di bus DDR3. Anche con tutte queste conoscenze, la costruzione di un'interfaccia DDR3 di alte prestazioni e completamente gestita a pipeline non è un compito banale, perché è richiesto l'utilizzo di diversi temporizzatori interni e di rilevatori di conflitti per implementare in maniera efficiente la gestione delle richieste di memoria.

### 3.5.2 Texas Instruments OMAP4430

Come secondo esempio di CPU esaminiamo adesso il SoC (*System-on-a-Chip*) Texas Instruments (TI) OMAP4430. Questa CPU implementa il set di istruzioni ARM ed è rivolto ad applicazioni mobili ed embedded come smartphone, tablet e apparati Internet. Giustamente chiamato SoC (*system-on-a-chip*), incorpora una vasta gamma di dispositivi in modo tale da implementare uno strumento completo di elaborazione in combinazione con le periferiche fisiche (touchscreen, memoria flash e così via).

Il SoC OMAP4430 include due Core ARM A9, degli acceleratori aggiuntivi e una vasta gamma di interfacce per periferiche. L'organizzazione interna dell'OMAP4430 è mostrata nella Figura 3.47. I Core ARM A9 sono microarchitetture superscalari a 2 livelli. In aggiunta, sul chip OMAP4430, ci sono tre acceleratori: un processore grafico PowerVR SGX540, un processore di segnale di immagine (ISP) e un processore video IVA3. Il processore SGX540 fornisce un efficiente rendering 3D programmabile, come le GPU che si trovano nei PC desktop, anche se è più piccolo e più lento. L'ISP è un processore programmabile per la manipolazione efficiente di immagini, destinato alla tipologia di operazioni che necessarie in una fotocamera digitale di fascia alta. L'IVA3 implementa un'efficiente codifica e decodifica video, con prestazioni sufficienti per supportare applicazioni 3D come quelle delle console di gioco portatili. Inoltre, nel SoC OMAP4430 si trova una vasta gamma di interfacce per periferiche, tra cui un controllore di touchscreen, un controllore di tastiera e le interfacce DRAM, Flash, USB e HDMI. Texas Instruments ha dettagliato la tabella di marcia per la serie di CPU OMAP. I progetti futuri avranno qualcosa in più di tutto: più core ARM, più GPU e più periferiche.

Il SoC OMAP4430 è stato introdotto nei primi mesi del 2011 con due core ARM A9 a 1 GHz e con l'utilizzo di una tecnologia a 45 nanometri. Un aspetto fondamentale della sua progettazione è che esegue gran quantità di calcoli con un consumo molto basso, proprio perché è rivolto a dispositivi mobili alimentati a batteria nei quali più efficiente è il progetto, più a lungo l'utente può aspettare prima di ricaricare la batteria.

I molti processori incorporati nell'OMAP4430 servono in maniera specifica per sostenere la sua missione di funzionamento a bassa potenza. Il processore grafico, ISP, e IVA3 sono acceleratori programmabili che forniscono buone capacità di calcolo con un minor consumo di energia rispetto a quella richiesta dalle stesse attività in esecuzione sulla CPU ARM A9. Completamente alimentato, il SoC OMAP4430 consuma solo 600 mW di potenza. Rispetto a un Core i7 di fascia alta, i chip OMAP4430 utilizzano

al massimo 1/250 della potenza. L'OMAP4430 implementa anche una modalità di sospensione molto efficace; quando tutti i componenti sono inattivi, il chip consuma solamente 100 µW. Una sospensione efficiente è fondamentale per applicazioni mobili con lunghi periodi di standby, come per esempio i telefoni cellulari. Minore è l'energia utilizzata in modalità di sospensione, più a lungo il cellulare può rimanere in standby.

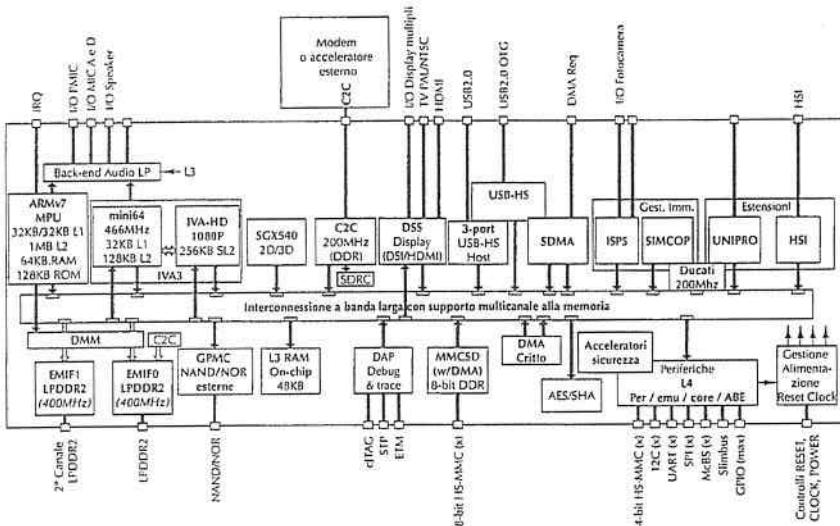


Figura 3.47 Organizzazione interna del SoC OMAP4430

Per ridurre ulteriormente le richieste energetiche dell'OMAP4430, il progetto incorpora varie funzionalità di gestione dell'energia, tra cui una *scala dinamica di tensione* e il *power gating*. La scala dinamica di tensione consente ai componenti di essere in esecuzione più lentamente a una tensione inferiore, con conseguente riduzione significativa dei requisiti di potenza. Se per i calcoli non è necessaria la massima velocità della CPU, la tensione può essere abbassata per rendere più lenta la CPU e risparmiare così più energia. Il *power gating* è uno strumento di gestione energetica ancora più aggressivo che permette di togliere completamente l'alimentazione ai componenti che non sono in uso, eliminandone in tal modo l'assorbimento di potenza. Per esempio, in un'applicazione per tablet, se l'utente non sta guardando un film il processore video IVA3 viene completamente spento. Al contrario, quando l'utente guarda un film, l'IVA3 funziona al massimo delle sue potenzialità per effettuare le operazioni di decodifica video, mentre le due CPU ARM A9 possono rimanere in sospensione.

Nonostante la loro vocazione al lavoro con misere quantità di joule, i core ARM A9 godono di una microarchitettura molto capace. Sono in grado di decodificare ed eseguire fino a 2 istruzioni per ogni ciclo. Come avremo modo di imparare nel Capitolo 4,

questo valore rappresenta il massimo throughput della microarchitettura. Non ci si aspetti però di poter eseguire questo numero di istruzioni in ogni ciclo. Piuttosto, si pensi a questo valore come al massimo delle prestazioni garantite dal produttore, un livello che il processore non supererà mai. In molti cicli verranno eseguite meno di due istruzioni a causa della miriade di "ostacoli" che possono mettere le istruzioni in una condizione di stallo, con conseguente riduzione del throughput. Per far fronte a molte di queste limitazioni del throughput, il core ARM A9 incorpora un potente predittore di salti (*branch predictor*), un'esecuzione fuori sequenza delle istruzioni e un sistema di memoria altamente ottimizzato.

Il sistema di memoria OMAP4430 ha due cache interne principali L1 per ogni processore ARM A9: una di 32 KB per le istruzioni e una di 32 KB per i dati. Come il Core i7, OMAP4430 dispone anche di una cache di livello 2 sul chip, ma a differenza del Core i7 questa cache ha dimensioni relativamente piccole (1 MB) ed è condivisa da entrambi i core ARM A9. Le cache sono alimentate attraverso due canali DRAM LPDDR2 a bassa potenza. L'interfaccia di memoria LPDDR2 è derivata dall'interfaccia standard DDR2, modificata per richiedere un minor numero di fili e operare a tensioni che offrano maggiore efficienza energetica. Inoltre, il controllore della memoria incorpora varie ottimizzazioni per l'accesso alla memoria, come il *tiled prefetching* e il supporto alla *in-memory rotation*.

Anche se nel corso del Capitolo 4 tratterremo le cache in maggior dettaglio è utile spendere ora alcune parole al riguardo. Tutta la memoria centrale è suddivisa in linee di cache (blocchi) di 32 byte. All'interno della cache di primo livello sono mantenute le 1024 linee d'istruzioni e le 1024 linee di dati maggiormente utilizzate. Le linee di cache usate frequentemente, ma che non trovano spazio nella cache di primo livello, vengono memorizzate in quella di secondo livello. Questa cache contiene dati e istruzioni provenienti dalle due CPU ARM A9, mischiati fra loro senza alcun ordine particolare. La cache di livello 2 contiene le 32.768 linee della memoria principale utilizzate più recentemente.

Quando si verifica un fallimento (miss) nella cache di primo livello la CPU spedisce l'identificatore della linea che sta cercando (tag dell'indirizzo) alla cache di secondo livello. La risposta (tag dei dati) comunica alla CPU se la linea si trova all'interno della cache di secondo livello e, in tal caso, in quale stato si trova. Se la linea è memorizzata nella cache la CPU può ottenerla. Per estrarre un valore dalla cache di secondo livello sono necessari 19 cicli. Un simile tempo di attesa è piuttosto lungo, per questo motivo i programmati intelligenti cercano di ottimizzare i loro programmi perché utilizzino meno dati, rendendo più probabile la presenza dei dati nella più veloce cache di livello 1.

Se la linea di cache non si trova neanche nella cache di secondo livello, occorre prelevarla dalla memoria centrale attraverso l'interfaccia LPDDR2. Questa interfaccia dell'OMAP4430 è incorporata sul chip per permettere la connessione diretta tra la DRAM e l'OMAP4430. Per accedere alla memoria, la CPU deve prima di tutto inviare la parte alta dell'indirizzo di memoria al chip DRAM utilizzando le 13 linee di indirizzo. Questa operazione, chiamata ACTIVATE, carica un'intera riga di memoria della DRAM in un buffer di riga. In seguito la CPU può emettere più comandi READ o WRITE inviando la parte restante dell'indirizzo sulle stesse 13 linee e inviando (o ricevendo) il dato sulle 32 linee di dati.

Mentre attende i risultati, la CPU può continuare a svolgere altri compiti. Per esempio, un fallimento di accesso alla cache per effettuare il prefetching di un'istruzione non impedisce alla CPU di proseguire l'esecuzione di una o più istruzioni già prelevate, ognuna delle quali può far riferimento a dati non presenti in nessuna cache. Sulle due interfacce LPDDR2 possono dunque essere effettuate più operazioni alla volta, anche da parte dello stesso processore. Tocca al controllore della memoria tener traccia di tutte queste operazioni e soddisfare le richieste di memoria nell'ordine più efficiente.

Quando finalmente i dati sono pronti, dalla memoria arrivano 4 byte alla volta. Un'operazione di memoria può utilizzare la lettura o la scrittura in una modalità detta *burst mode* che permette di leggere o scrivere diversi indirizzi contigui all'interno della stessa riga DRAM. Questa modalità è particolarmente efficiente per la lettura o scrittura di blocchi di cache. Notiamo qui che la descrizione del chip OMAP4430 appena fornita, come quella del Core i7 data in precedenza, è stata notevolmente semplificata.

Il chip OMAP4430 è distribuito in un supporto PBGA (*ball grid array*) con 547 pin (Figura 3.48). I PBGA sono simili agli LGA, salvo avere i collegamenti a forma di piccole sfere di metallo piuttosto che di pad quadrati. I due supporti non sono compatibili: ecco un'altra prova del fatto che non si può infilare un piolo quadrato in un buco tondo. Il supporto dell'OMAP4430 consiste in una matrice rettangolare di 28x26 sfere, con due anelli di sfere mancanti all'interno e due mezze righe e mezze colonne vuote disposte in maniera asimmetrica, utili a evitare l'inserimento non corretto del chip sul socket BGA.

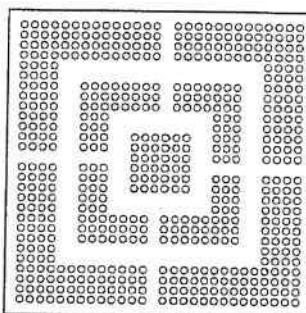


Figura 3.48 Disposizione fisica dei contatti del SoC OMAP4430.

È difficile confrontare chip CISC (come il Core i7) e chip RISC (come l'OMAP4430) basandosi soltanto sulla velocità di clock. Per esempio, i due core AMD A9 hanno una velocità di esecuzione di picco di quattro istruzioni per ciclo di clock, più o meno la stessa velocità di esecuzione del processore superscalare a 4 livelli Core i7. Il Core i7 raggiunge tuttavia una velocità di esecuzione più alta, perché dispone di fino a 6 processori che operano a una frequenza di clock di 3,5 volte più alta (3,5 GHz) rispetto all'OMAP4430. L'OMAP4430 può sembrare una tartaruga in confronto alla lepre Core i7, ma la tartaruga consuma molta meno energia e in qualche caso può arrivare prima, in particolare quando la capacità della batteria della lepre è ridotta.

### 3.5.3 Il microcontrollore Atmel ATmega168

Il Core i7 e l'OMAP4430 sono piattaforme ad alte prestazioni progettate per dispositivi con grandi capacità di calcolo. Il primo è destinato principalmente alle applicazioni desktop, e il secondo alle applicazioni mobili. Quando si pensa ai calcolatori si ha in mente questo tipo di sistemi. C'è però un altro intero mondo che è effettivamente ancora più vasto: quello dei sistemi integrati. In questo paragrafo daremo un rapido sguardo a questa realtà.

Forse è un po' esagerato affermare che ogni dispositivo elettronico dal costo superiore ai 100 \$ contiene al suo interno un calcolatore. Tuttavia al giorno d'oggi televisori, telefoni cellulari, agende elettroniche, fornaci a microonde, videocamere digitali, videoregistratori, stampanti laser, allarmi antifurto, protesi acustiche, giochi elettronici e tanti altri dispositivi sono controllati mediante calcolatori. Di solito si tende a rendere più economici i calcolatori inseriti all'interno di questi oggetti piuttosto che dotarli di elevate prestazioni; ciò porta a compromessi diversi rispetto a quelli che si compiono per le CPU ad alte prestazioni studiate finora.

Come abbiamo detto nel Capitolo 1, il microcontrollore Atmel ATmega168 è molto diffuso principalmente grazie al suo basso costo (circa 1\$). In più, come vedremo tra poco, è un chip particolarmente versatile che rende il collegamento agevole e poco costoso. Esaminiamo allora questo chip, la cui disposizione fisica dei contatti è mostrata nella Figura 3.49.

PC6	1	28	PC5
PD0	2	27	PC4
PD1	3	26	PC3
PD2	4	25	PC2
PD3	5	24	PC1
PD4	6	23	PC0
VCC	7	22	GND
GND	8	21	AREF
PB6	9	20	AVCC
PB7	10	19	PB5
PD5	11	18	PB4
PD6	12	17	PB3
PD7	13	16	PB2
PB0	14	15	PB1

Figura 3.49 Disposizione fisica dei contatti di Atmel ATmega168.

L'usuale supporto dell'ATmega168 ha 28 pin (ma ne esistono altre varianti). Si nota probabilmente a prima vista che la disposizione dei contatti del chip è piuttosto strana rispetto agli esempi analizzati in precedenza. In particolare, il chip non è dotato di linee indirizzo né di linee dati. Ciò è dovuto al fatto che il chip non è stato progettato per essere connesso a una memoria, ma soltanto a dispositivi. Tutta la memoria (SRAM o Flash) è contenuta all'interno del processore, eliminando così il bisogno di avere pin per i dati e gli indirizzi, come mostrato nella figura 3.50.

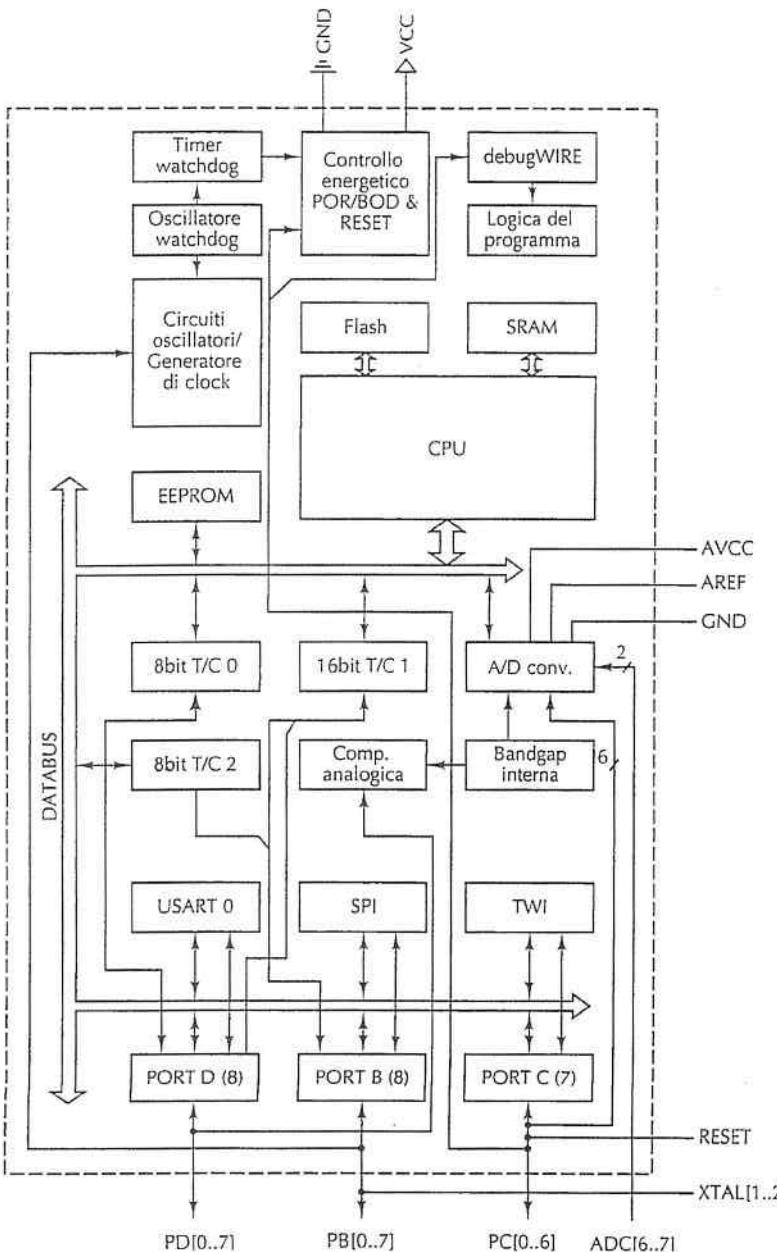


Figura 3.50 Architettura interna e disposizione logica dei contatti dell'ATmega168.

Al posto di tali pin, l'ATmega168 ha 27 porte di I/O, 8 linee nelle porte B e D e 7 linee nella porta C. Queste linee sono progettate per essere collegate alle periferiche di I/O e ognuna può essere configurata internamente dal software di avvio per essere utilizzata come input o come output. Per esempio, in un forno a microonde una linea digitale configurata in ingresso può essere il sensore di "forno aperto" e una linea in uscita può controllare l'accensione e spegnimento del generatore di microonde. Il software dell'ATmega168 controlla la chiusura del forno prima di accendere il generatore. Se il forno viene aperto all'improvviso, il software toglie l'alimentazione. Nella pratica c'è anche un sistema di blocco hardware.

Eventualmente, sei segnali in ingresso alla porta C possono essere configurati come I/O analogici. I pin analogici possono leggere il livello di tensione in ingresso o impostare il livello di tensione in uscita. Estendiamo il nostro esempio considerando un forno dotato di sensore di temperatura per riscaldare i cibi alla temperatura desiderata. Il sensore di temperatura viene connesso in ingresso alla porta C, così che il software possa leggere la tensione del sensore e trasformare questo valore in una temperatura mediante una funzione di traduzione specifica per il sensore. Gli altri pin dell'ATmega168 sono l'alimentazione (VCC), due pin messi a terra (GND) e due pin per configurare la circuiteria analogica di I/O (AREF, AVCC).

L'architettura interna dell'ATmega168 è, come nel caso di OMAP4430, un SoC con memoria e una vasta gamma di dispositivi interni. L'ATmega168 è distribuito con 16KB di memoria flash interna per la memorizzazione di informazioni che variano raramente (come le istruzioni del programma), 1KB di EEPROM e una memoria non volatile che può essere scritta dal software e nella quale vengono memorizzate le informazioni di configurazione di sistema. Nell'esempio del microonde, la EEPROM potrebbe memorizzare un bit per indicare se l'ora deve essere visualizzata in formato 12 o 24 ore. L'ATmega168 può incorporare anche 1KB di SRAM, dove il software può salvare temporaneamente le variabili.

Il processore interno esegue le 131 istruzioni AVR, ognuna lunga 16 bit. Si tratta di un processore a 8 bit, che opera su dati di 8 bit e ha registri interni da 8 bit. L'insieme di istruzioni include istruzioni speciali per permettere al processore di operare efficientemente su dati più grandi di 8 bit. Per esempio, per eseguire un'addizione su dati da 16 bit, o più grandi, il processore offre l'istruzione "somma con riporto" che somma due valori e aggiunge il riporto dell'addizione precedente. Tra i restanti componenti interni c'è un orologio in tempo reale e altre interfacce, tra cui il supporto per collegamenti seriali, i collegamenti PWM (pulse-width-modulated), il collegamento I2C (Inter-IC bus) e i controllori di I/O digitale e analogico.

### 3.6 Esempi di bus

I bus sono il collante che tiene insieme i sistemi di calcolo. In questo paragrafo daremo uno sguardo dettagliato ad alcuni modelli molto diffusi: il bus PCI e l'Universal Serial Bus. Il bus PCI è il bus di I/O delle periferiche principalmente utilizzato oggi sui PC. Ne esistono di due tipi: il vecchio bus PCI e il nuovo e più veloce PCI Express (PCIe). L'Universal Serial Bus è un bus per periferiche a bassa velocità, come mouse e tastiere,

che sta guadagnando sempre più popolarità. La seconda e la terza versione del bus USB funzionano a velocità molto più elevate. Nei paragrafi successivi analizzeremo questi bus per scoprire come funzionano.

### 3.6.1 Bus PCI

Sul PC IBM originario la maggior parte delle applicazioni era di tipo testuale. Gradualmente, con l'introduzione di Windows, presero piede le interfacce grafiche per l'utente. Nessuna di queste applicazioni richiedeva a vecchi bus di sistema come il bus ISA un eccessivo sforzo. Tuttavia la situazione cambiò radicalmente nel momento in cui molte applicazioni, in particolare i giochi multimediali, cominciarono a utilizzare il calcolatore per visualizzare immagini a pieno schermo e in movimento.

Facciamo un semplice calcolo. Consideriamo uno schermo con la risoluzione di  $1024 \times 768$  con 3 byte per pixel: ogni schermata contiene 2,25 MB di dati. Per ottenere un movimento senza scatti sono necessarie almeno 30 schermate al secondo per un tasso di trasferimenti dati complessivo di 67,5 MB/s. La situazione in realtà è ancora peggiore, dato che per visualizzare un video da un hard disk, da un CD-ROM o da un DVD i dati devono passare dal lettore del disco alla memoria lungo il bus. Successivamente, per essere visualizzati, i dati devono viaggiare nuovamente sul bus per raggiungere l'adattatore grafico. È necessaria quindi una larghezza di banda di 135 MB/s solo per il video, senza considerare la larghezza di banda per la CPU e altri dispositivi.

Il bus ISA, predecessore del bus PCI aveva una larghezza di banda massima di 16,7 MB/s, determinata da una velocità di 8,33 MHz e dalla capacità di trasferire 2 byte per ciclo; il bus EISA poteva invece spostare 4 byte per ciclo, raggiungendo così 33,3 MB/s. Nessuno di questi due bus si avvicinava alla larghezza di banda necessaria per visualizzare video a schermo intero.

Con i moderni schermi Full HD la situazione è ancora peggiore. Con una risoluzione di  $1920 \times 1080$  e 30 frame/sec la larghezza di banda è di 155 MB/s (310 se i dati devono attraversare due volte il bus). Ovviamente il bus EISA non può nemmeno pensare di poter gestire questa situazione.

Nel 1990 Intel colse questa necessità e progettò un nuovo bus con una larghezza di banda molto più ampia rispetto allo stesso bus EISA (*Extended ISA*) e lo chiamò PCI (*Peripheral Component Interconnect*). Per incoraggiarne l'adozione, Intel brevettò il bus PCI e poi rese di pubblico dominio i brevetti, in modo che qualunque azienda potesse costruire delle periferiche per il bus senza pagare alcun diritto. Intel fondò inoltre un consorzio, chiamato *PCI Special Interest Group* per la gestione del futuro del bus. La conseguenza di queste azioni fu l'incremento della sua diffusione. A partire dal Pentium, praticamente ogni calcolatore basato su Intel, così come molti altri computer, aveva un bus PCI. Anche Sun ha una versione di UltraSPARC, l'UltraSPARC IIIi, che utilizza il bus PCI. Shanley e Anderson (1999) e Solari e Willse (2004) trattano il bus PCI molto approfonditamente.

Il bus PCI originario trasferiva 32 bit per ciclo e funzionava a 33 MHz (tempo di ciclo di 30 ns) per una larghezza di banda totale di 133 MB/s. Nel 1993 fu introdotto il PCI 2.0 e nel 1995 il PCI 2.1. La versione PCI 2.2 ha invece delle funzionalità specifiche per i calcolatori portatili (la maggior parte delle quali serve a risparmiare energia). Il bus

PCI può funzionare fino a 66 MHz e può gestire trasferimenti di 64 bit, per una larghezza di banda totale di 528 MB/s; con questa banda è possibile (assumendo che il disco e il resto del sistema siano pronti per il compito) visualizzare in modo fluido i video a tutto schermo.

Anche se 528 MB/s suonano come una velocità apprezzabile, al bus PCI restavano ancora due problemi. Primo, la larghezza di banda non era sufficiente per essere usata come bus di memoria. Secondo, non era compatibile con tutte le vecchie schede ISA ancora in circolazione. La decisione presa da Intel fu di progettare calcolatori con tre o più bus. Com'è possibile vedere nella Figura 3.51 la CPU può comunicare con la memoria centrale attraverso uno speciale bus di memoria; inoltre è presente un bus ISA collegato al bus PCI. Questa soluzione soddisfaceva tutti i requisiti e per questo motivo nel corso degli anni '90 fu adottata in modo molto diffuso.

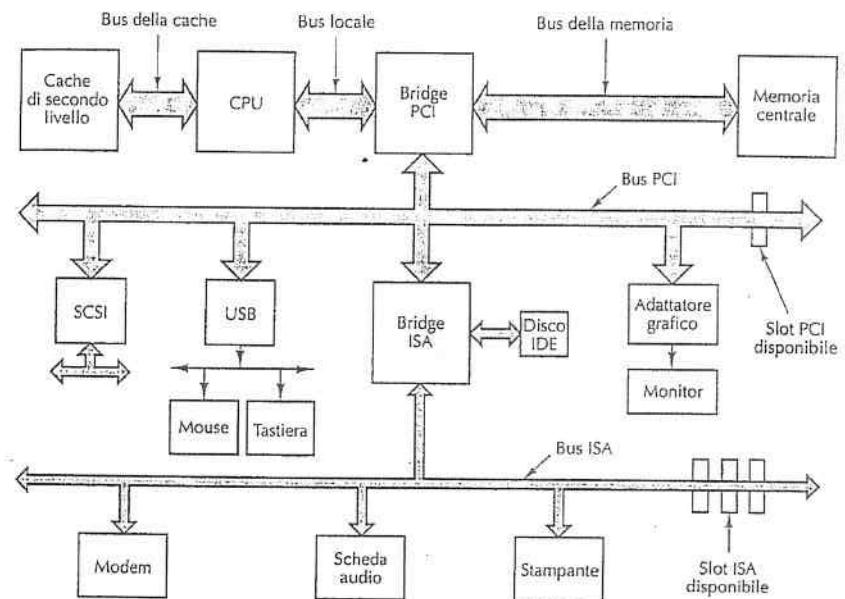


Figura 3.51 Architettura di uno dei primi Pentium. I bus più spessi hanno una maggiore larghezza di banda rispetto a quelli più sottili (la figura non è in scala).

In questa architettura le due componenti chiave sono i due chip bridge prodotti da Intel (che aveva quindi un grande interesse nell'intero progetto). Il bridge PCI connette la CPU, la memoria e il bus PCI, mentre il bridge ISA connette il bus PCI al bus ISA e supporta inoltre uno o due dischi IDE. Quasi tutti i sistemi con questa architettura hanno uno o più slot PCI liberi per aggiungere nuove periferiche ad alta velocità, e uno o più slot ISA per periferiche a bassa velocità.

Il vantaggio principale dell'architettura mostrata nella Figura 3.51 è che la CPU, usando un bus di memoria proprietario, ha una larghezza di banda estremamente alta verso la memoria. Il bus PCI offre invece un'ampia larghezza di banda per periferiche veloci come i dischi SCSI, gli adattatori grafici e così via; inoltre, grazie al bus ISA, è ancora possibile utilizzare le vecchie schede. Nella figura è presente un altro riquadro, definito USB, che si riferisce all'Universal Serial Bus e sarà trattato nel capitolo successivo.

Sarebbe stato più pratico avere un solo tipo di schede PCI. Sfortunatamente non è così, dato che vi sono varie possibilità per tensione, ampiezza e temporizzazione.

Il bus PCI supporta due diverse tensioni, dato che i calcolatori più vecchi funziona-

vano spesso a 5 volt, mentre con quelli più recenti si tende a usare 3,3 volt. I connettori sono gli stessi, tranne per due frammenti di plastica che impediscono all'utente di inserire una scheda a 5 volt in un bus PCI a 3,3 volt e viceversa. Fortunatamente esistono schede universali che supportano entrambe le tensioni e che possono essere inserite nei due tipi di slot. Le schede esistono anche in versione 32 bit oppure 64 bit. Quelle a 32 bit hanno 120 pin, e le altre hanno ulteriori 64 pin; l'aggiunta di pin è analoga al modo in cui fu esteso il bus del PC IBM per portarlo a 16 bit (vedi Figura 3.51). Un sistema con bus PCI che supporta schede a 64 bit può accettare anche schede a 32 bit, mentre non è vero il contrario. Infine i bus PCI e le loro schede possono funzionare a 33 oppure a 66 MHz e la scelta viene effettuata collegando un pin all'alimentazione oppure collegandolo alla terra. I connettori sono dello stesso tipo per entrambe le velocità.

Alla fine degli anni '90 quasi tutti furono d'accordo nel considerare il bus ISA ormai superato e decisero di escluderlo dalle nuove architetture. In quel periodo però le risoluzioni dei monitor aumentarono, raggiungendo in alcuni casi 1600 × 1200; inoltre crebbe la richiesta di video fluido e a tutto schermo, in particolar modo nell'area dei giochi interattivi. Intel decise quindi di aggiungere un altro bus per pilotare esclusivamente le schede grafiche. Questo bus prese il nome di AGP (*Accelerated Graphics Port*). La versione iniziale, AGP 1.0, funzionava a 264 MB/s. Questo bus, al quale ci si riferisce con 1x, era più lento del PCI, ma in compenso era dedicato solamente alla scheda grafica. Nel corso degli anni apparvero nuove versioni, come l'AGP 3.0 che raggiunse una larghezza di banda di 2,1 GB/s (8x). Al giorno d'oggi anche questo bus è stato soppiantato da tecnologie ancora più veloci, in particolare il bus PCI Express che può pompare la bellezza di 16 GB/sec di dati su collegamenti seriali ad alta velocità. La Figura 3.52 mostra un moderno sistema basato sul Core i7.

In un moderno sistema basato su Core i7 numerose interfacce sono integrate direttamente sul chip della CPU. I due canali di memoria DDR3 da 1333 transazioni al secondo connettono alla memoria principale e offrono una larghezza di banda aggregata di 10 GB/sec. All'interno della CPU c'è anche un canale PCI Express a 16 corsie che può essere configurato in maniera ottimale come singolo bus PCI Express a 16 bit o come due canali PCI Express indipendenti a 8 bit. L'insieme delle 16 corsie fornisce nel complesso una larghezza di banda di 16 GB/sec verso le periferiche di I/O.

La CPU si collega al bridge principale, il P67, per mezzo di un'interfaccia seriale DMI a 20 Gbit/sec (2,5 GB/sec). Il P67 dispone di numerose interfacce I/O ad alte prestazioni, offre 8 corsie PCI Express aggiuntive e l'interfaccia per dischi SATA e implementa 14 interfacce USB 2.0, una Ethernet 10G e l'interfaccia audio.

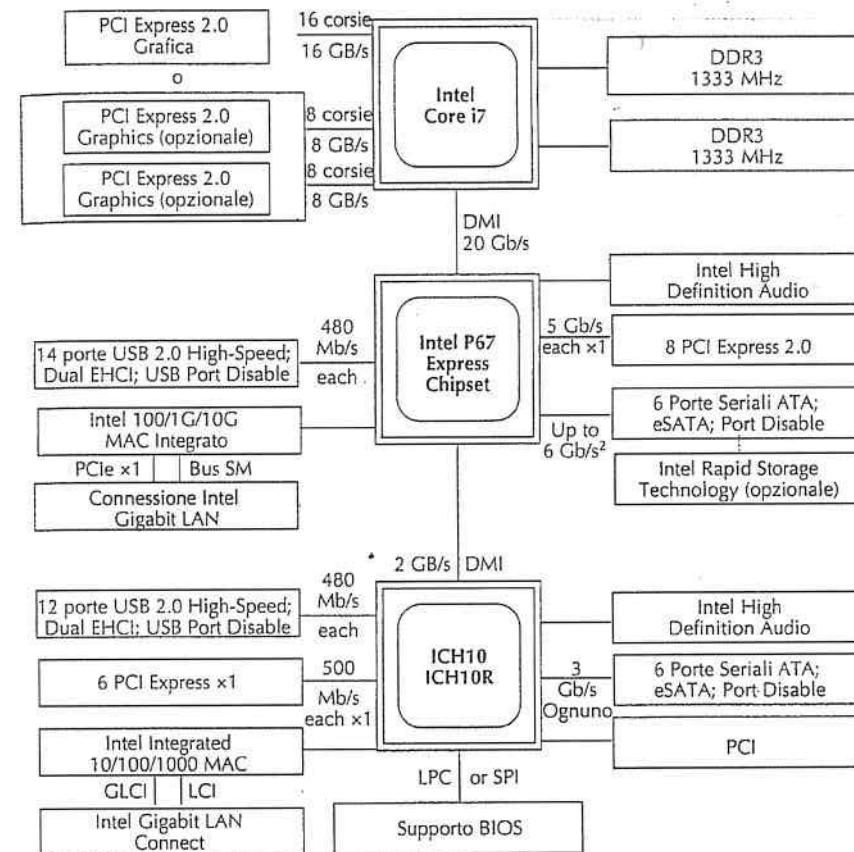


Figura 3.52 La struttura del bus di un moderno sistema Core i7.

Il chip ICH10 fornisce il supporto per le interfacce più datate e le periferiche più vecchie ed è connesso al P67 attraverso un'interfaccia DMI più lenta. ICH10 implementa il bus PCI, Ethernet 1G, porte USB, una PCI Express vecchio stile e le interfacce SATA. I nuovi sistemi potrebbero non incorporare il chip ICH10, richiesto solamente per la compatibilità con interfacce più vecchie.

Anche il bus PCI, come tutti quelli successivi al bus del PC IBM originario, è sincrono. Tutte le transazioni sul bus PCI avvengono tra un master, detto convenzionalmente **iniziatore**, e uno slave, chiamato convenzionalmente **destinatario**. Inoltre, per ridurre il numero totale di pin, le linee degli indirizzi e dei dati sono multiplexate. In questo modo sulle schede PCI sono necessari soltanto 64 pin per i segnali d'indirizzo e di dati, anche se PCI supporta sia indirizzi sia dati a 64 bit.

I pin multiplexati per gli indirizzi e per i dati funzionano nel modo seguente. Nel ciclo 1 di un'operazione di lettura il master immette l'indirizzo sul bus. Nel ciclo 2 il master rimuove l'indirizzo e il bus è invertito in modo che lo slave lo possa utilizzare; infine, nel ciclo 3, lo slave genera in output i dati richiesti. Nelle operazioni di scrittura il bus non deve essere invertito, dato che è il master a fornire sia l'indirizzo sia i dati. Ciononostante la transazione minima rimane tuttavia della durata di 3 cicli. Se non è in grado di rispondere nei tre cicli, lo slave può inserire degli stati di attesa. Sono inoltre consentiti trasferimenti di blocchi di dimensione illimitata, e diversi tipi di cicli di bus.

### Arbitraggio del bus PCI

Prima di poter utilizzare il bus PCI, i dispositivi devono acquisirlo. Come mostrato nella Figura 3.53 si utilizza un arbitro centralizzato. Nella maggior parte delle architetture l'arbitro del bus è integrato in uno dei chip bridge. Ogni dispositivo PCI ha due linee dedicate che lo collegano all'arbitro: una, REQ#, per richiedere il bus, e l'altra, GNT#, per riceverne la concessione. Una nota: REQ# è la notazione PCI per REQ.

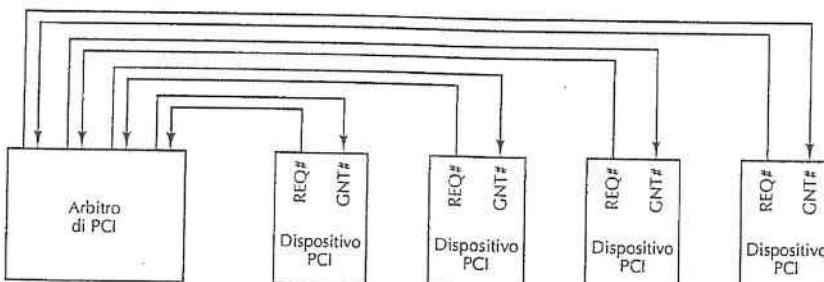


Figura 3.53 Il bus PCI utilizza un arbitro del bus centralizzato.

Per richiedere il bus un dispositivo PCI (compresa la CPU) deve asserire REQ# e attendere finché non veda che la linea GNT# è stata asserita dall'arbitro. Quando ciò si verifica il dispositivo può utilizzare il bus durante il ciclo successivo. Le specifiche PCI non definiscono l'algoritmo che l'arbitro deve utilizzare; sono consentiti arbitraggi round-robin, arbitraggi basati su priorità e altre politiche. Ovviamente un arbitro dovrebbe essere imparziale in modo che nessun dispositivo sia obbligato ad attendere all'infinito la concessione del bus.

Il bus viene concesso per una transazione alla volta, anche se può avere lunghezza teoricamente illimitata. Se un dispositivo vuole effettuare una seconda transazione e nessun altro dispositivo sta richiedendo il bus, allora può continuare a utilizzarlo; tuttavia tra la prima e la seconda transazione deve spesso essere inserito un ciclo di inattività. In particolari circostanze e in assenza di contesa per l'utilizzo del bus, un dispositivo può tuttavia effettuare transazioni in sequenza senza dover inserire un ciclo di inattività. L'arbitro può negare la linea GNT# nel caso in cui un master stia effettuando un trasferi-

mento molto lungo e contemporaneamente qualche altro dispositivo richieda il bus. Dato che il master del bus deve monitorare la linea GNT#, esso deve rilasciare il bus al ciclo successivo non appena vede che la linea è stata negata. Questo schema permette trasferimenti molto lunghi (che sono efficienti) quando c'è un solo candidato master, ma allo stesso tempo continua a garantire una risposta veloce quando più dispositivi competono per l'utilizzo del bus.

### Segnali del bus PCI

Nel bus PCI ci sono dei segnali obbligatori, mostrati nella Figura 3.54(a), e dei segnali optionali, mostrati nella Figura 3.54(b). I pin rimanenti dei 120 (o 184) totali sono utilizzati per la tensione, la terra e altre funzioni che non elencheremo. Le colonne *Master* (iniziatore) e *Slave* (destinatario) indicano chi asserisce il segnale su una normale transazione; nel caso in cui il segnale sia asserito da un altro dispositivo (per esempio, CLK) entrambe le colonne sono lasciate vuote.

Analizziamo ora, seppur brevemente, i segnali del bus PCI, iniziando con quelli obbligatori. Il segnale CLK pilota il bus ed è in sincronia con la maggior parte degli altri segnali. Diversamente dal bus ISA, le transazioni del bus PCI iniziano in corrispondenza del fronte di discesa di CLK, che si trova nel mezzo del ciclo invece che all'inizio.

I 32 segnali AD sono destinati agli indirizzi e ai dati (per le transazioni a 32 bit). In genere l'indirizzo viene asserito durante il ciclo 1, mentre i dati durante il ciclo 3. Il segnale PAR è un segnale di parità per AD. Il segnale C/BE# è utilizzato per due scopi distinti; nel ciclo 1 contiene il comando del bus (lettura di una parola, blocco e così via), mentre nel ciclo 2 contiene una stringa di 4 bit che indica quali byte di una parola a 32 bit sono validi. Usando C/BE# è possibile leggere oppure scrivere arbitrariamente 1, 2, 3 byte oppure una parola intera.

Il segnale FRAME# viene asserito dal master per iniziare una transazione sul bus comunicando allo slave che l'indirizzo e i comandi del bus sono validi. Di solito, nelle letture, IRDY# viene asserito nello stesso momento di FRAME#; esso comunica che il master è pronto ad accettare dati in ingresso. Nel caso di una scrittura, IRDY# viene invece asserito in un secondo momento, quando i dati si trovano sul bus.

Il segnale IDSEL è legato al fatto che ogni dispositivo PCI deve avere uno spazio di configurazione di 256 byte per le informazioni sulle sue proprietà e che ogni altro dispositivo può leggere (asserendo IDSEL). La proprietà Plug-and-Play di alcuni sistemi operativi utilizza questo spazio di configurazione per determinare quali dispositivi sono presenti sul bus.

Passiamo ora ai segnali asseriti dallo slave. Il primo di questi, DEVSEL#, annuncia che lo slave ha rilevato il proprio indirizzo sulle linee AD ed è pronto a instaurare una transazione. Se DEVSEL# non viene asserito entro un certo intervallo di tempo il master considera scaduto il tempo e assume che il dispositivo indirizzato è assente oppure è guasto.

Il secondo segnale per lo slave, TRDY#, viene asserito nelle letture per annunciare che i dati sono disponibili sulle linee AD, mentre nelle scritture per comunicare che è pronto ad accettare i dati.

Segnale	Linee	Master	Slave	Descrizione
CLK	1			Clock (33 MHz oppure 66 MHz)
AD	32	x	x	Linee d'indirizzo e dei dati multiplexate
PAR	1	x		Bit di parità dell'indirizzo e dei dati
C/BE	4	x		Comando del bus/bit map per i byte abilitati
FRAME#	1	x		Indica che AD e C/BE sono asserite
IRDY#	1	x		Lettura: il master accetterà i dati; scrittura: i dati sono presenti
IDSEL	1	x		Seleziona lo spazio di configurazione invece che quello di memoria
DEVSEL#	1		x	Lo slave ha decodificato il proprio indirizzo ed è in ascolto
TRDY#	1		x	Lettura: i dati sono presenti; scrittura: lo slave accetterà i dati
STOP#	1		x	Lo slave vuole interrompere immediatamente la transazione
PERR#	1			Il destinatario ha rilevato un errore sulla parità dei dati
SERR#	1			Errore sulla parità dell'indirizzo ed errore di sistema
REQ#	1			Arbitraggio del bus: richiesta di appropriazione del bus
GNT#	1			Arbitraggio del bus: concessione del bus
RST#	1			Reinizializza il sistema e tutti i dispositivi

(a)

Segnale	Linee	Master	Slave	Descrizione
REQ64#	1	x		Richiesta di eseguire una transazione a 64 bit
ACK64#	1		x	Concessione per eseguire una transazione a 64 bit
AD	32	x		32 bit aggiuntivi per l'indirizzo o per i dati
PAR64	1	x		Parità per i 32 bit aggiuntivi (indirizzo/dati)
C/BE64	4	x		4 bit aggiuntivi per l'abilitazione dei byte
LOCK	1	x		Blocco del bus per permettere transazioni multiple
SBO#	1			Successo di una cache remota (per un multiprocessore)
SDONE	1			È stato effettuato lo snooping (per un multiprocessore)
INTx	4			Richiesta di un interrupt
JTAG	5			Segnali per i test JTAG IEEE 1149.1
M66EN	1			Collegato all'alimentazione o terra (66 o 33 MHz)

(b)

Figura 3.54 Segnali (a) obbligatori e (b) opzionali del bus PCI.

I tre segnali successivi servono a notificare un errore. Il primo è **STOP#**: lo slave lo asserisce se si verifica qualcosa di disastroso e se desidera cancellare la transazione corrente. Il successivo, **PERR#**, è utilizzato per notificare che nel ciclo precedente si è verificato un errore sulla parità dei dati. In lettura viene asserito dal master, e in scrittura dal slave. Infine, **SERR#** serve a notificare errori di sistema o riguardanti gli indirizzi.

I segnali **REQ#** e **GNT#** sono utilizzati per l'arbitraggio del bus e non sono asseriti dal master corrente, ma dal dispositivo che intende diventare master del bus. L'ultimo segnale obbligatorio è **RST#**, utilizzato per resettare il sistema quando l'utente preme il pulsante RESET oppure quando un dispositivo notifica un errore irrimediabile. Quando questo segnale viene asserito tutti i dispositivi vengono resettati e il calcolatore viene riavviato.

Passiamo ora ai segnali opzionali, la maggior parte dei quali riguarda l'espansione da 32 a 64 bit. I segnali **REQ64#** e **ACK64#** consentono al master di richiedere l'autorizzazione a effettuare una transazione a 64 bit, e allo slave di accettarla. I segnali **AD**, **PAR64** e **C/BE64** sono semplicemente un'estensione dei corrispondenti segnali a 32 bit.

I tre segnali successivi non riguardano il passaggio da 32 a 64 bit, ma i sistemi multiprocessore, una funzionalità che le schede PCI non sono obbligate a dover supportare. Il segnale **LOCK** permette di bloccare il bus per transazioni multiple, mentre i due successivi riguardano lo snooping necessario per mantenere la coerenza della cache.

I segnali **INTx** sono utilizzati per richiedere interrupt. Sulle schede PCI ci possono essere fino a quattro dispositivi logici distinti e ciascuno di loro può avere la propria linea per la richiesta di interrupt. Il segnale **JTAG** serve per la procedura di test JTAG IEEE 1149.1. Infine il segnale **M66EN** è collegato a un valore alto o basso per impostare la velocità di clock e non deve essere modificato mentre il sistema è in funzione.

### Transazioni del bus PCI

Il bus PCI (come in genere tutti i bus) è in realtà molto semplice. Per entrare ancora più in sintonia con il suo funzionamento consideriamo il diagramma di temporizzazione della Figura 3.55 che mostra una transazione di lettura, seguita da un ciclo d'inattività, seguito a sua volta da una transazione di scrittura, effettuate tutte dallo stesso master del bus.

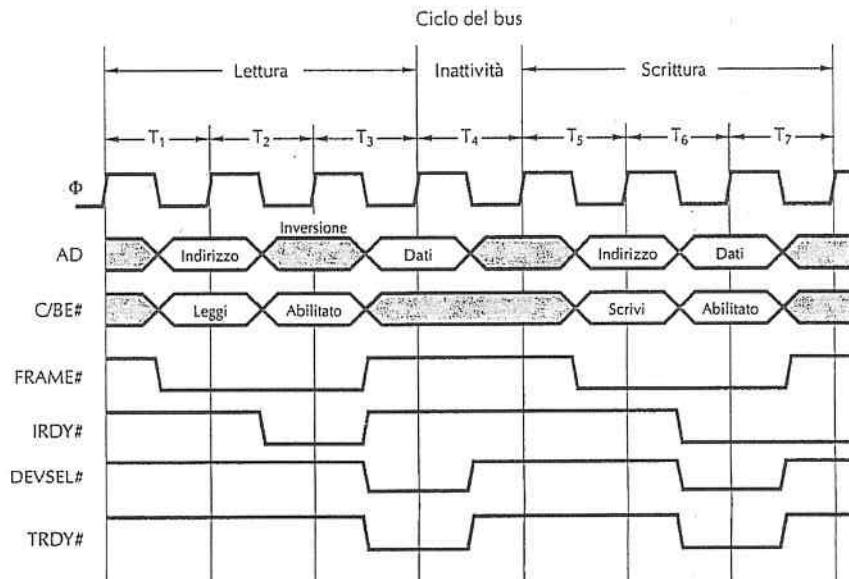


Figura 3.55 Esempi di transazioni di un bus PCI a 32 bit. I primi tre cicli sono utilizzati per un'operazione di lettura. Segue un ciclo d'inattività e infine tre cicli per un'operazione di scrittura.

Quando durante  $T_1$  si verifica il fronte di discesa del clock, il master inserisce l'indirizzo di memoria su AD e il comando del bus su C/BE#; asserisce inoltre FRAME# per iniziare la transazione sul bus.

Durante  $T_2$ , il master rilascia l'indirizzo del bus per permettere che venga invertito in modo che lo slave possa pilotarlo durante  $T_3$ . Il master modifica inoltre C/BE# per indicare quali byte della parola indirizzata vuole abilitare (cioè leggere).

Durante  $T_3$ , lo slave asserisce DEVSEL# in modo che il master sappia che ha rilevato il proprio indirizzo e che si sta adoperando per rispondere. Inoltre scrive i dati sulle linee AD e asserisce TRDY# per notificare il fatto al master. Se lo slave non è in grado di rispondere velocemente deve tuttavia asserire DEVSEL# per annunciare la propria presenza, ma deve lasciare TRDY# negato finché non possa fornire i dati richiesti. Questa procedura potrebbe inserire uno più stati di attesa. In questo esempio (e spesso nelle situazioni reali) il ciclo successivo è inattivo. All'inizio di  $T_5$  vediamo lo stesso master iniziare una scrittura. Come al solito inizia scrivendo sul bus l'indirizzo e il comando del bus. In questo caso però asserisce i dati già nel secondo ciclo; infatti, dato che è lo stesso dispositivo a pilotare le linee AD, non c'è bisogno di un ciclo per l'inversione del bus. In  $T_7$ , la memoria accetta i dati.

### 3.6.2 PCI Express

Anche se il bus PCI funziona in modo adeguato per la maggior parte delle applicazioni attuali, la richiesta di una maggior larghezza di banda per l'I/O sta creando confusione nell'architettura dei PC, un tempo più ordinata. Dalla Figura 3.53 risulta chiaro che il bus PCI non è più l'elemento centrale che tiene unite le varie parti del PC. Parte di quel ruolo è stato preso dal chip bridge.

Il cuore del problema è che ci sono sempre più dispositivi di I/O la cui velocità è eccessiva per il bus PCI. Spingere ulteriormente la frequenza di clock non è una buona idea, perché i problemi di disallineamento del bus, interferenze tra i cavi ed effetti di capacità renderebbero la situazione ancora peggiore. Ogni volta che un dispositivo di I/O è diventato troppo veloce per il bus PCI (come le schede grafiche, gli hard disk, le reti e così via), Intel ha aggiunto al chip bridge una nuova e speciale porta per permettere a quel dispositivo di scavalcare il bus PCI. Ovviamente questa non può essere una soluzione a lungo termine.

Un ulteriore problema del bus PCI è che le schede sono piuttosto grandi. In genere le schede PCI standard misurano 17,5 x 10,7 cm e quelle più piccole misurano 12 x 3,6 cm. Nessuna di queste schede può trovare comodamente spazio in un computer portatile, né tantomeno in un dispositivo mobile, quando invece i produttori cercano di produrre dispositivi ancora più piccoli. Alcune compagnie vorrebbero inoltre separare i componenti del PC, inserendo per esempio la CPU e la memoria in un piccolo contenitore sigillato e l'hard disk all'interno del monitor. Con le schede PCI questo non è possibile.

Sono state proposte varie soluzioni, ma quella che ha vinto un posto nei computer attuali (aveva possibilità rilevanti, dato che c'era dietro Intel) è chiamata PCI Express. Questo prodotto ha poco a che fare con il bus PCI, e tra l'altro non è nemmeno un vero e proprio bus, ma gli addetti al marketing non volevano lasciarsi sfuggire un nome così

conosciuto come PCI. I PC con un bus PCI Express sono oggi standard. Scopriamo ora come funziona questo bus.

### Architettura di PCI Express

Il cuore della soluzione PCI Express (spesso abbreviata in PCIe) è la rinuncia al bus parallelo con i suoi numerosi master e slave e la scelta di un'architettura basata su connessioni seriali punto-a-punto ad alta velocità. Questa soluzione rappresenta un radicale elemento di rottura con la tradizione ISA/EISA/PCI e adotta molte idee provenienti dal mondo delle reti locali e in particolare dalla commutazione Ethernet. L'idea base si riduce a questo: in fondo in fondo un PC è un insieme di CPU, memoria e chip controller di I/O, che necessitano di essere connessi fra loro. Quello che fa PCI Express è fornire un commutatore di uso generale per connettere i chip mediante collegamenti seriali. La Figura 3.56 illustra una tipica configurazione.

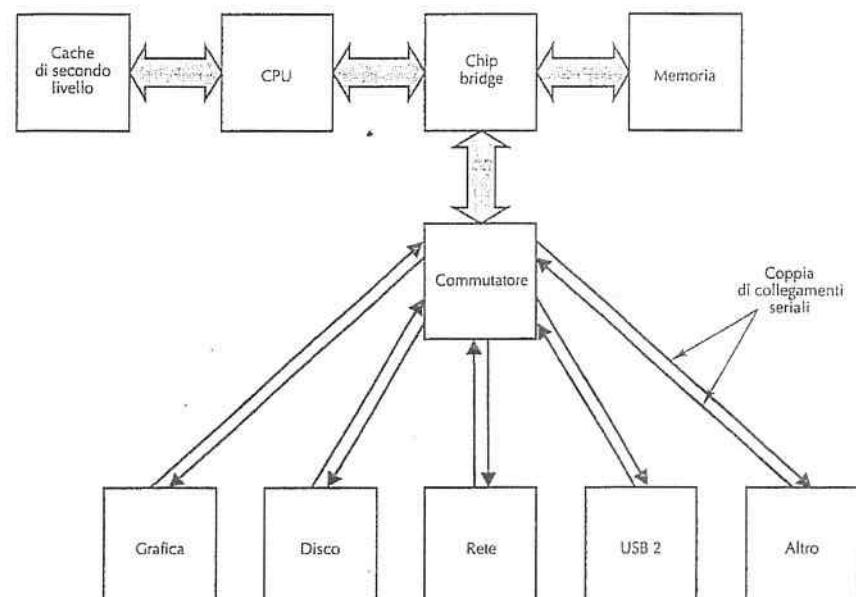


Figura 3.56 Tipico sistema PCI Express.

Come mostra la Figura 3.56, CPU, memoria e cache sono connesse al chip bridge in modo tradizionale. La novità è rappresentata da un commutatore connesso al bridge (parte integrante del bridge stesso, oppure integrato direttamente all'interno del processore). Ogni chip di I/O ha una connessione dedicata punto-a-punto con il commutatore, che consiste in una coppia di canali unidirezionali, uno che giunge al commutatore e uno

che parte da esso. Ciascun canale è composto da due fili, uno per trasportare il segnale e uno per la terra, in modo da fornire un'elevata immunità al rumore durante le connessioni ad alta velocità. Questa architettura ha sostituito le precedenti con uno schema molto più uniforme, nel quale tutti i dispositivi sono trattati in modo uguale.

L'architettura PCI Express differisce da quella vecchia basata sul bus PCI in tre aspetti principali. Due di loro sono già stati esaminati: un commutatore centralizzato al posto di un bus con più connessioni e l'uso di strette connessioni seriali punto-a-punto al posto di un ampio bus parallelo. La terza differenza è più sottile. Il modello concettuale del bus PCI è quello di un master che lancia allo slave un comando per leggere una parola oppure un blocco di parole. Il modello di PCI Express è invece quello di un dispositivo che spedisce un pacchetto di dati a un altro dispositivo. Il concetto di pacchetto, che consiste di un'intestazione e di un campo dati, deriva dal mondo delle reti. L'intestazione contiene le informazioni di controllo, eliminando quindi la necessità dei vari segnali di controllo presenti sul bus PCI; il campo dati contiene invece i dati che devono essere trasferiti. Un PC con PCI Express è in realtà una rete a commutazione di pacchetto in miniatura.

Oltre a queste tre principali rotture con il passato vi sono anche altre differenze. Per esempio, l'utilizzo di un codice per la rilevazione degli errori nei pacchetti fornisce un grado di affidabilità più elevato rispetto a quello di PCI. Inoltre, la connessione tra un chip e il commutatore è diventata più lunga, arrivando fino a 50 cm e permettendo il partizionamento del sistema; si introduce l'espandibilità, dato che un dispositivo può essere in realtà un altro commutatore (fino a un massimo di tre). E ancora: i dispositivi sono inseribili "a caldo", cioè possono essere aggiunti o rimossi dal sistema mentre è in funzione. Infine, dato che le connessioni seriali sono molto più piccole dei vecchi connettori PCI, i dispositivi e i calcolatori possono essere realizzati in dimensioni inferiori. Insomma, PCI Express rappresenta una deviazione molto significativa rispetto alla strada del bus PCI.

### Pila di protocolli di PCI Express

Il sistema PCI Express, rimanendo aderente al modello delle reti a commutazione di pacchetto, possiede una pila stratificata di protocolli. Un **protocollo** è un insieme di regole che governano la comunicazione tra due parti. Una pila di protocolli è una gerarchia di protocolli, che gestisce diversi problemi in livelli distinti. Consideriamo per esempio una lettera commerciale. Esistono precise convenzioni riguardo il posizionamento e il contenuto dell'intestazione, l'indirizzo del destinatario, la data, la formula di saluto, il formato, la firma e così via. Tutto ciò potrebbe essere visto analogamente. Inoltre esiste un altro gruppo di convenzioni riguardo la busta, come la sua dimensione, dove e in che formato indicare gli indirizzi del mittente e del destinatario, dove affrancare il francobollo e così via. Questi due livelli e i loro protocolli sono indipendenti l'uno dall'altro. È possibile per esempio cambiare completamente l'impaginazione della lettera, pur utilizzando la stessa busta o viceversa. La stratificazione dei protocolli consente una progettazione modulare e flessibile e per decenni è stata usata ampiamente nel mondo del software delle reti. La novità è che in questo caso viene adottata nell'hardware del "bus". La pila di protocolli di PCI Express è mostrata nella Figura 3.57(a), e discussa di seguito.

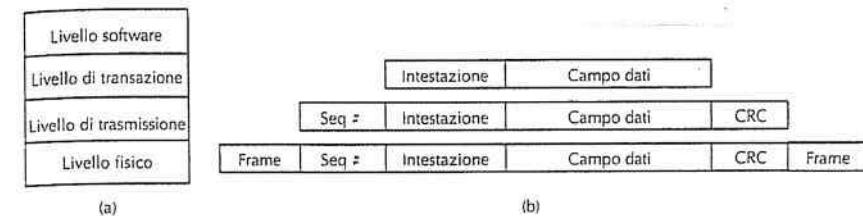


Figura 3.57 (a) Pila di protocolli di PCI Express. (b) Formato di un pacchetto.

Esaminiamo i livelli a partire dal basso. Il livello inferiore è il **livello fisico** che tratta lo spostamento dei bit da un mittente a un destinatario lungo una connessione punto-a-punto. Ciascuna connessione punto-a-punto consiste in una o più coppie di collegamenti simplex (cioè unidirezionali). Nel caso più semplice vi è soltanto una coppia in ciascuna direzione, ma se ne possono avere anche 2, 4, 8, 16 o 32. Ciascun collegamento è chiamato **corsia** e il numero di corsie in ciascuna direzione deve essere lo stesso. La prima generazione di prodotti deve supportare un tasso di trasferimento dati in ciascuna direzione di almeno 2,5 Gbps, ma ci si aspetta che la velocità in ciascuna direzione raggiungerà fra non molto i 10 Gbps.

Diversamente dai bus ISA/EISA/PCI, PCI Express non ha un clock principale. I dispositivi sono liberi di trasmettere non appena hanno dei dati da spedire; questa libertà rende più veloce il sistema, ma solleva un problema. Supponiamo che il bit 1 sia codificato con +3 volt e il bit 0 con 0 volt. Se i primi byte sono tutti 0 come può il destinatario sapere che ci sono dei dati in fase di trasmissione? Dopo tutto una sequenza di bit 0 ha lo stesso identico aspetto di un collegamento inattivo. Questo problema è risolto utilizzando quella che viene chiamata **codifica 8b/10b**, in cui si utilizza un simbolo a 10 bit per codificare 1 byte di dati effettivi. Fra tutte le 1024 possibili stringhe di 10 bit sono state opportunamente scelte quelle che hanno sufficienti transizioni del clock per tenere sincronizzati il mittente e il destinatario sul limite di un bit, anche senza l'utilizzo di un clock principale. La conseguenza della codifica 8b/10b è che un link con la capacità grezza di 2,5 Gbps può trasportare soltanto 2 Gbps di dati (netti) dell'utente.

Se il livello fisico gestisce la trasmissione di bit, il **livello di trasmissione** si occupa della trasmissione dei pacchetti. Esso prende l'intestazione e il campo dati forniti dal livello di transazione e vi aggiunge un numero di sequenza e un codice a correzione di errore chiamato **CRC** (*Cyclic Redundancy Check*, "controllo a ridondanza ciclica"). Il CRC viene generato eseguendo un particolare algoritmo sull'intestazione e sul campo dati. Quando il destinatario riceve un pacchetto esegue la stessa computazione sull'intestazione e sul campo dati e confronta il proprio risultato con il CRC trasportato nel pacchetto. Se corrispondono, restituisce un breve **pacchetto di acknowledgment** ("conferma") per confermare la corretta ricezione del pacchetto. Se invece i due valori non corrispondono, il destinatario richiede una ritrasmissione. In questo modo l'integrità dei dati risulta maggiore rispetto ai sistemi basati sul bus PCI, che non si preoccupano di verificare ed eventualmente ritrasmettere i dati lungo il bus.

Si utilizza inoltre un meccanismo di controllo di flusso per evitare che un mittente veloce sommerga di pacchetti un destinatario più lento. Nel meccanismo utilizzato un destinatario dà al mittente un certo numero di crediti, corrispondenti alla quantità di spazio di buffer che ha a disposizione per memorizzare i pacchetti in entrata. Quando terminano i crediti il mittente deve interrompere la spedizione finché non ottenga nuovi crediti. Questo schema, utilizzato diffusamente in tutte le reti, evita che i dati vadano persi a causa di una differenza di velocità tra il mittente e il destinatario.

Il livello di transazione gestisce le azioni sul bus. La lettura di una parola dalla memoria richiede due transazioni: una iniziata dalla CPU o dal canale DMA per richiedere alcuni dati e una iniziata dal mittente per fornire i dati. I compiti del livello di transazione vanno però oltre la semplice gestione di letture e scritture. Esso aggiunge un valore al pacchetto di trasmissione fornитogli dal livello di trasmissione. Per iniziare, la trasmissione può dividere ciascuna corsia in più circuiti virtuali, fino a un massimo di otto, ciascuno dei quali gestisce una diversa classe del traffico. Il livello di transazione può etichettare i pacchetti in base alla loro classe di traffico, che può comprendere attributi che indicano per esempio: alta priorità, bassa priorità, non effettuare lo snooping, ammettere la consegna in ordine arbitrario, e altro ancora. Il commutatore può utilizzare questi tag quando deve scegliere il nuovo pacchetto da gestire.

Ciascuna transazione utilizza uno dei quattro seguenti spazi d'indirizzi.

1. Spazio di memoria (per letture e scritture ordinarie).
2. Spazio di I/O (per indirizzare i registri del dispositivo).
3. Spazio di configurazione (per l'inizializzazione del sistema e così via).
4. Spazio dei messaggi (per la segnalazione, gli interrupt e così via).

Lo spazio di memoria e lo spazio di I/O sono simili a quelli dei sistemi attuali. Lo spazio di configurazione può essere utilizzato per implementare funzionalità come il plug-and-play. Lo spazio dei messaggi prende il ruolo di molti dei segnali di controllo esistenti, ed è necessario in quanto nessuna delle linee di controllo del bus PCI è presente in PCI Express.

Il livello software che interfaccia il sistema PCI Express al sistema operativo può emulare il bus PCI in modo che sia possibile utilizzare sistemi operativi non modificati. Operando in questo modo ovviamente non si trae pieno vantaggio dalla potenza di PCI Express; la retrocompatibilità è tuttavia necessaria finché i sistemi operativi non saranno modificati per utilizzare appieno PCI Express. L'esperienza mostra che ciò può richiedere un certo periodo di tempo.

La Figura 3.58(b) mostra il flusso d'informazioni. Quando viene passato un comando a livello software, esso lo passa a livello di transazione, che lo riformula in termini di intestazione e campo dati. Queste due parti sono poi passate a livello di trasmissione, che aggiunge un numero di sequenza in testa e il CRC in coda. Questo pacchetto, allungato, viene successivamente passato a livello fisico, che aggiunge a entrambe le estremità informazioni per formare un pacchetto fisico che infine può essere effettivamente trasmesso. Dal lato del destinatario si svolge il processo inverso, durante il quale vengono eliminate l'intestazione e la coda e il risultato viene passato a livello di transazione.

Il concetto secondo il quale ogni livello aggiunge nuova informazione ai dati mentre lo passa al protocollo sottostante è stato utilizzato per decenni e con grande successo nel mondo delle reti. La differenza principale tra una rete e PCI Express è che nel campo delle reti il codice dei vari livelli è quasi sempre una parte software del sistema operativo, mentre con PCI Express fa interamente parte dell'hardware del dispositivo.

PCI Express è un argomento complesso e, per maggiori informazioni, si consultino (Mayhew e Krishnan, 2003; Solari e Congdon, 2005). Si tratta anche di una tecnologia in evoluzione. Nel 2007 è stato rilasciato PCIe 2.0, con velocità di 500 MB/s per corsia e supporto per 32 corsie, per una larghezza di banda totale di 16 GB/s. Nel 2011 è quindi arrivato PCIe 3.0, che ha cambiato la codifica da 8b/10b a 128b/130b e che può eseguire 8 miliardi di transazioni al secondo, il doppio di PCIe 2.0.

### 3.6.3 Universal Serial Bus

I bus PCI e PCI Express sono adatti per collegare periferiche ad alta velocità al calcolatore, ma sono troppo costosi per dispositivi a bassa velocità come tastiere e mouse. Tradizionalmente i dispositivi standard di I/O erano connessi al calcolatore in un modo specifico, mentre restavano liberi alcuni slot ISA e PCI per aggiungere nuovi dispositivi. Purtroppo questo schema è stato fonte di problemi fin dall'inizio.

Per esempio, ciascun nuovo dispositivo di I/O dispone sia della scheda ISA sia di quella PCI e spesso è compito dell'utente impostare interruttori e ponticelli per assicurarsi che non vadano in conflitto con le altre schede. L'utente deve quindi aprire il contenitore, inserire attentamente la scheda, chiudere il contenitore e riavviare il sistema. Per molti utenti queste operazioni appaiono complesse e si prestano a errori; inoltre il numero di slot ISA e PCI è molto limitato (in genere due o tre). Anche con le schede plug-and-play l'utente deve aprire il calcolatore per inserire la scheda; inoltre non si risolve il problema dei pochi slot a disposizione.

Nel 1993, per risolvere questo problema, i rappresentanti di sette società (Compaq, DEC, IBM, Intel, Microsoft, NEC, e Northern Telecom) si riunirono per progettare un modo migliore per collegare a un calcolatore periferiche a bassa velocità. In seguito, a queste società se ne aggiunsero altre centinaia e, nel 1998 venne rilasciato uno standard detto USB (*Universal Serial Bus*, "bus seriale universale") che oggi è implementato diffusamente nei personal computer. Lo standard è descritto in modo approfondito in Anderson (1997) e Tan (1997).

In seguito sono elencati alcuni degli obiettivi delle aziende che hanno concepito originariamente USB e che hanno dato via al progetto.

1. Gli utenti non dovrebbero essere obbligati a impostare interruttori e contatti sulle schede dei dispositivi.
2. Gli utenti non dovrebbero essere obbligati ad aprire il computer per installare nuovi dispositivi di I/O.
3. Dovrebbe esserci un solo tipo di cavo per connettere tutti i dispositivi.
4. I dispositivi di I/O dovrebbero trarre la propria alimentazione dal cavo.
5. Dovrebbe essere possibile collegare fino a 127 dispositivi a un calcolatore.

6. Il sistema dovrebbe supportare dispositivi funzionanti in tempo reale (per esempio dispositivi audio, telefoni).
7. I dispositivi dovrebbero essere installabili a calcolatore acceso.
8. Non dovrebbe essere necessario riavviare il sistema dopo aver installato un nuovo dispositivo.
9. Il nuovo bus e i suoi dispositivi di I/O dovrebbero avere bassi costi di produzione.

USB raggiunge tutti questi obiettivi. USB è progettato per dispositivi a bassa velocità come tastiere, mouse, macchine fotografiche digitali, scanner, telefoni digitali e così via. La Versione 1.0 ha una larghezza di banda di 1,5 Mbps, che è sufficiente per le tastiere e i mouse, mentre la Versione 1.1 funziona a 12 Mbps, un valore sufficiente per stampanti, macchine fotografiche digitali e molti altri dispositivi. La versione 2.0 offre ai dispositivi una velocità massima di 480 Mbps, sufficiente per il supporto di dischi fissi esterni, webcam ad alta definizione e interfacce di rete. Il più recente USB 3.0 spinge la velocità a 5 Gbps. Solamente il tempo ci dirà quali nuove e super-assetate applicazioni scaturiranno da questa interfaccia con larghezza di banda altissima.

Un sistema USB è composto da un hub principale che si collega al bus del sistema (vedi Figura 3.51) e che possiede delle prese per i cavi che connettono i dispositivi di I/O o per hub di espansione, in modo da fornire un maggior numero di prese. La topologia di un sistema USB è quindi un albero la cui radice è l'hub principale. I cavi hanno connettori diversi sull'estremità dell'hub e su quella del dispositivo, in modo da evitare che l'utente possa accidentalmente connettere due prese dell'hub.

Il cavo consiste in quattro collegamenti: due per i dati, uno per l'alimentazione (+5 volt) e uno per la terra. Il sistema di segnalazione trasmette uno 0 come una transizione di tensione e un 1 come l'assenza di transizione; in questo modo una lunga sequenza di 0 genera un impulso regolare.

Quando viene collegato un nuovo dispositivo l'hub principale rileva l'evento e interrompe il sistema operativo; quest'ultimo interroga il dispositivo per sapere di che periferica si tratta e di quanta banda ha bisogno. Se il sistema operativo decide che c'è sufficiente larghezza di banda per il dispositivo, gli assegna un indirizzo univoco (tra 1 e 127) e scarica nel dispositivo, oltre a questo indirizzo, anche altre informazioni necessarie a configurare i registri. In questo modo è possibile connettere nuovi dispositivi "al volo", senza alcuna configurazione da parte dell'utente e senza dover installare una scheda ISA o PCI. Le schede non inizializzate hanno indirizzo 0 e possono dunque essere indirizzate. Per rendere più semplici i collegamenti molti dispositivi USB contengono al loro interno un hub, in modo da poter accettare ulteriori periferiche USB. Un monitor può per esempio avere due porte USB per potervi collegare gli altoparlanti destro e sinistro.

Da un punto di vista logico il sistema USB può essere visto come un insieme di *pipe* (condotti) di bit che partono dall'hub principale e arrivano alle periferiche di I/O. Ciascun dispositivo può dividere la sua pipe in un massimo di 16 condotti per diversi tipi di dati (per esempio, audio e video). All'interno di ogni condotto i dati viaggiano dall'hub principale alla periferica o viceversa, mentre non esiste alcun traffico di dati tra due dispositivi di I/O.

Esattamente ogni  $1,00 \pm 0,05$  ms, l'hub principale invia in broadcast un nuovo frame per mantenere sincronizzati tutti i dispositivi. Un frame è associato a un condotto di bit e consiste in pacchetti, il primo dei quali viaggia dall'hub principale verso il dispositivo. I pacchetti successivi di un frame possono viaggiare nello stesso senso oppure dal dispositivo all'hub principale. La Figura 3.58 mostra una sequenza di quattro frame. Come si può vedere, dato che i frame 0 e 2 non implicano alcuna azione è sufficiente un pacchetto SOF (*Start of Frame*, "inizio del pacchetto"). Questo pacchetto viene sempre inviato in broadcast a tutti i dispositivi. Per fare un esempio il frame 1 potrebbe essere un'interrogazione a uno scanner, affinché restituiscano i bit che ha rilevato nell'immagine che sta scannerizzando; il frame 3 potrebbe consistere invece nella spedizione di dati a qualche dispositivo, per esempio una stampante.

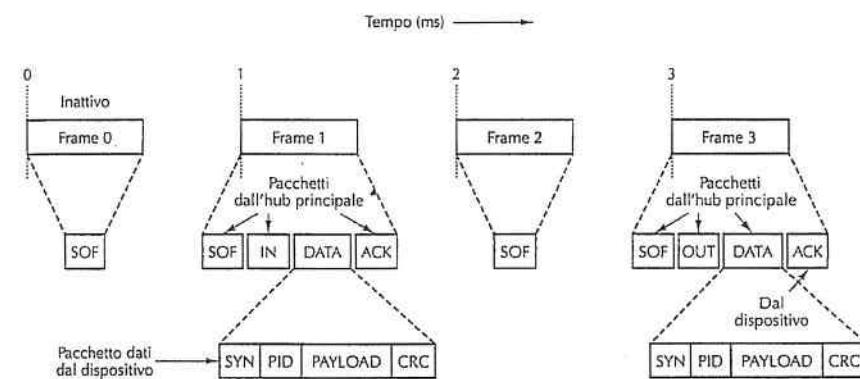


Figura 3.58 L'hub principale USB invia frame ogni millisecondo.

USB supporta quattro tipi di frame: controllo, isocrono, *bulk* e interrupt. I frame di controllo sono usati per configurare i dispositivi, assegnare loro dei comandi e interrogarli sul loro stato. I frame isocroni servono per i dispositivi funzionanti in tempo reale (come microfoni, altoparlanti o telefoni) che devono spedire o accettare dati a precisi intervalli temporali. Hanno un ritardo altamente prevedibile, ma non forniscono alcuna ritrasmissione in caso di errore.

I frame *bulk* sono utilizzati per trasferimenti di grandi dimensioni per dispositivi che non richiedono un funzionamento in tempo reale, come le stampanti. Infine i frame di interrupt sono necessari in quanto USB non supporta gli interrupt. Per esempio, invece di avere una tastiera che provoca un interrupt ogni volta che viene premuto un tasto, il sistema operativo la interroga ogni 50 ms per raccogliere i dati pendenti relativi alla pressione dei tasti.

Un frame è composto da uno o più pacchetti, alcuni dei quali possono spostarsi in entrambe le direzioni. Esistono quattro tipi di pacchetti: *token*, dati, *handshake* e pacchetti speciali. I pacchetti *token* (come SOF, IN e OUT della Figura 3.59) viaggiano

dall'hub principale fino alla periferica e servono per il controllo del sistema. Il pacchetto **SOF** (*Start of Frame*) è il primo di ogni frame e ne marca l'inizio: se non c'è alcuna operazione da compiere esso è l'unico pacchetto del frame. Il pacchetto token **IN** è un'interrogazione (*poll*) utilizzata per richiedere al dispositivo la resistuzione di alcuni dati. I suoi campi indicano quale flusso di bit viene interrogato in quel momento, in modo che il dispositivo possa sapere quale dato deve restituire (nel caso in cui abbia più flussi). Il pacchetto token **OUT** segnala che seguiranno dei dati per il dispositivo. Un quarto tipo di pacchetto token, **SETUP** (non mostrato nella figura), è utilizzato invece per la configurazione.

Il formato dei pacchetto dati (mostrato nella Figura 3.59) consiste in un campo di 8 bit per la sincronizzazione, in un identificatore a 8 bit del tipo di pacchetto (PID), nel campo dati e in un CRC a 16 bit per rilevare gli errori. Esistono tre tipi di pacchetti handshake: **ACK** (il pacchetto precedente è stato ricevuto correttamente), **NAK** (è stato rilevato un errore CRC) e **STALL** (attendere prego, in questo momento sono occupati).

Relativamente alla Figura 3.59, osserviamo che ogni millisecondo l'hub principale deve spedire un frame, anche se non deve essere effettuata alcuna operazione. I frame 0 e 2 consistono semplicemente di un pacchetto **SOF**, che indica che non vi è nulla da fare. Il frame 1 è un'interrogazione e inizia quindi con i pacchetti **SOF** e **IN** che viaggiano dal calcolatore alla periferica, seguiti poi da un pacchetto **DATA** dalla periferica al calcolatore. Il pacchetto **ACK** comunica alla periferica che i dati sono stati ricevuti correttamente; in caso di errore viene invece restituito un pacchetto **NAK** e viene rispedito il pacchetto bulk di dati (ma ciò non avviene per i dati isocroni). La struttura del frame 3 è simile a quella del frame 1, tranne per il fatto che in questo caso il flusso di dati va dal calcolatore al dispositivo.

Dopo aver terminato, nel 1998, la definizione dello standard USB, i tecnici coinvolti nel progetto, non avendo nient'altro da fare, cominciarono a lavorare a una nuova versione ad alta velocità di USB: **USB 2.0**. Questo standard è simile al vecchio USB 1.1 ed è retrocompatibile con esso, tranne per il fatto che aggiunge una terza velocità, pari a 480 Mbps, alle due già esistenti. Sono state inoltre introdotte alcune differenze di minor entità, che riguardano per esempio l'interfaccia tra hub principale e controllore. USB 1.1 ammetteva due tipi d'interfaccia. Il primo, **UHCI** (*Universal Host Controller Interface*), fu progettato da Intel lasciando gran parte dell'onere agli ingegneri software (leggi: Microsoft). Il secondo, **OHCI** (*Open Host Controller Interface*), fu progettato da Microsoft scaricando gran parte del lavoro agli ingegneri hardware (leggi: Intel). In USB 2.0 tutti si sono trovati d'accordo nell'utilizzare una nuova e unica interfaccia chiamata **EHCI** (*Enhanced Host Controller Interface*).

USB, che ora funziona a 480 Mbps, è diventato un chiaro concorrente del bus seriale IEEE 1394, indicato di solito con il nome FireWire, che raggiunge la velocità di 400 o 800 Mbps. Visto che virtualmente ogni computer basato su Intel ha il supporto USB 2.0 o USB 3.0 è probabile che, a tempo debito, 1394 spariscia. Questa perdita non sarebbe portata tanto da questioni di tecnologia, quanto piuttosto da una guerra per il territorio. USB è un prodotto dell'industria dei computer, mentre 1394 è un prodotto dell'industria dell'elettronica di consumo. Quando si deve collegare una fotocamera a un computer ogni produttore vorrebbe sì usasse il suo cavo, e questa volta pare abbiano

vinto quelli dei computer. Lo standard USB 3.0 è stato annunciato otto anni dopo l'introduzione di USB 2.0.

USB 3.0 supporta sul suo cavo l'enorme larghezza di banda di 5 Gbps, sebbene la modulazione del collegamento sia adattativa e si riesca dunque a raggiungere la velocità massima solo con un cablaggio di livello professionale. I dispositivi USB 3.0 sono strutturalmente identici ai precedenti dispositivi USB e implementano totalmente lo standard USB 2.0: se collegati con una presa USB 2.0, funzionano correttamente.

## 3.7 Interfacce

Un classico calcolatore di piccole e medie dimensioni è composto da un chip della CPU, da un chipset, da alcuni chip di memoria e da alcuni dispositivi di I/O, tutti connessi fra loro mediante un bus. A volte tutti questi componenti vengono integrati in un unico SoC, come nel caso del TI OMAP4430. Abbiamo già esaminato alcuni dettagli delle memorie, delle CPU e dei bus. È giunto ora il momento di analizzare l'ultima tessera del puzzle, le interfacce di I/O; è attraverso queste porte che il calcolatore comunica con il mondo esterno.

### 3.7.1 Interfacce di I/O

Numerose interfacce di I/O sono già disponibili e con il passare del tempo ne vengono introdotte sempre di nuove. Chip molto comuni sono gli UART, gli USART, i controllori dei CRT, i controllori dei dischi e i PIO. Una **UART** (*Universal Asynchronous Receiver Transmitter*) è un'interfaccia che può leggere un byte da un bus di dati e generarlo in output, un bit alla volta, su una linea seriale per un terminale, oppure può ricevere input da un terminale. Generalmente le interfacce UART consentono velocità comprese tra 50 e 19.200 bps, dimensione dei caratteri variabili tra 5 e 8 bit, uno, uno e mezzo o due bit di stop e possono eventualmente fornire un controllo sulla parità pari o dispari; tutto questo avviene sotto il controllo del software. Le interfacce **USART** (*Universal Synchronous Asynchronous Receiver Transmitters*) possono gestire trasmissioni sincrone utilizzando vari protocolli, oltre a supportare tutte le funzionalità UART. Dato che le interfacce UART hanno perso importanza con la scomparsa dei modem telefonici, analizzeremo ora l'interfaccia parallela come esempio di chip di I/O.

#### Interfacce PIO

Un esempio tipico di interfaccia **PIO** (*Parallel Input/Output*), basata sul progetto originale Intel 8255A, è mostrato nella Figura 3.59. L'interfaccia è dotata di una collezione di linee di I/O (per esempio, nella figura ce ne sono 24) che possono collegare qualsiasi interfaccia di dispositivo digitale, per esempio tastiere, interruttori, luci, stampanti. In poche parole il programma della CPU può scrivere 0 o 1 su ogni linea, oppure può leggere in input lo stato di ogni linea, garantendo così un'elevata flessibilità. Un piccolo sistema basato su CPU che usa PIO può controllare diversi dispositivi fisici, come robot, tostapane o forni a microonde.

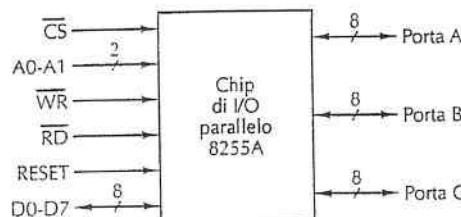


Figura 3.59 Interfaccia PIO a 24 bit.

L'interfaccia PIO viene tipicamente utilizzata nei sistemi integrati. L'interfaccia PIO viene configurata grazie a un registro di configurazione di 3 bit, che specifica se le tre porte indipendenti a 8 bit devono essere utilizzate per un segnale digitale in ingresso (0) o in uscita (1). Scrivendo il valore appropriato nel registro di configurazione è possibile impostare ogni configurazione di ingresso e uscita per le tre porte. A ciascuna porta è associato un registro latch a 8 bit. Per impostare le linee su una porta, la CPU scrive semplicemente un numero a 8 bit nel registro corrispondente e tale numero rimane sulla linea di output finché il registro non venga riscritto. Per utilizzare una porta in input, la CPU non fa che leggere il registro corrispondente.

È possibile costruire interfacce PIO più sofisticate. Per esempio, un utilizzo diffuso è per l'handshaking con dispositivi esterni. Per spedire un output verso un dispositivo che non è sempre pronto ad accettare i dati, l'interfaccia PIO può renderli disponibili su una porta di output e attendere che il dispositivo restituisca un segnale per indicare che ha accettato i dati e che ne desidera degli altri. La logica necessaria per memorizzare questi impulsi e renderli disponibili alla CPU include un segnale di ready e una coda di registri a 8 bit per ogni porta.

Dal diagramma funzionale del PIO possiamo vedere che oltre ai 24 pin necessari per le tre porte esso dispone di otto linee che lo connettono direttamente al bus dei dati, una linea per la selezione del chip, le linee per la lettura e la scrittura, due linee d'indirizzo e una linea per resettare il chip. Le due linee d'indirizzo selezionano uno dei quattro registri interni, corrispondenti alle porte A, B, C e al registro di configurazione. Di solito le due linee d'indirizzo sono connesse ai bit meno significativi del bus degli indirizzi. La linea CS permette di combinare più interfacce PIO a 24 bit per formare un'interfaccia PIO più grande, aggiungendo linee di indirizzo e utilizzandole per la selezione dell'interfaccia appropriata, individuata asserendo la corretta linea CS.

### 3.7.2 Decodifica dell'indirizzo

Finora siamo stati intenzionalmente vaghi circa il modo in cui la selezione del chip è assegnata sulla memoria e sui chip di I/O. Per vedere con maggior attenzione come ciò si svolge, consideriamo un semplice calcolatore integrato a 16 bit composto da una CPU, una EPROM per il programma da  $2\text{KB} \times 8$  byte, una RAM per i dati da  $2\text{KB} \times 8$  byte e un'interfaccia PIO.

Questo piccolo sistema potrebbe essere utilizzato come prototipo del cervello di un giocattolo economico o di un semplice elettrodomestico. Una volta entrata in fase produttiva la EPROM dovrebbe essere sostituita da una ROM.

L'interfaccia PIO può essere selezionata come un vero dispositivo di I/O o come parte della memoria. Se sceglieremo di usarla come un dispositivo di I/O, dobbiamo selezionarla usando una linea di bus che indica esplicitamente che ci si sta riferendo a un dispositivo di I/O e non a una memoria. Se sceglieremo il secondo approccio, quello dell'**I/O mappato in memoria**, dobbiamo assegnargli 4 byte dello spazio della memoria, necessari per indirizzare le tre porte e il registro di controllo. La scelta è praticamente arbitraria; noi sceglieremo l'I/O mappato in memoria in quanto mette in evidenza alcuni interessanti problemi riguardanti l'interfaccia di I/O.

Sia la EPROM sia la RAM richiedono uno spazio degli indirizzi di 2 KB, mentre il chip PIO richiede soltanto 4 byte. Dato che lo spazio degli indirizzi del nostro esempio è di 64 KB dobbiamo semplicemente scegliere dove collocare i tre dispositivi. La Figura 3.61 mostra una scelta possibile. La EPROM occupa gli indirizzi fino a 2 KB, la RAM occupa gli indirizzi compresi tra 32 e 34 KB, e il chip PIO occupa gli ultimi 4 byte dello spazio degli indirizzi, da 65532 a 65535. Dal punto di vista del programmatore lo spazio degli indirizzi scelto non fa alcuna differenza; dal punto di vista dell'interfaccia invece sì. Se avessimo scelto d'indirizzare il chip PIO mediante lo spazio di I/O non ci sarebbe stato bisogno di alcun indirizzo di memoria (ma sarebbero stati necessari quattro indirizzi dello spazio di I/O).

Usando l'assegnamento degli indirizzi della Figura 3.60 la EPROM potrebbe essere selezionata da uno qualsiasi degli indirizzi di memoria a 16 bit della forma 00000xxxxxxxxx (binario). In altre parole ogni indirizzo in cui i 5 bit più significativi sono 0 cade nei 2 KB bassi della memoria, cioè nella EPROM. La selezione del chip della EPROM potrebbe quindi essere collegata a un comparatore a 5 bit con input impostato al valore 00000.

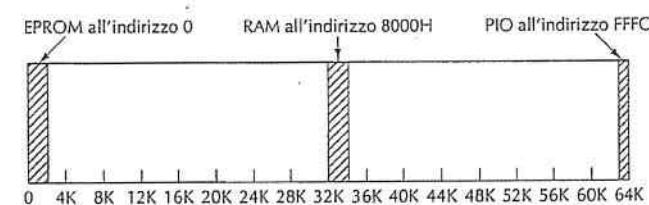


Figura 3.60 Posizione della EPROM, della RAM e del chip PIO nel nostro spazio d'indirizzamento di 64 KB.

Un modo migliore per ottenere lo stesso risultato consiste nell'usare una porta OR a cinque ingressi collegati alle linee d'indirizzo che vanno da A11 a A15. Se e solo se tutte le linee valgono 0 l'output varrà 0, asserendo di conseguenza il segnale CS (asserito con il valore basso). Questo approccio di indirizzamento è illustrato nella Figura 3.60(a) ed è chiamato decodifica piena degli indirizzi (*full-address decoding*).

Si può utilizzare lo stesso principio anche per la RAM, che però dovrebbe rispondere agli indirizzi binari della forma 10000xxxxxxxxxx. Quindi, come mostra la figura, è necessario un invertitore aggiuntivo. La decodifica degli indirizzi del chip PIO è invece leggermente più complicata, dato che il chip viene selezionato dai quattro indirizzi della forma 111111111111xx. Nella figura è mostrato un circuito che asserisce  $\bar{CS}$  soltanto quando sul bus degli indirizzi appare l'indirizzo corretto. Il circuito utilizza due porte NAND per alimentare una porta OR. Utilizzando SSI per costruire la logica di decodifica degli indirizzi della Figura 3.61(a) sono necessari sei chip: i quattro chip a otto input, una porta OR e un chip con tre invertitori.

Tuttavia, se il calcolatore è realmente composto soltanto dalla CPU, da due chip di memoria e dal chip PIO, possiamo ricorrere a un trucco che semplifica di gran lunga la decodifica degli indirizzi. Il trucco è basato sul fatto che tutti e soli gli indirizzi della EPROM hanno uno 0 nel bit più significativo, A15. Quindi, come mostra la Figura 3.61(b), possiamo semplicemente collegare  $\bar{CS}$  direttamente ad A15.

A questo punto la scelta di collocare la RAM a 8000H appare meno arbitraria. La decodifica della RAM può basarsi sull'osservazione che gli unici indirizzi validi della forma 10xxxxxxxxxxxx ricadono nella RAM; è quindi sufficiente decodificare 2 bit. In modo analogo ogni indirizzo che inizia con 11 deve per forza essere un indirizzo del chip PIO. La logica per la decodifica completa può quindi essere costituita solamente da due porte NAND e da un invertitore. Dato che un invertitore è realizzabile collegando i due ingressi di una porta NAND, tutto ciò di cui abbiamo bisogno è un chip con quattro porte NAND.

La logica di decodifica degli indirizzi della Figura 3.61(b) è chiamata **decodifica parziale dell'indirizzo**, dato che non utilizza l'indirizzo completo. Una proprietà di questo metodo è che la lettura degli indirizzi 0001000000000000, 0001100000000000 o 0010000000000000 fornirà lo stesso risultato. In realtà qualsiasi indirizzo della metà inferiore della memoria selezionerà la EPROM; ciò non provocherà alcun danno dato che gli indirizzi extra non saranno utilizzati. Se però si sta progettando un calcolatore che potrà in futuro essere espanso (cosa poco probabile nel caso di un giocattolo) bisognerebbe evitare la decodifica parziale, poiché vincola eccessivamente lo spazio degli indirizzi.

Un'altra comune tecnica d'indirizzamento consiste nell'usare un decodificatore, come quello mostrato nella Figura 3.13. Connnettendo i tre input alle tre linee dell'indirizzo più significative otteniamo otto output, corrispondenti agli indirizzi nei primi 8K, a quelli nei secondi 8K e così via. Per un calcolatore con otto RAM di 8K x 8, un simile chip fornisce una decodifica completa. Anche per un calcolatore con otto chip di memoria 2K x 8 è sufficiente un singolo decodificatore, fermo restando che i chip di memoria devono essere collocati in diversi settori da 8 KB dello spazio degli indirizzi. Ricordiamoci della precedente osservazione su quanto sia importante la posizione dei chip della memoria e dei chip di I/O all'interno dello spazio degli indirizzi.

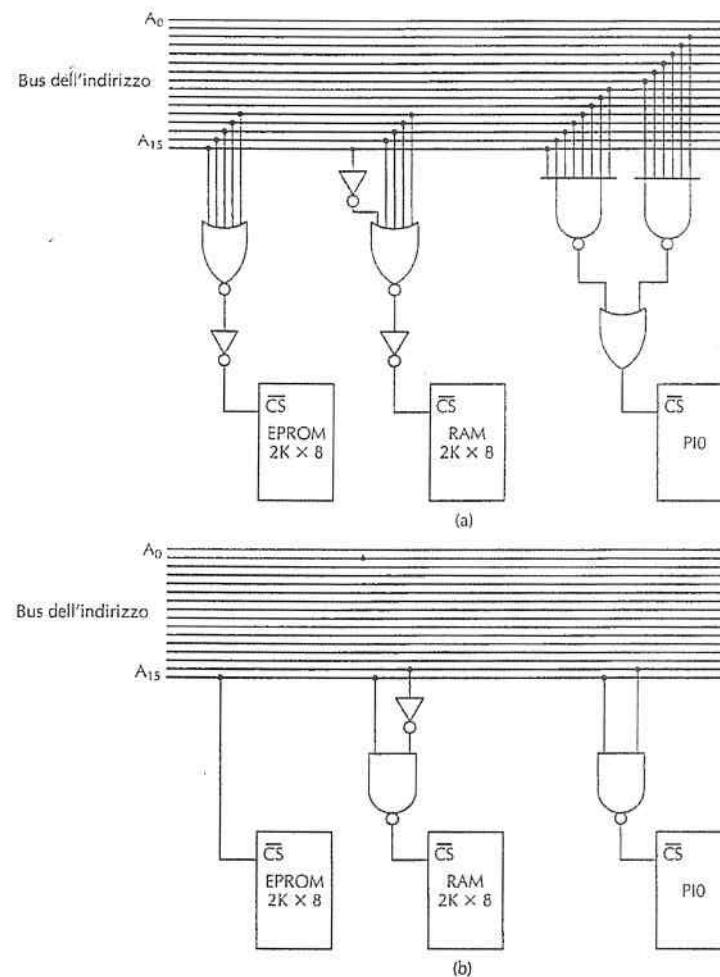


Figura 3.61 Decodifica dell'indirizzo: (a) completa; (b) parziale.

### 3.8 Riepilogo

I calcolatori sono costruiti a partire da chip di circuiti integrati contenenti piccoli elementi di commutazione chiamati porte logiche, tra cui le più comuni sono AND, OR, NAND, NOR e NOT. È possibile costruire semplici circuiti combinando direttamente le singole porte.

Alcuni esempi di circuiti sono i multiplexer, i demultiplexer, i codificatori, i decodificatori, gli shifter e le ALU. Utilizzando un PLA è possibile programmare funzioni booleane arbitrarie; se sono necessarie molte funzioni booleane spesso i PLA risultano ancora più efficienti. Utilizzando le leggi dell'algebra di Boole è possibile trasformare un circuito in un circuito equivalente dalla forma diversa; in molti casi usando queste tecniche è possibile produrre circuiti più economici.

L'aritmetica dei calcolatori è realizzata mediante sommatori. Per sommare una copia di bit si possono usare due semisommatori. Un sommatore per una parola composta da più bit può essere realizzato connettendo più sommatori in cascata.

I componenti delle memorie (statiche) sono i latch e i flip-flop, ciascuno dei quali può memorizzare un bit d'informazione. Possono essere combinati linearmente in latch e flip-flop aventi una qualsiasi dimensione di parola. Le memorie possono essere di vario tipo: RAM, ROM, PROM, EPROM, EEPROM e flash. Le memorie statiche non devono essere riaggiornate, dato che mantengono i valori memorizzati fintanto che ricevono alimentazione. Le memorie dinamiche, al contrario, devono essere aggiornate periodicamente per compensare la scarica dei piccoli condensatori che si trovano sul chip.

I componenti di un sistema di un calcolatore sono connessi fra loro mediante bus. Molti pin di un classico chip della CPU guidano direttamente una linea del bus. Le linee del bus possono essere divise in linee d'indirizzo, di dati e di controllo. I bus sincroni sono guidati da un clock principale, mentre i bus asincroni usano una strategia di handshaking per sincronizzare lo slave con il master.

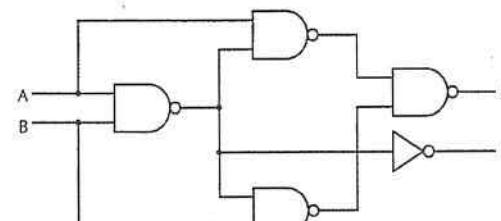
Il Core i7 è un esempio di moderna CPU. I sistemi che lo utilizzano hanno un bus di memoria, un bus PCIe e un bus USB. La connessione PCIe è lo strumento principale di collegamento ad alta velocità tra le parti interne di un computer. Anche l'ARM è una moderna CPU dalle elevate prestazioni, ma è pensata per i sistemi integrati e i dispositivi mobili, nei quali è importante il basso consumo di potenza. L'Atmel ATmega168 è un esempio di chip a basso costo adatto ad apparecchi piccoli ed economici e a molte altre applicazioni particolarmente sensibili ai costi.

Interruttori, luci, stampanti e molti altri dispositivi di I/O possono essere interfacciati utilizzando interfacce parallele di I/O. Secondo le necessità questi chip possono essere configurati per essere parte dello spazio di I/O o parte dello spazio di memoria. A seconda dell'applicazione possono essere decodificati parzialmente oppure in modo completo.

### PROBLEMI

1. I circuiti analogici sono soggetti a rumori che provocano distorsioni alle loro uscite. I circuiti digitali sono immuni al rumore? Si discuta la risposta.
2. Un professore di logica matematica entra in un bar e dice: "Voglio un cappuccino oppure un toast e una birra". Sfortunatamente il barista, essendo stato bocciato in quinta elementare, non sa (o non se ne preoccupa) se "e" abbia oppure no la precedenza su "o" ma, per quanto gliene possa interessare, un'interpretazione vale l'altra. Quali dei seguenti casi sono un'interpretazione corretta dell'ordine (si noti che qui la parola "oppure" significa "or esclusivo").  
 a. Solo un cappuccino.  
 b. Solo un toast.  
 c. Solo una birra.

- d. Un toast e una birra.  
 e. Un cappuccino e una birra.  
 f. Un toast e un cappuccino.  
 g. Tutti e tre.  
 h. Nulla – il professore non verrà servito visto che si è comportato da scontentone.
3. Un missionario che si è perso nella California del Sud si ferma a un bivio della strada. Sa che nella zona ci sono due bande di motociclisti, una delle quali dice sempre la verità, mentre l'altra mente sempre. Il missionario vuole sapere quale strada porta a Disneyland; quale domanda dovrebbe porre?
4. Si utilizzi una tabella di verità per mostrare la funzione  $X = (X \text{ AND } Y) \text{ OR } (X \text{ AND NOT } Y)$ .
5. Esistono quattro funzioni booleane di una singola variabile e 16 di due variabili. Quante funzioni di tre variabili esistono? E di  $n$  variabili?
6. Si mostri com'è possibile costruire la funzione AND mediante due porte NAND.
7. Utilizzando il multiplexer a tre vie della Figura 3.12 si implementi una funzione il cui output sia la parità degli input; l'output deve quindi essere 1 se e solo se un numero pari degli input vale 1.
8. Proviamo a ragionare: il multiplexer della Figura 3.12 è in grado di calcolare effettivamente un'arbitraria funzione booleana a quattro variabili. Si dimostri in che modo, si fornisca un esempio e si disegni il diagramma logico per la funzione che vale 0 se il numero della riga della tabella di verità si scrive con un numero pari di lettere, mentre vale 1 se le lettere sono dispari. Per esempio, 0000 = zero = 4 lettere  $\rightarrow$  0; 0111 = sette = 5 lettere  $\rightarrow$  1; 1101 = tredici = 7 lettere  $\rightarrow$  1. Suggerimento: se chiamiamo D la quarta variabile di input possiamo collegare le otto linee di input a  $V_{cc}$  alla terra, a D oppure a D.
9. Si disegni il diagramma logico di un codificatore a 2 bit; il circuito ha quattro linee di input di cui una soltanto ha valore alto in un dato istante e due linee di output il cui valore binario a 2 bit indica quale input è alto.
10. Si disegni il diagramma logico di un demultiplexer a 2 bit; il circuito ha una singola linea di input che è indirizzata verso una delle quattro linee di output in base allo stato delle due linee di controllo.
11. Si ridisegni, con sufficiente dettaglio, il PLA della Figura 3.15, per mostrare com'è possibile implementare la maggior parte delle funzioni logiche della Figura 3.3. In particolare si presti attenzione a mostrare quali connessioni sono presenti in entrambe le matrici.
12. Che cosa fa questo circuito?



13. Un chip MSI molto comune è il sommatore a 4 bit. È possibile agganciare quattro di questi chip per formare un sommatore a 16 bit; quanti pin ci si aspetta che avrà il sommatore? Perché?
14. Un sommatore a  $n$  bit può essere costruito collegando in cascata  $n$  sommatori in serie, in cui il riporto  $C_i$  dello stadio  $i$  giunge dall'output dello stadio  $i - 1$ . Il riporto  $C_0$  nello stadio 0 vale 0. Dato che ciascuno stadio richiede  $T$  ns per produrre la propria somma e il proprio riporto, il riporto allo stadio  $i$  sarà valido solo  $iT$  ns dopo l'inizio della somma. Per valori di  $n$  elevati il tempo necessario affinché il riporto giunga fino all'ultimo stadio po-

- trebbe essere eccessivamente lungo. Si progetti un sommatore che funzioni più velocemente. Consiglio: ciascun  $C_i$  può essere espresso in termini dei bit dell'operando  $A_{hi}$  e  $B_{hi}$  e del riporto  $C_{hi}$ . Utilizzando questa relazione è possibile esprimere  $C_i$  come una funzione degli input agli stadi compresi tra 0 e  $i - 1$ ; è quindi possibile generare simultaneamente tutti i riporti.
15. Si assuma che nella Figura 3.19 tutte le porte logiche abbiano un ritardo di propagazione di 1 ns e che sia possibile ignorare tutti gli altri ritardi. Qual è il tempo minimo al quale si ha la certezza che un circuito progettato in questo modo genererà un output valido?
  16. La ALU della Figura 3.20 è in grado di eseguire somme a 8 bit in complemento a due. È in grado di eseguire anche le sottrazioni in complemento a due? In caso affermativo si spieghi in che modo. In caso negativo si modifichi il circuito in modo che sia in grado di compiere le sottrazioni.
  17. Una ALU a 16 bit è costruita mediante 16 ALU a 1 bit, ciascuna delle quali richiede 10 ns per compiere una somma. Se è presente un ulteriore ritardo di 1 ns dovuto alla propagazione tra una ALU e la successiva, dopo quanto tempo appare il risultato di una somma a 16 bit?
  18. Talvolta per una ALU a 8 bit come quella della Figura 3.20 è utile generare in output il valore costante  $-1$ . Si forniscano due modi diversi per raggiungere questo scopo. Per ciascun modo si specifichino i valori dei sei segnali di controllo.
  19. Che cos'è lo stato di riposo degli input R e S di un latch SR costruito mediante due porte NAND?
  20. Il circuito della Figura 3.26 è un flip-flop pilotato dal fronte di salita del clock. Si modifichi questo circuito per ottenerne un flip-flop che sia pilotato dal fronte di discesa del clock.
  21. La memoria  $4 \times 3$  della Figura 3.29 utilizza 22 porte AND e tre porte OR. Quante di queste porte sarebbero necessarie se si espandesse il circuito strutturandolo come  $256 \times 8$ ?
  22. Per poter pagare il tuo nuovo personal computer hai deciso di dare consulenze a produttori di chip SSI con poca esperienza. Su richiesta di un'importante cliente uno di questi produttori sta pensando di costruire un chip composto da quattro flip-flop D, ciascuno contenente sia Q sia  $\bar{Q}$ . Il progetto proposto prevede inoltre che tutti e quattro i segnali di clock siano legati fra loro, mentre non sono presenti né i segnali di preset né di clear. Si dia un parere professionale sul progetto.
  23. Man mano che in un singolo chip la dimensione della memoria viene ridotta, aumenta il numero di pin richiesti per indirizzarla. Spesso non è conveniente avere un grande numero di pin dell'indirizzo su un singolo chip. Si pensi a un metodo per indirizzare  $2^n$  parole di memoria utilizzando meno di  $n$  pin.
  24. Un calcolatore con un bus di dati largo 32 bit utilizza una RAM dinamica implementata con un chip 1 Mbit  $\times 1$ . Qual è la più piccola memoria (in byte) che questo calcolatore può avere?
  25. Facendo riferimento al diagramma di temporizzazione della Figura 3.38 si supponga di rallentare il clock da 10 ns (com'è mostrato nella figura) a 20 ns, senza modificare i vincoli di temporizzazione. Nel caso peggiore quanto tempo impiega la memoria per prelevare i dati dal bus durante  $T_d$ , dopo che MREQ è stato asserito?
  26. Facendo nuovamente riferimento alla Figura 3.38 si supponga di lasciare il clock a 100 MHz, ma di aumentare  $T_{os}$  fino a 4 ns. Sarebbe possibile utilizzare chip di memoria da 10 ns?
  27. Nella Figura 3.38(b) si specifica che TML deve essere di almeno 3 ns. È possibile immaginare un chip in cui ciò è negativo? In particolare, la CPU potrebbe asserire MREQ prima che l'indirizzo diventi stabile? Se sì, perché?
  28. Si assume che il trasferimento a blocchi della Figura 3.42 venga effettuato sul bus della Figura 3.38. Nel caso di lunghi blocchi di dati quanta larghezza di banda si guadagna usando un trasferimento a blocchi invece che più trasferimenti singoli? Si assume successivamente che il bus abbia una larghezza di 32 bit invece che di 8 bit e si risponda nuovamente alla domanda.

29. Si indichino con  $T_{A1}$  e  $T_{A2}$  i tempi di transizione delle linee d'indirizzo della Figura 3.39, con  $T_{MREQ}$  e  $T_{MREQ2}$  i tempi di transizione di MREQ e così via. Si scrivano tutte le disuguaglianze determinate dal protocollo di full handshaking.
30. I chip multicore, con più CPU sullo stesso chip, sono diventati molto diffusi. Che vantaggi hanno rispetto a un sistema costituito da diversi PC collegati via Ethernet?
31. Perché sono improvvisamente apparsi sul mercato i chip multicore? Ci sono fattori tecnologici che hanno tracciato il cammino? La legge di Moore ha giocato qualche ruolo?
32. Qual è la differenza tra un bus di memoria e un bus PCI?
33. La maggior parte dei bus a 32 bit permette letture e scritture a 16 bit. Esiste qualche ambiguità sulla posizione in cui vengono posti i dati? Si argomenti la risposta.
34. Molte CPU hanno un tipo speciale di ciclo di bus per la conferma degli interrupt. Perché?
35. Un calcolatore a 64 bit con un bus a 400 MHz richiede quattro cicli per leggere una parola a 64 bit. Quanta larghezza di banda consuma la memoria nel caso peggiore, ovvero assumendo che ogni volta vi siano riletture e riscritture consecutive?
36. Un calcolatore a 64 bit con un bus a 400 MHz richiede quattro cicli per leggere una parola a 64 bit. Quanta larghezza di banda consuma la memoria nel caso peggiore, ovvero assumendo ogni volta continue riletture e riscritture consecutive?
37. Una CPU a 32 bit con linee d'indirizzo A2-A31 richiede che tutti i riferimenti alla memoria siano allineati; le parole devono quindi essere indirizzate a multipli di 4 byte e le mezze-parole devono essere indirizzate in corrispondenza dei byte pari. I byte possono essere ovunque. Quante sono le combinazioni legali per le letture della memoria e quanti pin sono necessari per esprimere? Si diano due risposte e si presenti un esempio per ciascuna.
38. Le CPU moderne hanno uno, due o anche tre livelli di cache sul chip. Perché sono necessari più livelli di cache?
39. Si supponga che una CPU abbia una cache di primo livello e una di secondo livello, con tempi di accesso rispettivamente di 1 ns e 2 ns. Il tempo di accesso alla memoria centrale è di 10 ns. Se hanno successo il 20% degli accessi alla cache di primo livello e il 60% degli accessi alla cache di secondo livello, qual è il tempo medio di accesso?
40. Si calcoli la larghezza di banda necessaria per visualizzare un filmato 1280 x 960 a colori a 30 fotogrammi al secondo e in true-color. Si assuma che i dati debbano passare sul bus due volte, la prima dal CD-ROM alla memoria e la seconda dalla memoria allo schermo.
41. Quali dei segnali della Figura 3.56 non sono strettamente necessari affinché il protocollo del bus funzioni?
42. Un sistema basato su PCI Express ha collegamenti a 10 Mbps (capacità lorda). Per funzionare a 16x quanti collegamenti sono necessari in ciascuna direzione? Qual è la capacità lorda in ciascuna direzione? Quale la capacità netta?
43. Tutte le istruzioni di un calcolatore richiedono due cicli di bus, uno per prelevare l'istruzione e uno per prelevare i dati. Ciascun ciclo di bus richiede 10 ns e ciascuna istruzione 20 ns (cioè l'elaborazione interna è trascurabile). Il calcolatore ha un disco con 2048 settori, 512 byte per traccia e una rotazione di 5 ms. Rispetto alla normalità a quale percentuale si riduce la velocità del calcolatore durante un trasferimento DMA a 32 bit, considerando che ciascuno richiede un ciclo di bus?
44. Su un bus USB la massima dimensione del campo dati di un pacchetto dati isocrono è di 1023 byte. Assumendo che un dispositivo possa spedire solo un pacchetto dati per frame, qual è la massima larghezza di banda per un singolo dispositivo isocrono?