CAPITOLO X
SISTEMI DIGITALI A LARGA SCALA

10.1) Introduzione.

Calcolatori digitali e sistemi digitali di grandi dimensioni sono chiari esempi di circuiti digitali, tuttavia differiscono da quelli fin qui esaminati per la loro complessita', notevolmente maggiore. I componenti principali, escludendo le memorie, della maggior parte dei computer e dei sistemi periferici sono registri composti da numerosi flip-flop e il numero di stati puo' raggiungere e superare il centinaio. E' evidente quindi che il grafo e la tabella degli stati non sono un mezzo pratico per la descrizione dei sistemi a larga scala. L'intento del presente capitolo e' quello di sviluppare una tecnica piu' adatta alla rappresentazione dei sistemi a larga scala e poiche' la massima parte d'essi e' del tipo "clock mode" ci si limitera' ad esaminare solo questi ultimi. Un primo passo che puo' essere compiuto in questa direzione consiste nel modificare il modello fondamentale, spezzandolo in due circuiti sequenziali separati, come illustrato in fig. 10.1.1.

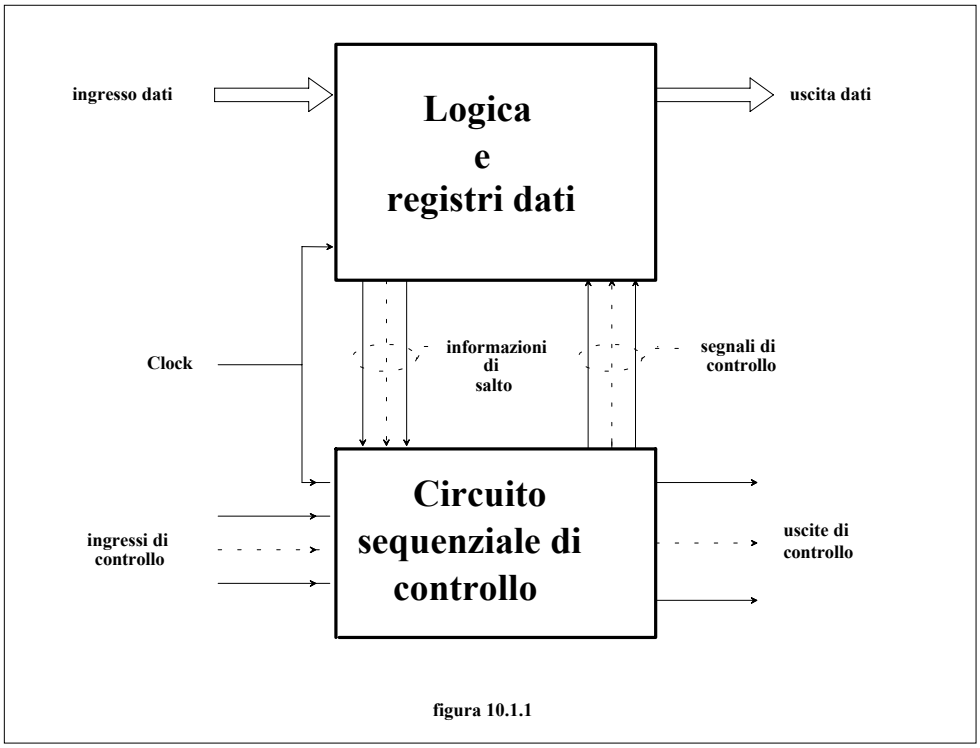

figura 10.1.1

Il blocco superiore consiste in un insieme di registri di dati interconnessi da una logica combinatoria, mentre il blocco inferiore fornisce una sequenza di segnali di controllo, che gestiscono, in unione al clock, l'appropriato trasferimento dei dati tra i registri.

Il numero di segnali e di ingressi di controllo e' normalmente piccolo rispetto al numero di linee di ingresso e di uscita dei dati e delle linee di interconnessione interne del blocco superiore.

Gli scopi e i vantaggi di questo modello non sono tuttavia a questo punto chiari. Essi verranno resi evidenti nei successivi paragrafi; ci si limitera' per il momento a esaminare in dettaglio il modello proposto, attraverso un certo numero di esempi.

10.2) Ingresso di controllo del clock.

In tutte le discussioni precedenti i circuiti sequenziali funzionanti secondo la modalita' "clock mode" prevedevano il collegamento diretto del segnale di clock a tutti i flip-flop, come e' illustrato in fig. 10.2.1 (a). In certe circostanze tuttavia, in particolare quando si usino flip-flop di tipo D, tale approccio puo' risultare restrittivo.

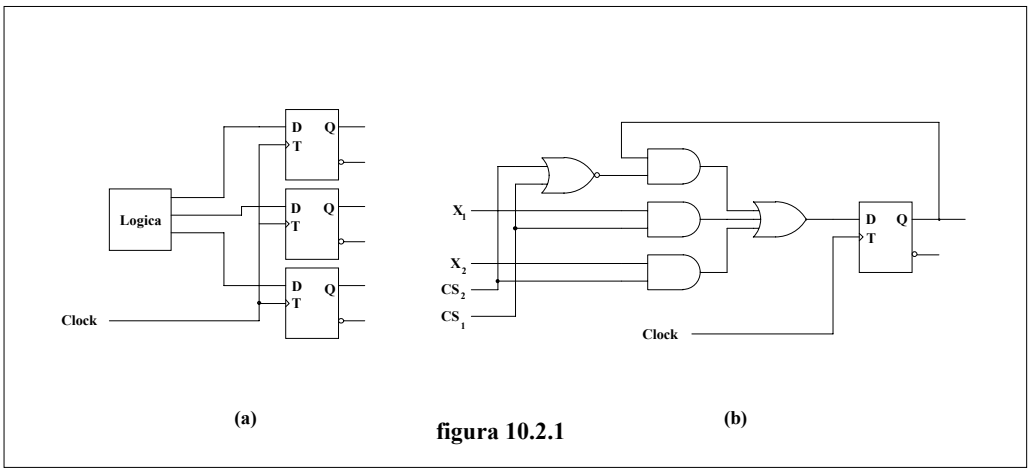

figura 10.2.1

In fig. 10.2.1 (b) e' riportato un circuito sequenziale con collegamento diretto del clock. Se il segnale di controllo CS_1 e' pari a 1, X_1 viene memorizzato nel flip-flop, mentre se al valore logico 1 si trova il segnale CS_2 nel flip-flop viene memorizzato il segnale X_2. E' necessario che il valore da memorizzare sia presente all'ingresso del flip-flop prima che giunga il segnale di clock. Di conseguenza, per mantenere l'informazione quando ambedue i segnali di controllo sono nulli, il valore corrente memorizzato nel flip-flop deve essere riportato all'ingresso D; tale operazione viene realizzata con un ulteriore gate AND e un NOR.

La rete di fig. 10.2.1 (b) puo' essere semplificata rinunciando a collegare il clock direttamente al flip-flop. Si supponga infatti di permettere che il segnale di clock raggiunga l'ingresso di sincronizzazione del flip-flop solamente se un nuovo dato deve venir memorizzato. A questo scopo e' sufficiente abilitare il segnale di clock con l'OR dei segnali CS_1 e CS_2, come illustrato in fig. 10.2.2 (a).

Il risparmio ottenuto consiste unicamente in un invertitore, ma con sistemi piu' complessi, in particolare quando piu' flip-flop sono combinati per realizzare un registro, i risultati possono essere piu' soddisfacenti. Infatti in tal caso lo stesso segnale di controllo viene utilizzato per introdurre i dati nei diversi flip-flop del registro e la rete di clock puo' essere comune a tutti i flip-flop, come illustrato in fig. 10.2.2 (b). Con l'approccio di fig. 10.2.1 (b) non si puo' invece conseguire alcun risparmio.

E' opportuno tuttavia far notare che l'usare segnali di controllo per abilitare il clock non altera il tipo di funzionamento che rimane sempre "clock mode". Infatti, anche se alcuni flip-flop sono controllati in modo da essere comandati solo in determinate situazioni, rimane pur sempre il fatto che ciascun flip-flop puo' cambiare stato solo in corrispondenza all'impulso di clock e quindi le modalita' di funzionamento non vengono alterate.

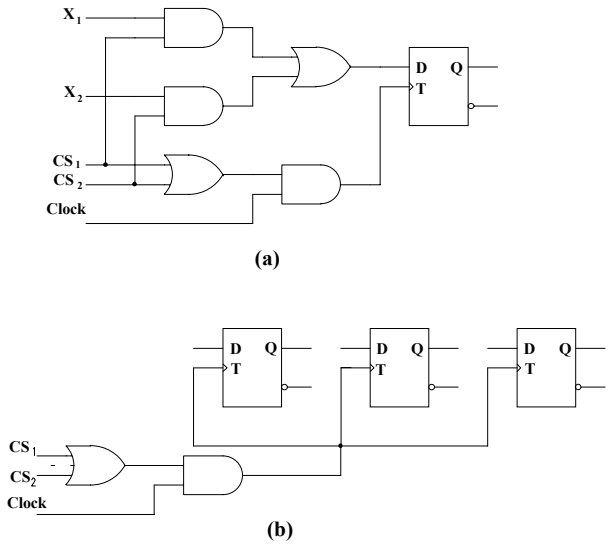

figura 10.2.2

10.3) Estensione della tavola di stato.

In fig. 10.3.1 sono mostrati tre flip-flop, B_0, B_1, B_2 interconnessi da una linea di trasferimento dei dati controllata dai segnali C_1, C_2, C_3 e C_4 generati a loro volta da un opportuno circuito di controllo sequenziale. La natura del circuito di controllo e' tale che solamente uno dei quattro segnali C_i puo' assumere il valore logico 1 a un determinato istante.

Un esame piu' accurato del circuito rivela che vi sono quattro modalita' di trasferimento dei dati, che possono aver luogo nel blocco inferiore di fig. 10.3.1. Se C_1=1 allora i segnali di ingresso X_0 e X_1 vengono trasferiti nei flip-flop B_0 e B_1 rispettivamente; la notazione normalmente usata per identificare tale trasferimento e':

$$B_0 \leftarrow X_0 \qquad B_1 \leftarrow X_1 \qquad (10.3.1)$$

Se C_2=1 l'informazione contenuta nei tre flip-flop viene spostata verso destra di una posizione. Il contenuto del flip- flop all'estrema destra viene ruotato in B_0.

$$B_1 \leftarrow B_0 \qquad B_2 \leftarrow B_1 \qquad B_0 \leftarrow B_2 \qquad (10.3.2)$$

Se C_3=1 il trasferimento e':

$$B_1 \leftarrow B_0 . B_1 \qquad (10.3.3)$$

mentre se C_4=1 si ha:

$$B_2 \leftarrow B_1 \oplus B_2 \qquad (10.3.4)$$

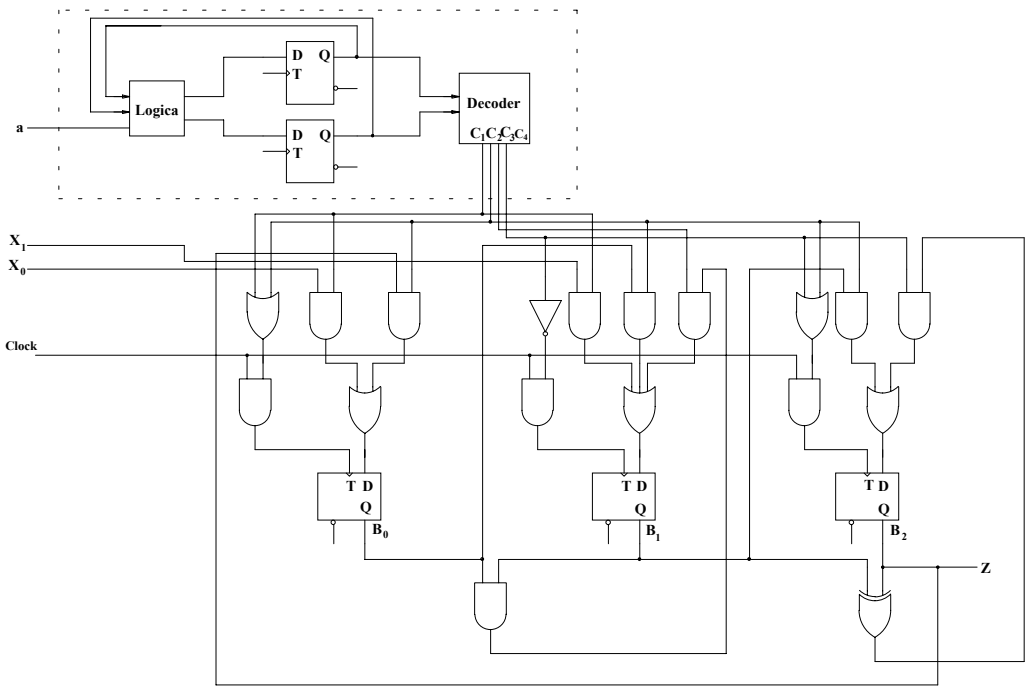

figura 10.3.1

Quanto illustrato descrive i trasferimenti che possono aver luogo, ma la sequenza dei trasferimenti e quindi la sequenza delle uscite dipende dalla sequenza dei segnali di controllo. Poiche' vi sono solo quattro possibili combinazioni di questi segnali, si puo' caratterizzare il circuito sequenziale di controllo come circuito di Moore con quattro stati, indicabili con C_1, C_2, C_3 e C_4, in corrispondenza a ciascuno dei quali si ha un 1 logico sulla corrispondente uscita.

Si supponga inoltre che la sequenza con cui questi stati si presentano sia comandata da un segnale **a** di ingresso di controllo della sequenza, come illustrato in fig. 10.3.2.

Stato \ a	0	1	Trasferimento
C_1	C_3	C_2	$B_0 \leftarrow X_0 ; B_1 \leftarrow X_1$
C_2	C_1	C_1	$B_1 \leftarrow B_0 ; B_2 \leftarrow B_1 ; B_0 \leftarrow B_2$
C_3	C_4	C_4	$B_1 \leftarrow B_0 . B_1$
C_4	C_2	C_1	$B_2 \leftarrow B_1 \oplus B_2$

figura 10.3.2

Tale assunzione e' coerente con il circuito di fig. 10.3.1, in cui **a** e' il segnale di ingresso del circuito sequenziale di controllo a quattro stati.

Si aggiunga a questo punto alla tradizionale tavola di stato un'ulteriore colonna contenente l'indicazione dei trasferimenti che hanno luogo in corrispondenza a ciascun stato. Assieme all'equazione di uscita:

$$z = B_2$$

questa tabella di stato estesa fornisce una completa descrizione del comportamento desiderato per il circuito sequenziale. Si deduce infatti da tale descrizione che vi sono tre flip-flop di dati e che sono necessari almeno due flip-flop di stato. Al contrario le tradizionali descrizioni viste ai capitoli precedenti richiederebbero una tabella con 32 stati, chiaramente meno conveniente di quella di fig. 10.3.2.

10.4) Descrizione a programma.

Sebbene la tabella di fig. 10.3.2 sia una conveniente descrizione del corrispondente circuito, l'utilizzarla per circuiti con un gran numero di ingressi e di stati di controllo darebbe comunque luogo ad una rappresentazione poco maneggevole. Scopo del presente paragrafo e' quello di sviluppare una descrizione dei circuiti sequenziali ancora piu' compatta, che abbia un aspetto formale simile a quello di un programma per calcolatore.

La lista dei trasferimenti di fig. 10.3.2 somiglia infatti alle istruzioni di un programma; i flip-flop corrispondono alla variabili scalari e i nuovi valori di queste variabili sono le espressioni sulla destra di ciascuna freccia di trasferimento. Per convenzione si assumera' che i trasferimenti descritti nella stessa riga avvengano simultaneamente. Nel seguito verra' poi introdotta una notazione vettoriale per descrivere in modo piu' efficiente i trasferimenti simultanei. Le informazioni di salto ovviamente non sono comprese nella lista dei trasferimenti, ma nella tavola di stato. Ad esempio, a partire dallo stato C_1 o dallo stato C_4 viene eseguito un salto condizionato dal valore della variabile **a**. Se invece si considera lo stato C_2 il salto avviene sempre verso C_1 senza alcuna condizione.

Per ottenere il voluto tipo di descrizione e' sufficiente adottare un'opportuna notazione che permetta di trasformare la lista dei trasferimenti in un completo programma di descrizione del circuito. Si puo' pensare, per descrivere i salti, di utilizzare una notazione simile a quella dell'APL, inserendo dopo ciascuna istruzione di trasferimento un'istruzione di salto. Ciascun passo del programma sara' numerato con il numero rappresentativo del relativo stato. L'unico caso in cui l'istruzione di salto potra' essere omessa sara' quando il controllo dovra' portarsi comunque verso il successivo passo numerato.

La forma generale di un'istruzione di salto sara':

$$\rightarrow \left((f_1 . s_1) + (f_2 . s_2) + ... + (f_n . s_n) \right) \qquad (10.4.1)$$

dove s_i sono i numeri dei passi del programma e f_i sono funzioni booleane mutuamente esclusive e di cui almeno una deve assumere istante per istante il valore logico 1. Esse devono quindi soddisfare alle condizioni:

$$f_i . f_j = 0 \qquad \text{per } i \neq j$$
$$f_1 + f_2 + ... + f_n = 1$$

per tutte le combinazioni delle variabili che non sono condizioni d.c.c.

Di conseguenza una e una sola delle funzioni f_i sara' a 1 ciascun istante e il risultato aritmetico della (10.4.1) individuera' il numero di passo di programma cui saltare.

Seguendo le convenzioni descritte, la tavola di flusso 10.3.2 si trasforma nel programma:

1. $B_0 \leftarrow X_0; B_1 \leftarrow X_1$
$\rightarrow \left(\left(\bar{a}.3\right)+\left(a.2\right)\right)$

2. $B_1 \leftarrow B_0; B_2 \leftarrow B_1; B_0 \leftarrow B_2$
$\rightarrow (1)$

3. $B_1 \leftarrow B_0 . B_1$

4. $B_2 \leftarrow B_1 \oplus B_2$
$\rightarrow \left(\left(\bar{a}.2\right)+\left(a.1\right)\right)$

In tutto il resto del presente capitolo si usera' il termine "**sequenza di controllo**" per identificare la descrizione di un circuito nella forma appena introdotta.

Nel caso che si sta esaminando si e' ottenuta una descrizione completa del circuito di fig. 10.3.1, contenente tutte le informazioni della tavola di flusso estesa di fig. 10.3.2. Tuttavia l'efficienza della descrizione a programma non appare chiaramente dall'esempio scelto. Si noti infatti che una sola delle istruzioni di salto, quella al passo 3., e' stata omessa; nei sistemi di maggiori dimensioni questa situazione e' molto piu' frequente. Inoltre, se anziche' un unico ingresso si ha un gran numero di ingressi di controllo, accade normalmente che le f_i di un determinato passo siano funzione solamente di alcuni di questi ingressi.

10.5) La sintesi.

Uno dei principali vantaggi della descrizione a programma di un circuito e' che essa puo' venir usata come strumento di sintesi. Anzi, per sistemi a larga scala, e' l'unico approccio che permette di affrontare il discorso del progetto e che va al di la' dei metodi empirici che ricorrono alla manipolazione a tentativi dei blocchi logici.

Nel resto del presente capitolo verra' mostrato come la descrizione a programma possa essere utilmente impiegata per descrivere un certo numero di sistemi troppo complessi per essere descritti con una tavola di stato.

Si ricordi tuttavia che il nuovo strumento messo a punto e' **un linguaggio e solamente un linguaggio e non possiede una struttura algebrica** sulla base della quale condurre una minimizzazione degli stati. D'altra parte la struttura stessa del linguaggio guida il progettista ad una buona realizzazione, anche se non minima. C'e' inoltre da osservare che la disponibilita' di circuiti integrati complessi a basso costo rende relativamente poco importante la minimizzazione, anche se quest'ultima osservazione va presa necessariamente con le dovute cautele.

ESEMPIO 1

Si voglia ottenere la sequenza di controllo di un circuito seriale di conversione di codice illustrata nella seguente tavola di verita'.

$x^n\, x^{n-1}\, x^{n-2}$	$f_2\ f_1\ f_0$ $z^{n+2}\, z^{n+1}\, z^n$
000	**000**
001	**011**
010	**100**
011	**101**
100	**110**
101	**001**
110	**010**
111	**111**

Come indicato, vi e' una linea di ingresso x e una di uscita z. Alla linea di ingresso si presentano parole di tre bit e ciascun carattere e' seguito immediatamente da un altro. Vi e' poi un ulteriore ingresso r di reset che, quando assume il valore logico 1, sincronizza le operazioni, dando inizio al ciclo operativo. Il carattere di uscita tradotto sara' ritardato di due impulsi di clock rispetto a quello di ingresso.

Soluzione

La traduzione non puo' aver luogo finche' tutti i tre bit di ingresso non siano stati ricevuti all'istante n. Di conseguenza i due primi bit di ciascuna parola devono venir accumulati in uno shift register formato dai flip-flop B_0 e B_1. All'istante n il bit x^{n-2} si trovera' in B_1 e il bit x^{n-1} in B_0. Quando giunge x^n tutti i tre bit vengono elaborati simultaneamente, dando luogo all'uscita immediata z^n . Gli altri due bit z^{n+1} e z^{n+2} verranno accumulati in B_1 e B_0 rispettivamente, poiche' l'informazione sui valori dell'ingresso nei due precedenti periodi di clock non serve piu'. All'istante n+1 $B1=z^{n+1}$ verra' presentato in uscita, $B_0=z^{n+2}$ sara' trasferito in B_1 e il primo bit dal successivo carattere di ingresso verra' accumulato in B_0. All'istante n+2 infine z^{n+2} verra' trasferito in uscita, il precedente bit di ingresso verra' trasferito da B_0 a B_1 e il nuovo bit verra' caricato in B_0, in modo che all'ingresso ancora successivo la sequenza possa riprendere. Le operazioni del circuito possono venir descritte da una sequenza ripetitiva di tre istruzioni, di cui le prime due comandano la memorizzazione nello shift register e la terza la conversione di codice e il caricamento parallelo del registro con i nuovi valori.

1. $B_0, B_1 \leftarrow x, B_0;$ $\qquad z = B_1$
$\qquad \rightarrow ((r.1)+(\bar{r}.2))$

2. $B_0, B_1 \leftarrow x, B_0;$ $\qquad z = B_1$
$\qquad \rightarrow ((r.1)+(\bar{r}.3))$

3. $B_0, B_1 \leftarrow f_2(x, B_0, B_1), f_1(x, B_0, B_1);$ $\qquad z = f_0(x, B_0, B_1)$
$\qquad \rightarrow (1)$

Le tre funzioni di uscita del passo 3. possono essere ottenute convertendo la tabella di funzionamento assegnata precedentemente nelle tre mappe di Karnaugh di fig. 10.5.1.

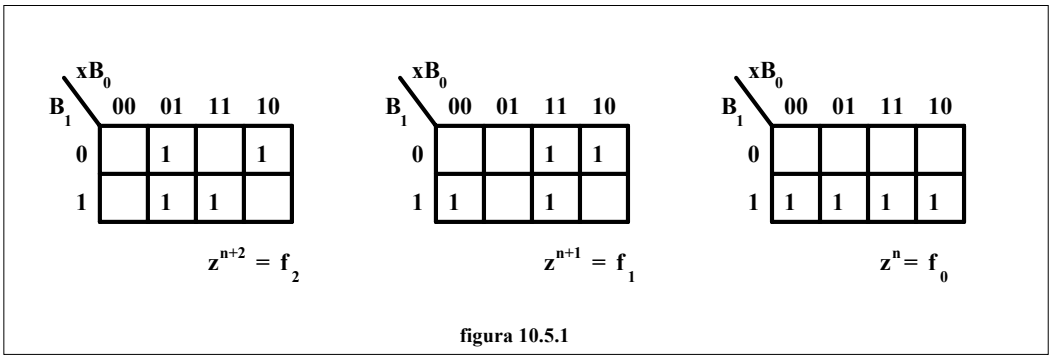

figura 10.5.1

Le funzioni risultanti sono:

$$f_0 = B_1$$

$$f_1 = x^n . B_0 + x^n . \overline{B_1} + \overline{x^n} . \overline{B_0} . B_1$$

$$f_2 = \overline{x^n} . B_0 + B_0 . B_1 + x^n . \overline{B_0} . \overline{B_1}$$

Il meccanismo di reset e' stato incorporato nel programma di tre istruzioni. Quando l'ingresso r = 1 il controllo viene restituito al passo 1 del programma, riinizializzando le operazioni con l'ingresso di un nuovo carattere.

Si noti che nel programma di descrizione l'uscita z e' specificata a ciascun passo. Cio' si rende necessario in quanto l'uscita non e' una funzione fissa dei dati e/o dei segnali di ingresso come nel caso del circuito di fig. 10.3.1. Essa e' invece una funzione differente per ciascun passo del programma ed e' quindi funzione anche dei segnali di controllo.

L'esempio considerato rende evidenti le differenze tra il metodo a programma di descrizione e il convenzionale metodo a grafo di stato. Costruire il diagramma di stato richiede unicamente la comprensione delle volute sequenze di ingresso e di uscita, mentre non e' necessaria alcuna ipotesi sul numero di variabili di stato o sulla struttura interna del circuito. Al contrario l'approccio attraverso il programma di descrizione prende le mosse proprio dall'aver ipotizzato una struttura base e pertanto richiede una notevole capacita' intuitiva nell'ideare una possibile configurazione per il circuito. Il risultato comunque ben difficilmente sara' minimo nel senso esposto ai capitoli precedenti.

Una volta che si sia ottenuta la sequenza di controllo la parte creativa del progetto e' completata. La traduzione della sequenza di controllo in circuito logico richiede unicamente il rigoroso rispetto di alcune semplici regole e l'operazione potrebbe addirittura essere automatizzata e normalmente lo e'.

E' tuttavia opportuno, anche per maggior chiarezza nei confronti del lettore, seguire passo a passo la realizzazione circuitale.

E' conveniente sviluppare separatamente la **parte di controllo** e la **parte di memorizzazione dei dati e di logica**. Quest'ultima e' illustrata in fig. 10.5.2.

Si noti che i passi 1 e 2 del programma di descrizione sono identici e quindi sara' sufficiente un'unica linea di controllo per eseguirli ambedue. Una seconda linea di controllo sara' necessaria per eseguire il terzo passo. Inoltre, poiche' il contenuto dei flip-flop viene

modificato ad ogni passo, la linea di clock va connessa senza interposizione di alcun circuito agli ingressi C di ciascun flip-flop.

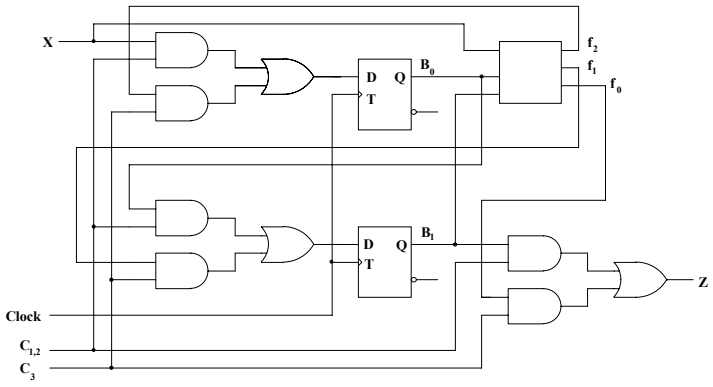

figura 10.5.2

Normalmente l'uscita z del circuito, se dipende dai segnali di controllo, verra' realizzata con un gate OR. Ad ogni passo, cioe' ad ogni impulso di clock, ad un ingresso dell'OR verra' riportato il valore da trasferire in uscita, mentre nel contempo gli altri ingressi dovranno esser posti a zero. Pertanto ciascun segnale di controllo del passo ed il relativo valore di uscita verranno portati all'ingresso di un AND della rete logica a due livelli che realizza z.

Reti logiche AND-OR a due livelli, del tutto identiche a quella di uscita, vengono poste in ingresso ai due flip-flop in modo che airispettivi ingressi D vengano riportati i valori corrispondenti al passo in corso.

Rimane da realizzare il circuito che sintetizza i segnali di controllo $C_{1,2}$ e C_3. Il primo passo e' quello di ottenere il diagramma di stato per il controllore di sequenza; ciascun passo del programma di descrizione corrispondera' ad uno stato, mentre i rami del grafo andranno ricavati dalle istruzioni di salto. Si ottiene in tal modo il diagramma di fig. 10.5.3.

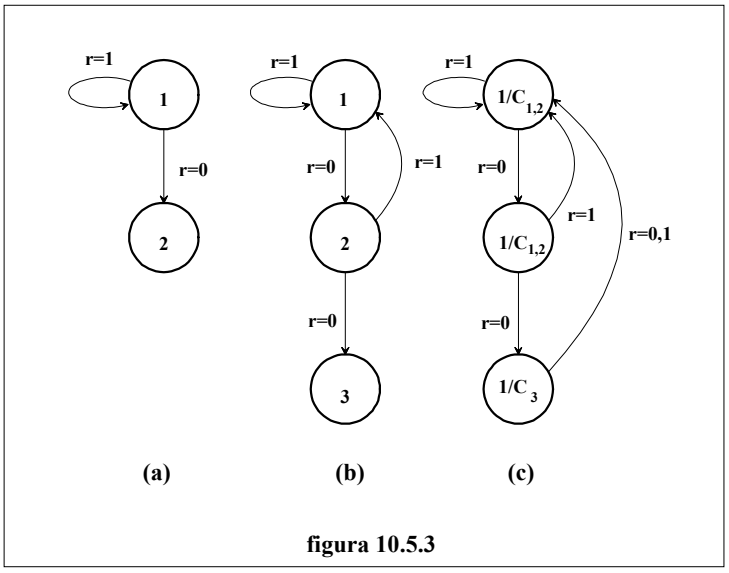

figura 10.5.3

Il circuito di controllo sara' pertanto un circuito di Moore che fornira' un'uscita a livelli per tutta la durata di ciascun passo. Si noti che il circuito ha due uscite di cui solamente una puo' essere diversa da 0 a ciascun istante. Per semplicita' di notazione a ciascun stato e' stata associata solo l'uscita che in corrispondenza a quello stato e' diversa da 0.

Una volta che il diagramma di stato sia stato ottenuto, la sintesi del circuito di controllo puo' essere condotta con gli usuali metodi visti ai capitoli precedenti. Si puo' facilmente verificare che il diagramma di fig. 10.5.3 e' gia' minimo. La codifica ottenuta applicando le regole 1 e 2 del paragrafo 7.7 e' riportata in fig. 10.5.4.

$y_2 y_1$ \ r	0	1
00	01/10	00/10
01	11/10	00/10
11	00/01	00/01
10	---/---	---/---

stato futuro , $C_{1,2}$, C_3

figura 10.5.4

e da' luogo alle seguenti equazioni:

$$y'_1 = \bar{r}.\overline{y_2} \qquad y'_2 = \bar{r}.\overline{y_2}.y_1$$

$$C_{1,2} = \overline{y_2} \qquad C_3 = y_2$$

Un'implementazione di queste espressioni e' riportata in fig. 10.5.5.

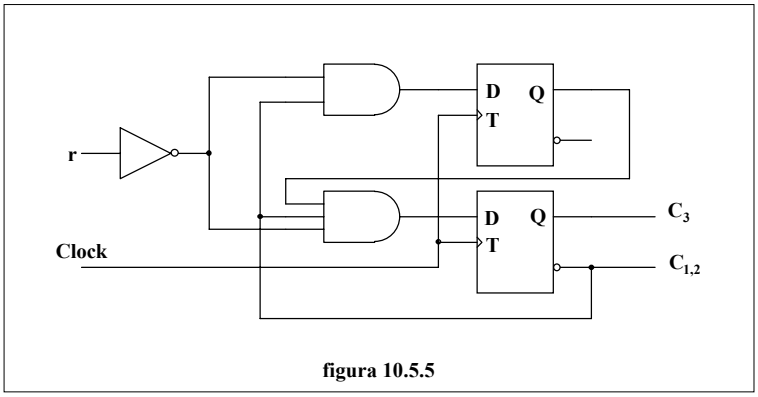

figura 10.5.5

L'unione del controllore di sequenza cosi' ottenuto e della sezione di memorizzazione dei dati e di logica di fig. 10.5.3 e' con elevata probabilita' una realizzazione altamente efficiente. Infatti la realizzazione minima, ottenuta con le tecniche formali dei capitoli precedenti, prevede 12 stati che si accordano perfettamente con 3 stati di controllo x 4 stati per i dati, che sono stati qui ottenuti. Questo non significa ovviamente che i due circuiti siano identici, ma solamente che hanno lo stesso grado di complessita'.

Sebbene la tecnica illustrata per ottenere il controllore di sequenza porti alla realizzazione minima dello stesso, in certe circostanze e' preferibile giungere a un'implementazione che utilizza un flip-flop per ciascun stato, usando un'opportuna rete combinatoria per trasferire il controllo al passo appropriato. Una rete logica atta a realizzare l'istruzione di salto del passo 1:

$$\rightarrow \left((r.1)+\left(\bar{r}.2\right)\right)$$

e' illustrata in fig. 10.5.6. Prima dell'impulso di clock che fa eseguire il passo 1, l'uscita del flip-flop sia 1. Questo segnale viene utilizzato come uno degli ingressi dei due AND della sezione di salto. L'altro ingresso di ciascun AND e' il relativo addendo dell'istruzione di salto. In generale vi sara' un AND per ciascun addendo dell'espressione di salto; poiche' uno e solo uno di tali addendi puo' essere 1 a un determinato istante, solamente la corrispondente uscita della rete logica sara' 1.

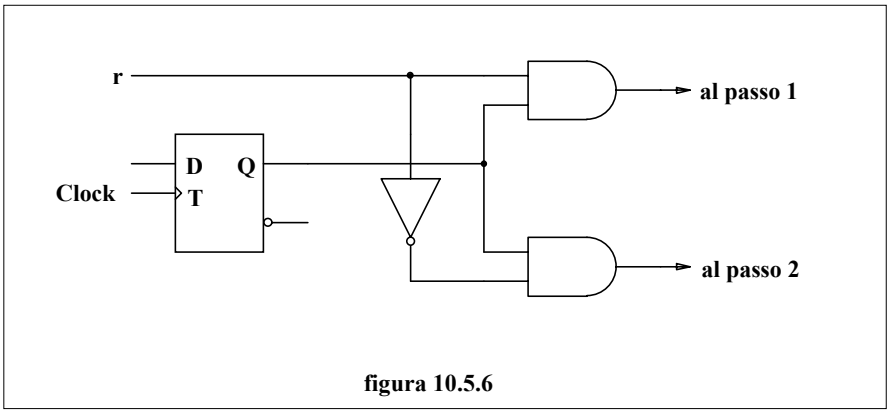

figura 10.5.6

L'uscita **Passo 1** sara' 1 se r=1 e Q=1, mentre se r=0 e Q=1 allora varra' 1 l'uscita **Passo 2**. Questo segnale andra' riportato all'ingresso D del flip-flop che comanda l'esecuzione del

passo 2. Se r=0 l'impulso di clock che esegue il passo 1 propaghera' il segnale nel flip-flop del passo 2, trasferendo il controllo al passo 2 stesso.

Quando all'ingresso di un flip-flop di controllo si deve riportare piu' di un'uscita di una rete di salto e' evidentemente necessario usare un gate OR come illustrato in fig. 10.5.7. In tal caso, se uno qualsiasi dei segnali dai passi 1, 2 o 3 e' a livello logico 1 quando arriva l'impulso di clock, allora il flip-flop del passo 1 viene posizionato a 1.

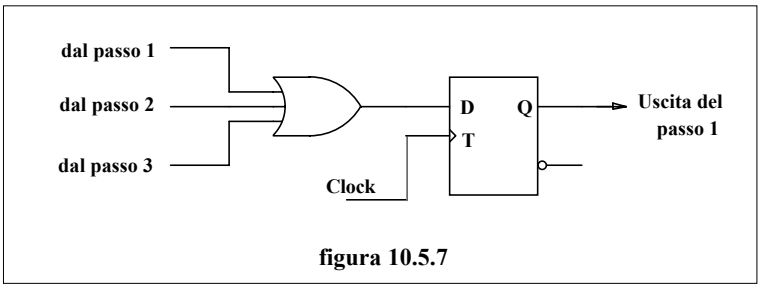

figura 10.5.7

Il circuito completo del controllore di sequenza dell'esempio che si sta trattando e' illustrato in fig. 10.5.8. Si noti che tale circuito puo' essere ottenuto con una traduzione diretta della sequenza di controllo, evitando la costruzione di un diagramma degli stati, che si rende necessaria solo quando si voglia puntare a una realizzazione minima.

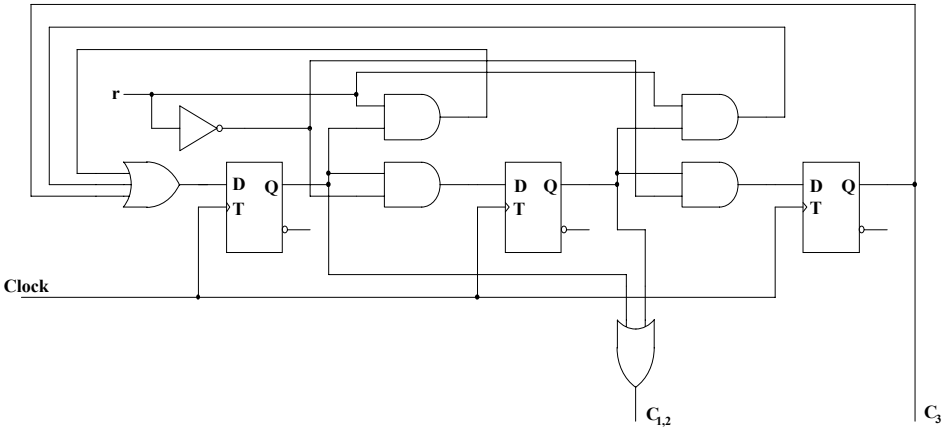

figura 10.5.8

Poiche' il controllore di sequenza ottenuto con la tecnica della minimizzazione formale e' chiaramente piu' economico di quello ottenuto per traduzione diretta della sequenza di controllo, ci si potrebbe chiedere perche' quest'ultima tecnica debba essere preferibile. La cosa e' facilmente intuibile quando si consideri che in molti casi, malgrado la divisione del circuito in sezione dati e in sezione di controllo, il numero di stati del controllore di sequenza e' comunque tanto grande da rendere il metodo formale di minimizzazione impraticabile. Inoltre anche se la minimizzazione conduce a un circuito con il minimo numero di stati, cio' non significa necessariamente che il circuito sia minimo nel suo complesso. Infine, anche se la traduzione diretta della sequenza di controllo e' un approccio inizialmente meno economico,

lo scegliere un flip-flop per passo rende piu' semplice l'esecuzione dei test, la diagnosi dei guasti e la manutenzione.

Prima di passare ad un esempio successivo e' opportuno assegnare alcune regole utili a specificare le uscite del circuito. Con il metodo fin qui visto il circuito risulta diviso nella sezione di memorizzazione e logica e nel controllore di sequenza. Le uscite del circuito possono dipendere da una sola o da ambedue le sezioni e pertanto la notazione usata per specificare l'uscita dovra' variare a seconda del caso.

Se ad esempio si prende in considerazione il circuito di fig. 10.3.1 l'uscita e' funzione della sola sezione di memorizzazione e logica e la relativa equazione e':

$$z = B_2$$

In sostanza cio' sta ad indicare che l'uscita varia ogni volta che varia B_2, ma l'uscita e' prelevata sempre nello stesso punto del circuito; pertanto e' sufficiente specificare l'uscita una volta sola. In tutti i casi simili a questo l'uscita puo' allora venir specificata con una speciale istruzione che precede la sequenza di controllo.

OUTPUT : $z = B_2$

In altri casi, in cui le uscite siano piu' d'una e dipendano da piu' di un flip-flop, l'istruzione puo' assumere ad esempio la forma:

OUTPUT : $z_1 = B_2$, $z_2 = A_0$

oppure

OUTPUT : $z = A_0.B_2$

Nell'esempio di fig. 10.5.2 l'uscita e' invece funzione sia della sezione di memorizzazione e logica che di quella di controllo. E' necessario pertanto specificare l'uscita per ciascun passo della sequenza di controllo. Il valore attuale dell'uscita dipendera' dal contenuto dei flip-flop a ciascun passo di esecuzione; la specificazione dell'uscita a ciascun passo serve a identificare la funzione logica, variabile di volta in volta, che la sintetizza. Infine, anche se fino a questo momento non se ne e' visto alcun esempio, vi possono essere dei casi in cui l'uscita sia derivata solamente dalla sezione di controllo. Anche in tal caso l'uscita va specificata passo per passo.

Per chiarezza, se ci si riferisce all'esempio che verra' presentato al paragrafo 10.8, l'istruzione:

$$\text{read} = 1$$

sta a indicare che l'uscita di nome read vale 1 al passo 3; si assume che in corrispondenza ai passi per i quali l'uscita non sia specificata, essa valga 0.

Quest'ultimo esempio chiarisce inoltre l'importante differenza tra l'istruzione:

$$\text{read} \leftarrow 1$$

e

$$\text{read} = 1$$

La prima infatti e' un'istruzione di trasferimento e indica che il valore logico 1 viene caricato nel flip-flop di nome read. In contrapposizione la seconda e' un'istruzione di interconnessione e stabilisce che la linea di nome read viene collegata al valore logico 1.

In modo del tutto simile l'istruzione:

OUTPUT : z = B

non specifica il trasferimento di un'informazione a z, bensi' che la linea chiamata z risulta collegata all'uscita del flip-flop B. E' bene osservare che la distinzione tra trasferimento e interconnessione e' sottile; tuttavia essa e' molto importante e nel seguito va sempre tenuta presente.

ESEMPIO 2

Si determini il controllore di sequenza per un circuito di verifica della parita' di sequenze di 4 bit che si presentano su una linea di ingresso seriale x. Sia prevista una linea di reset che, quando passa al livello logico 1, indica che il successivo bit in arrivo e' il primo di una sequenza. L'uscita sia costantemente a 0 tranne dopo l'arrivo del quarto bit se il test ha dato responso pari.

Soluzione

E' necessario un unico flip-flop per la sezione dati, nel quale memorizzare via via il risultato del test di parita'. La sequenza di controllo, indicando con p tale flip-flop, sara':

$$\begin{aligned}
&\textbf{1.} && p \leftarrow (p \oplus x)\cdot\bar{r};\ z = 0 \\
& && \quad \rightarrow ((r.1) + (\bar{r}.2)) \\
&\textbf{2.} && p \leftarrow (p \oplus x)\cdot\bar{r};\ z = 0 \\
& && \quad \rightarrow ((r.1) + (\bar{r}.3)) \\
&\textbf{3.} && p \leftarrow (p \oplus x)\cdot\bar{r};\ z = 0 \\
& && \quad \rightarrow ((r.1) + (\bar{r}.4)) \\
&\textbf{4.} && p \leftarrow 0;\ z = \overline{p \oplus x} \\
& && \quad \rightarrow (1)
\end{aligned} \qquad (10.5.1)$$

Al quarto passo il flip-flop p viene sempre azzerato, mentre l'uscita sara' 1 solo se il controllo di parita' avra' dato risultato pari. Poiche' l'uscita e' funzione sia della sezione di memorizzazione dei dati che della sezione di controllo, essa va specificata a ciascun passo della sequenza di controllo.

La situazione di quest'esempio, in cui l'uscita e' costantemente 0 e solo occasionalmente passa ad un altro valore, e' abbastanza comune. Per semplificare la notazione pertanto si adotta la convenzione di specificare l'uscita solo quando assume un valore diverso da zero.

E' abbastanza semplice realizzare la sezione di memorizzazione dei dati e il controllore di sequenza. Per quest'ultimo l'implementazione minima sara' data da un contatore modulo 4 con linea di reset separata.

Esiste una relazione tra la sequenza di controllo e la partizione degli stati introdotta ai capitoli precedenti nella discussione dei circuiti sequenziali sincroni. Si puo' vedere che qualsiasi sequenza di controllo di un circuito sequenziale, che non contenga dati di salto, definisce una partizione chiusa degli stati del circuito sequenziale che fa corrispondere una classe di equivalenza a ciascun passo della sequenza di controllo. Al capitolo VII e' stato messo in luce che una partizione chiusa degli stati permette sempre di raggruppare gli elementi di memoria del circuito in sottoinsiemi in modo che gli ingressi di questi elementi non siano funzione di valori memorizzati al di fuori dell'insieme di flip-flop. Nel presente contesto i flip-flop che formano questo sottoinsieme sono quelli del controllore di sequenza.

Vi sono diversi modi di sviluppare la parte circuitale destinata alla memorizzazione dei dati e alla logica prima di realizzare il controllore di sequenza. Non tutte queste realizzazioni ovviamente conterranno un numero minimo di flip- flop; inoltre una realizzazione ottimale del circuito sequenziale non e' unica e a ciascuna di esse corrisponde un diverso controllore di sequenza. E' presumibile che a ciascuna di queste realizzazioni corrisponda una diversa partizione chiusa degli stati. In particolare deve esistere sia la partizione banale in cui tutti gli stati si trovano nella stessa classe di equivalenza che quella in cui ogni stato si trova in una diversa classe. In quest'ultimo caso il controllore di sequenza coincide con il circuito sequenziale completo.

Nel primo caso invece vi e' un unico stato (e di conseguenza nessun elemento di memoria) nel controllore di sequenza. Una sequenza di controllo ad un passo e' evidentemente uno strumento poco efficace per descrivere un sistema a larga scala.

Per il controllore di parita', relativamente semplice, puo' tuttavia essere conveniente prendere in considerazione una sequenza di controllo a un passo. Essa e':

1.
$$p \leftarrow (p \oplus x)\bar{r}.\overline{N_1 . N_2}; N_2 \leftarrow (N_1 \oplus N_2)\bar{r};$$
$$N_1 \leftarrow \overline{N_2}.\bar{r}; z = \overline{(p \oplus x)}. N_1 . N_2 \qquad (10.5.2)$$

Si vede immediatamente che una realizzazione minima richiede 8 stati. Dovranno quindi essere aggiunti alla sezione di memorizzazione dei dati due flip-flop, in quanto la sezione di controllo non ha memoria. Questi due flip-flop N_1 e N_2 funzionano continuamente come un contatore modulo 4 senza segnale di reset. Si noti che ora le espressioni per z e per il prossimo stato di p sono funzioni di N_1 e N_2. Di conseguenza z e' uguale a 0 tranne quando $N_1 = N_2 = 1$ e la parita' e' memorizzata in p tranne quando $N_1 = N_2 = 1$ e il flip flop viene azzerato. Poiche' il controllore di sequenza minimo e' effettivamente un contatore modulo 4, la realizzazione (10.5.2) e' probabilmente molto simile a quella (10.5.1).

10.6) Operazioni vettoriali.

Ai paragrafi precedenti e' stata introdotta una notazione che permette di scrivere la sequenza di controllo per qualsiasi circuito sequenziale. Essa tuttavia risulta conveniente solo quando il prossimo stato di ciascun elemento di memoria deve essere specificato separatamente. Nei sistemi di maggiori dimensioni e' piu' conveniente trattare un insieme di flip-flop come un registro, evitando in tal modo di scrivere espressioni separate per il prossimo stato di ciascun flip-flop del registro.

E' necessario pertanto introdurre un'opportuna notazione per rappresentare le operazioni logiche e i trasferimenti tra registri. Si stabilisca pertanto che le grandezze scalari (i singoli flip-flop) siano identificate con lettere minuscole, mentre le grandezze vettoriali, cioe' i registri, siano indicati con lettere maiuscole. Come nei linguaggi di programmazione e' poi necessario specificare il numero di elementi di cui si compone un registro; cio' puo' esser fatto con un'istruzione di **DIMENSION**, a simulitudine di quanto si fa ad esempio nel FORTRAN, posta in testa alla sequenza di controllo di qualsiasi circuito sequenziale che contenga un registro.

Ad esempio l'istruzione:

DIMENSION A(3),B(8)

all'inizio di una sequenza di controllo stara' ad indicare che il circuito sequenziale contiene almeno 11 flip-flop, 3 nel registro A e 8 nel registro B.

Quando si rendesse necessario trasferire una costante in un registro si puo' usare la notazione:

$$A \leftarrow (1,0,1,1)$$

che sta ad indicare che nei quattro flip-flop del registro A vengono caricate rispettivamente le costanti logiche 1,0,1,1. Un particolare vettore di costanti logiche che e' conveniente definire e' $\varepsilon(n)$, cioe' un vettore di n valori logici 1.

In certi casi poi potra' essere necessario prendere in considerazione un singolo flip-flop di un certo registro. A tale fine si puo' adottare una numerazione per ciascun flip-flop da sinistra verso destra e iniziando con lo zero. Si identifichera' in tal modo il singolo flip-flop usando il suo numero come indice da accoppiare alla variabile che identifica l'intero registro.

E' inoltre possibile combinare, con l'operatore di **concatenazione**, piu' registri in un singolo vettore, che per alcune operazioni puo' venir usato come un registro di maggiori dimensioni, scrivendo i simboli dei singoli registri consecutivamente separati da una virgola. Se ad esempio A e' un registro da 4 bit, B uno da 3 bit e p un singolo flip- flop, allora:

$$C = A,p,B$$

e' un registro da 8 bit, tale che

$$C_0 = A_0 \quad C_1 = A_1 \quad C_2 = A_2 \quad C_3 = A_3 \quad C_4 = p \quad C_5 = B_0 \quad C_6 = B_1 \quad C_7 = B_2$$

E' sempre possibile infine formare un vettore da un sottoinsieme di flip-flop di un registro, attraverso l'uso degli indici e la concatenazione.

Se, ad esempio, si vuole trasferire nel registro B (da tre bit) il contenuto dei flip-flop dispari del registro D da sei bit, si puo' scrivere:

$$B \leftarrow D_1, D_3, D_5 \qquad (10.6.1)$$

Una notazione piu' complessa, ma piu' compatta puo' essere la seguente, che definisce l'operazione detta di **compressione**.

$$B \leftarrow R/Y$$

dove R e' un vettore contenente valori 1 e 0 e Y e' un registro con lo stesso numero di elementi di R. Per definizione la compressione R/Y da' origine ad un registro formato dai flip-flop di Y che corrispondono a valori 1 di R. Ad esempio la (10.6.1) puo' essere scritta, utilizzando l'operatore di compressione, nel modo seguente:

$$B \leftarrow (0,1,0,1,0,1)/D$$

Quando sia necessario isolare i primi o gli ultimi n flip-flop di un registro si possono definire due particolari forme di compressione. A questo scopo si utilizza il prefisso α^n o ω^n rispettivamente. Con tali definizioni α^n/Y indica un vettore composto dai primi n flip-flop del registro Y, mentre con ω^n/Y si indica il registro composto dagli ultimi n flip-flop di Y.

Ad esempio, se A e' un registro da 5 bit, si ha:

$$\alpha^3/A = A_0,A_1,A_2$$

e

$$\omega^2/A = A_3,A_4$$

10.7) Funzioni logiche di vettori.

Affinche' la notazione vettoriale introdotta al paragrafo precedente sia realmente in grado di semplificare la descrizione a programma di un circuito, dev'essere possibile esprimere in modo compatto le operazioni booleane che coinvolgono registri.

Una notazione che e' conveniente introdurre e' l'operazione di **riduzione** riferita agli operatori AND e OR e simbolicamente rappresentata con ./A e +/A rispettivamente.

L'operazione ./A rappresenta l'AND di tutti gli elementi di A, cioe':

$$./A = A_0.A_1 \text{ ..... } A_{\rho a-1} \qquad (10.7.1)$$

e in analogia

$$+/A = A_0 + A_1 + \text{.....} + A_{\rho a-1} \qquad (10.7.2)$$

dove ρ_a e' il numero di elementi di A.

I tre operatori logici fondamentali AND, OR e NOT possono essere applicati anche ai vettori. Ad esempio:

$$\overline{A} = \overline{A_0}, \overline{A_1}, ..., \overline{A_{\rho a-1}} \qquad (10.7.3)$$

o ancora

$$A.B = A_0.B_0, A_1.B_1, ..., A_{\rho A-1}.B_{\rho B-1} \qquad (10.7.4)$$

Evidentemente l'ultima operazione e' definita unicamente se i due registri A e B hanno lo stesso numero di elementi. Si puo' tuttavia definire un'operazione di AND e OR tra uno scalare e un vettore, intendendo in tal caso che l'operazione viene compiuta tra lo scalare e ciascun elemento del vettore. Ad esempio:

$$a.B = a.B_0, a.B_1, ..., a.B_{\rho B-1} \qquad (10.7.5)$$

In alcuni casi potra' essere necessario risistemare gli elementi del vettore. L'operazione $n \uparrow A$ significhera' ruotare gli elementi di A di n posizioni verso sinistra, mentre $n \downarrow A$ indichera' sempre una rotazione di n posizioni, ma verso destra. Nel caso dell'operazione $n \uparrow A$ gli n bit piu' a destra di A saranno rimpiazzati dagli n piu' a sinistra e analogo sara' il comportamento nel caso dell'operazione $n \downarrow A$. Ambedue tali operazioni verranno chiamate **rotazione**.

Le operazioni di **scorrimento a sinistra** $n \Uparrow A$ e di **scorrimento a destra** $n \Downarrow A$ sono strettamente correlate con quelle di rotazione, tuttavia nel caso di $n \Uparrow A$ gli n bit lasciati liberi sulla destra vengono riempiti con zeri, mentre nel caso dell'operazione $n \Downarrow A$ sono gli n bit a sinistra ad essere riempiti da zeri.

ESEMPIO 1

Valutare il contenuto dei registri ad ogni passo del seguente programma

(a) $A \leftarrow 2 \downarrow R$

(b) $B \leftarrow \Uparrow R$

(c) $C \leftarrow (./R) + S$

(d) $D \leftarrow \overline{R.S}$

assumendo che i valori iniziali siano

$R = (1,0,0,1,0,1,1) \qquad S = (1,1,0,0,0,0,1)$

Soluzione:

(a) $2 \downarrow R = 2 \downarrow (1,0,0,1,0,1,1) = (1,1,1,0,0,1,0)$

(b) $\Uparrow R = \Uparrow (1,0,0,1,0,1,1) = (0,0,1,0,1,1,0)$

(c) $$(./R)+S=(./1,0,0,1,0,1,1)+S=(1.0.0.1.0.1.1)+S=0+S=S$$

(d) $$\overline{R.S}=\overline{(1,0,0,0,0,0,1)}=(0,1,1,1,1,1,0)$$

Con l'introduzione dei simboli appena descritti si e' venuti in possesso di un linguaggio sufficientemente potente, atto a descrivere in modo efficiente la sequenza di controllo di circuiti sequenziali complessi. C'e' tuttavia da osservare che parecchie delle logiche combinatorie complesse usate nella realizzazione di un sistema digitale sono di frequente uso. E' conveniente quindi provvedere a specifiche abbreviazioni per queste funzioni in modo da rendere ancora piu' compatta la scrittura di una sequenza di controllo.

A titolo di esempio si prenda in considerazione un selettore da tre bit b_0,b_1 e b_2 . Si supponga cioe' che un passo di una sequenza di controllo usi il contenuto di un registro B da tre bit per selezionare un particolare bit di un registro A da 8 bit, ponendolo in un flip-flop r. Si avra', con le notazioni fin qui introdotte:

$$r \leftarrow \left(\left(\overline{B_0}.\overline{B_1}.\overline{B_2}\right).A_0\right)+|\left(\left(\overline{B_0}.\overline{B_1}.B_2\right).A_1\right)+\left(\left(\overline{B_0}.B_1.\overline{B_2}\right).A_2\right)+\left(\left(\overline{B_0}.B_1.B_2\right).A_3\right)+$$

$$+\left(\left(B_0.\overline{B_1}.\overline{B_2}\right).A_4\right)+\left(\left(B_0.\overline{B_1}.B_2\right).A_5\right)+\left(\left(B_0.B_1.\overline{B_2}\right).A_6\right)+\left((B_0.B_1.B_2).A_7\right) \qquad (10.7.6)$$

Si supponga ora di assegnare un nome al vettore di 8 bit, uscita del selettore da 3 bit, usando poi tale nome nella scrittura della sequenza di controllo. Si usi cioe' la notazione:

DECODE (R)

per rappresentare le 2^n uscite di un selettore di n variabili. L'espressione (10.7.6) potra' allora essere scritta come:

$$r \leftarrow +/(\text{DECODE}(B).A)$$

Usata in tal modo la notazione DECODE (B) somiglia moltissimo ad una subroutine di un programma scritto ad esempio in FORTRAN. Per questo motivo essa viene chiamata **subroutine logica combinatoria**.

Un'altra rete per la quale si usa correntemente la notazione in forma di subroutine logica combinatoria e' il sommatore. Per rappresentare l'addizione degli n bit di un registro D e degli n bit di un accumulatore AC, mettendo il risultato nel vettore di n+1 bit formato dall'accumulatore stesso e dall'extra flip-flop m, si puo' usare la notazione:

$$m, AC \leftarrow \text{ADD}(AC, D)$$

E' poi comune avere l'occasione di usare una terza subroutine logica combinatoria per incrementare il contenuto di un registro, sommando cioe' uno al contenuto del registro stesso (si assuma che il bit 0 sia quello piu' significativo). La rete logica necessaria per incrementare il contenuto di un arbitrario registro R da quattro bit e' illustrata in fig. 10.7.1. L'uscita di questa rete e' il vettore da quattro bit INC (R).

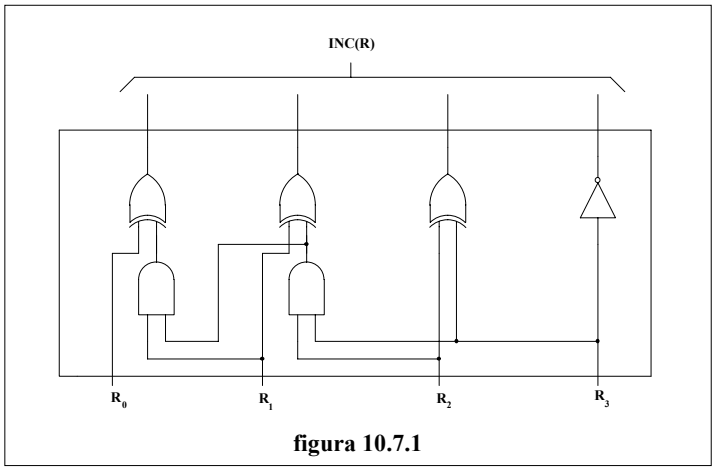

figura 10.7.1

L'istruzione di trasferimento:

$$R \leftarrow INC(R) \qquad (10.7.7)$$

fa si' che il contenuto di R sia aumentato di un'unita' quando il corrispondente passo della sequenza di controllo viene eseguito.

Nei capitoli precedenti tuttavia e' gia' stato fatto notare che la realizzazione piu' efficiente di un contatore e' quella che usa flip-flop di tipo T. Se quindi un registro e' usato unicamente come contatore, cioe' appare solo in istruzioni del tipo (10.7.7), allora si puo' utilizzare tale contatore proprio per realizzare la funzione di incremento.

In questo caso l'istruzione:

$$CT \leftarrow INC(CT)$$

puo' essere usata per rappresentare un segnale di controllo capace di accumulare un impulso di clock nel contatore CT, come illustrato in fig. 10.7.2, e di conseguenza si puo' usare allo scopo che ci si propone un contatore standard costruito in un singolo involucro e scelto tra gli elementi a media scala di integrazione.

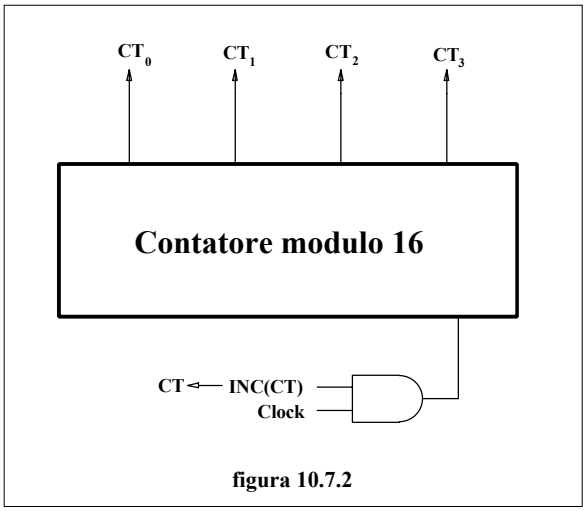

figura 10.7.2

L'esempio appena fatto mette in luce che una subroutine logica combinatoria descrive in modo univoco l'operazione da eseguire; tuttavia la realizzazione circuitale puo' essere di diverso tipo e variare di volta in volta.

10.8) Applicazioni.

In questo paragrafo verranno presi in considerazione due esempi di uso del linguaggio che si e' appena introdotto. In ambedue gli esempi sara' necessario manipolare una notevole quantita' di dati e cio' rende poco pratico l'uso dei metodi classici di progetto per i circuiti sequenziali sincroni.

ESEMPIO 1

Si vuole progettare un controller da usare in unione a un registratore digitale a cassette magnetiche per realizzare un sistema di verifica di circuiti combinatori che abbiano fino a otto ingressi e otto uscite. I circuiti da verificare sono realizzati mediante elementi logici integrati e sono disponibili solo i terminali di ingresso e di uscita. Il sistema che si vuol realizzare inoltre deve solo essere in grado di segnalare malfunzionamenti. Se cioe' alcune uscite non assumono il valore corretto in corrispondenza ad un dato vettore di ingresso, allora il circuito sotto esame non deve venir accettato.

Sul nastro sono memorizzate coppie di caratteri da 8 bit, rappresentative di un ingresso e del corrispondente valore che l'uscita deve assumere. Il controller deve quindi essere in grado di manipolare fino a 256 coppie.

Poiche' il sistema non ha alcuna necessita' di operare a frequenze di clock molto elevate, si puo' supporre per semplicita' che controller e registratore siano pilotati in modo sincrono dello stesso segnale di clock.

Nei periodi di clock, durante i quali e' presente sulla relativa linea il segnale di controllo **read**, deve essere eseguita dal nastro la lettura di un carattere di 8 bit e tale carattere viene caricato nel registro di transito **TAPE**. Nei periodi invece in cui e' attiva la linea **rewind**, il nastro viene riavvolto. Alla fine delle operazioni di riavvolgimento, oppure quando il sistema viene acceso per la prima volta, il lettore di nastro attiva per un periodo di clock il segnale **ready**.

Il controller deve fornire gli appropriati segnali di controllo all'unita' a nastro, piazzare ciascun vettore di ingresso in un registro connesso con gli ingressi del circuito in esame e

confrontare le uscite con il vettore di uscita desiderato. Quando viene rilevata una discrepanza tra il vettore d'uscita attuale e il vettore di uscita desiderato deve venir commutato un segnale di uscita z, che dovra' rimanere tale finche' il sistema non viene riazzerato con l'applicazione di un segnale generato manualmente e applicato alla linea **go**, dopo che un nuovo circuito da sottoporre a test sia stato collegato al tester.

Soluzione

Gli elementi di memoria, siano essi vettori o scalari, richiesti per la sezione dati e logica del tester, sono illustrati in fig. 10.8.1. Vi sono tre registri da 8 bit. Il registro **TV** sara' collegato ai terminali di ingresso del circuito sotto esame, quello **YD** sara' usato per memorizzare il valore desiderato dell'uscita, mentre il terzo registro **COUNT** sara' usato come contatore modulo 256 per contare il numero di vettori di test applicati al circuito. Quando il contatore raggiunge lo zero, un segnale applicato alla linea **rewind** fara' riavvolgere il registratore e avvisera' l'operatore che il circuito esaminato non presenta inconvenienti e che un nuovo circuito puo' essere inserito.

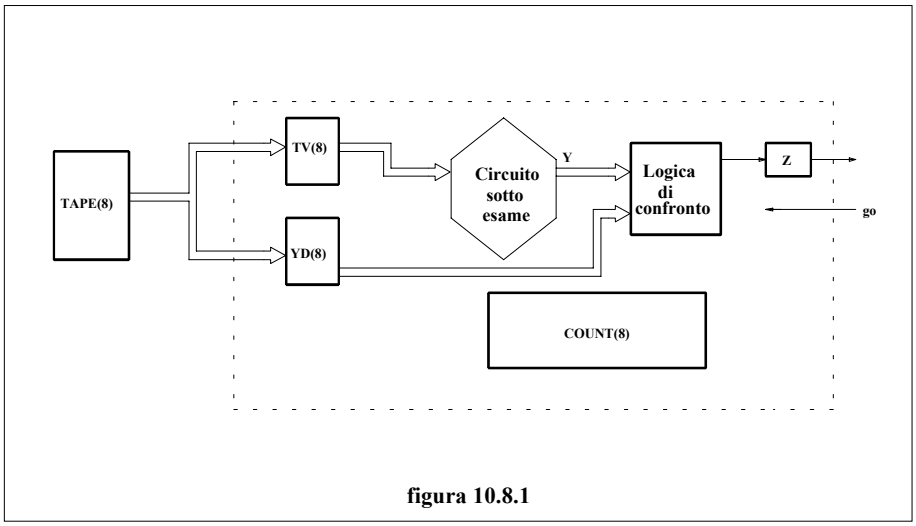

figura 10.8.1

Le linee connesse alle uscite del circuito sotto test verranno chiamate per ragioni di convenienza **Y**. Infine il flip- flop **z** verra' posto a 1 quando verra' rivelato un errore.

Le rimanenti linee di segnale dall'unita' di controllo al registratore non vengono illustrate, in quanto in fig. 10.8.1 e' riportata unicamente la sezione dei dati e logica.

La descrizione della sequenza di controllo puo' iniziare con una lista degli ingressi e delle uscite che non dipendono dall'unita' di controllo.

INPUT : TAPE(8), go, ready
OUTPUT : z

Questa non e' un'operazione strettamente necessaria, tuttavia nei circuiti complessi in cui ci sono parecchi segnali tra interni ed esterni e' opportuno specificare chiaramente quali siano i segnali che comunicano con il mondo esterno, in contrapposizione con quelli che agiscono solo all'interno del circuito.

L'effettiva sequenza di controllo inizia appena prima dell'inizio del test di un nuovo circuito. Il solo trasferimento associato ai primi due passi e' l'azzeramento del flip-flop z.

1. $\rightarrow \left((\text{ready}.2)+(\overline{\text{ready}}.1)\right)$

2. $z \leftarrow 0$
$\rightarrow \left((\text{go}.3)+(\overline{\text{go}}.2)\right)$

Ciascun test individuale, relativo a un ben preciso pattern di ingresso, inizia al passo 3. Dapprima un segnale **read** = 1 fa si' che un vettore di ingresso sia caricato nel registro **TAPE**. Questo vettore viene poi trasferito in **TV**, eccitando il circuito sotto prova. Successivamente il vettore di uscita desiderato viene trasferito in **YD** sempre attraverso **TAPE**.

3. $\text{read} = 1$

4. $\text{TV} \leftarrow \text{TAPE}$

5. $\text{read} = 1$

6. $\text{YD} \leftarrow \text{TAPE}$

Al passo 7 **COUNT** viene incrementato e allo stesso tempo l'uscita attuale del circuito in prova viene confrontata con l'uscita desiderata. Il simbolo di OR esclusivo verra' utilizzato come abbreviazione dell'equivalente logico in termini di AND, OR e NOT.

Se vi e' una differenza tra due qualsiasi bit dell'uscita attuale e dell'uscita desiderata, allora l'espressione:

$$(+/\text{Y} \oplus \text{YD})$$

assumera' il valore 1 e il controllo verra' trasferito al passo 8. A questo punto il flip-flop z di errore verra' posto a 1, verra' iniziato il riavvolgimento della cassetta, il contatore verra' azzerato e il controllo trasferito al passo 1.

7. $\text{COUNT} \leftarrow \text{INC}(\text{COUNT})$
$\rightarrow \left(((+/\text{Y} \oplus \text{YD}).8)+(\overline{(+/\text{Y} \oplus \text{YD})}.9)\right)$

8. $z \leftarrow 1, \text{rewind} = 1, \text{COUNT} \leftarrow \bar{\varepsilon}(8)$
$\rightarrow (1)$

Se il test ha invece successo si passa al passo 9 dove il contenuto di **COUNT** viene esaminato. Se e' pari a 0 si prosegue verso il passo 10, in caso contrario si ritorna al passo 3.

9. $\rightarrow \left(\left(\overline{(+/\text{COUNT})}.10\right)+((+/\text{COUNT}).3)\right)$

10. $\text{rewind} = 1$

$\rightarrow (1)$

Il circuito di controllo della sequenza e' illustrato in fig. 10.8.2. Come nel caso di circuiti progettati con metodi convenzionali, l'inizio delle operazioni necessita di un segnale di reset,

che non e' incluso nelle specifiche iniziali. Tale segnale deve porre a 1 il flip-flop 1 e azzerare tutti gli altri. Il flip-flop 1 abilita i gate che ricevono il segnale **ready**.

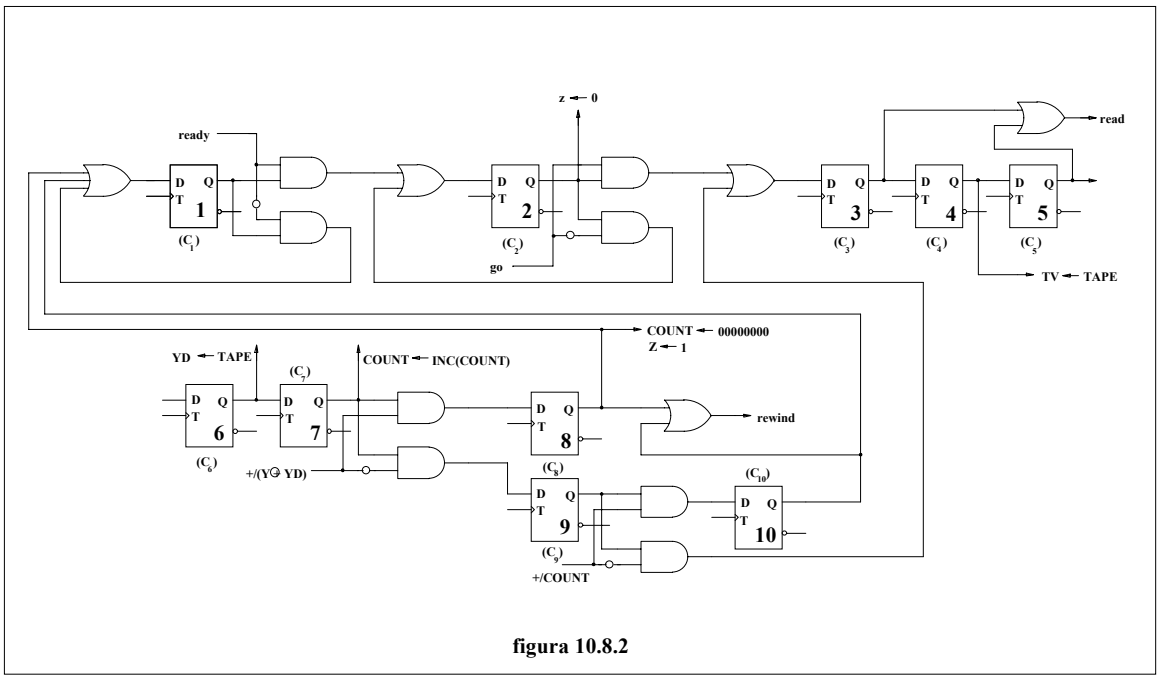

figura 10.8.2

Finche' **ready** rimane a zero, gli impulsi di clock mantengono il flip-flop 1 al valore logico 1, mentre quando **ready** passa a 1, il successivo impulso di clock azzera il flip-flop 1 e setta il flip-flop 2. Viene in tal modo generato il segnale di controllo C_2, connesso all'ingresso K del flip-flop z, in modo che al successivo impulso di clock esso venga azzerato. C_2 inoltre abilita i gate cui e' applicato il segnale **go**, in modo che il flip-flop 2 sia azzerato e il flip-flop 3 sia settato quando **go** passa da zero a uno.

Si noti che le linee di uscita di ciascun flip-flop sono indicate con il trasferimento che ciascuna di esse abilita oppure con il nome del segnale di controllo generato.

I passi 3 e 5 sono esempi di casi in cui le uscite del circuito sono generate unicamente dal controllore di sequenza, come e' stato gia' precedentemente accennato, con la linea **read** posta a 1 durante tali passi. Come illustrato in fig. 10.8.2 cio' viene ottenuto semplicemente con l'operazione di OR di C_3 e C_5 ; si ha pertanto **read** = 1 durante i passi 3 e 5 e zero durante tutti gli altri passi. Il segnale **rewind** e' generato in modo analogo ai passi 8 e 10.

Il presente paragrafo puo' essere concluso con un altro esempio in cui viene richiesta una maggiore quantita' di memoria per immagazzinarvi le informazioni. Si puo' soddisfare questa nuova esigenza introducendo una notazione matriciale per rappresentare un insieme di registri.

Si supponga infatti di avere un gran numero di registri, ciascuno con lo stesso numero di bit. Anziche' assegnare un diverso simbolo a ciascuno di questi registri e' piu' conveniente usare lo stesso simbolo per tutti i registri dell'insieme, distinguendo ciascun registro con un indice.

Un insieme di registri, organizzato in tal guisa, puo' essere chiamato **memoria** e ciascun registro puo' essere chiamato parola. In definitiva una memoria M di m parole sara' formata dai registri $M^0, M^1, \ldots, M^{m-1}$.

Il modo in cui un'informazione puo' venir recuperata da una memoria varia a seconda della realizzazione della memoria stessa.

Per l'esempio che segue si supporra' di usare una memoria allo stato solido in cui i livelli di uscita di ciascun registro siano continuamente disponibili. Eseguendo l'AND dei livelli di uscita di ciascun registro con le linee di uscita di un decodificatore di indirizzo e mettendo in OR i vettori risultanti e' possibile trasferire i livelli d'uscita del registro voluto alle linee di uscita.

Indicando con AR il registro degli indirizzi, l'espressione logica combinatoria relativa e', se i bit di indirizzo sono n (indicando con m la quantita' 2^n-1):

$$\left(M^0 . \mathrm{DECODE}(AR)_0\right) + \left(M^1 . \mathrm{DECODE}(AR)_1\right) + \ldots + \left(M^m . \mathrm{DECODE}(AR)_m\right)$$

Tale espressione puo' essere usata sul lato destro di un'istruzione di trasferimento in un registro e un'operazione di lettura da una memoria puo' essere considerata terminata quando il vettore di uscita sara' stato trasferito in un registro di dati.

Per ragioni di convenienza, l'espressione appena introdotta puo' essere abbreviata introducendo la subroutine logica combinatoria:

$$\mathrm{SELECT}(M, \mathrm{DECODE}(AR))$$

Nell'esempio che segue si introdurra' inoltre la notazione per il **trasferimento condizionato** che permette di trattare casi in cui in un dato punto di una sequenza di controllo vi siano parecchi possibili trasferimenti in funzione di altri dati.

ESEMPIO 2

Si vuole progettare un semplice controllore per una macchina utensile che usi una memoria di sola lettura per la memorizzazione delle sequenza delle posizioni dell'utensile stesso. Vengono usati numeri a 18 bit per specificare la posizione spaziale di quest'ultimo; il modo in cui questa posizione viene specificata non interessa in questa sede.

Esistono quattro possibili sequenze di posizioni, ciascuna delle quali puo' venir richiesta dall'operatore a qualsiasi istante e ciascuna sequenza e' lunga 256 parole. Il sistema dovra' quindi comprendere un registro **PR** da 18 bit per memorizzare la posizione corrente dell'utensile. Il governo elettronico della macchina interroghera' continuamente questo registro per mezzo di tre convertitori digitali/analogici.

Le comunicazioni con l'operatore sono realizzate con un flip-flop **ss** di start-stop e un registro selettore di sequenza **SSR** da 2 bit in cui le combinazioni di bit 00,01,11 e 10 corrispondono rispettivamente alle sequenze A, B, C, D.

Quando l'operatore mette a 1 **ss** fa si' che il controllore inizi a leggere la sequenza specificata da **SSR**. Quando invece **ss** passa a zero, la sequenza in atto viene interrotta e il registro **PR** viene azzerato. Se il controllore completa la sequenza, cioe' legge in ordine tutte le 256 parole di quest'ultima, allora vengono azzerati **ss** e **PR**.

L'operatore puo' agire per modificare il contenuto di **ss** e **SSR** in qualsiasi istante, tuttavia vi sia un meccanismo di sincronizzazione che impedisce la modifica durante la fase attiva di un impulso di clock.

Si deve notare che nel caso che si sta trattando gli ingressi vengono trattati differentemente che non nei precedenti esempi. In questo caso infatti gli ingressi possono essere modificati in qualsiasi istante e questi cambiamenti vengono immediatamente memorizzati, tuttavia non si ha alcun effetto finche' non si raggiunge un appropriato punto della sequenza.

Le linee di ingresso non appariranno pertanto nella sequenza di controllo. La loro esistenza sara' tuttavia implicita, come molto spesso e' implicita l'esistenza di una linea di reset.

Soluzione

I soli registri in piu' necessari, oltre a quelli gia' descritti, sono un registro a 10 bit per memorizzare l'indirizzo voluto della posizione di memoria da leggere e la memoria stessa. I loro nomi saranno rispettivamente **AR** e **ROM**. La completa configurazione dei registri e' illustrato in fig. 10.8.3; la memoria ROM contiene 1024 parole da 18 bit e la sequenza A e memorizzata nelle prime 256 posizioni, la B nelle 256 successive e cosi' via. La sequenza di controllo e' riportata di seguito. Essa inizia con una specificazione di uscita, che ricorda che il registro **PR** pilota i convertitori i quali a loro volta posizionano l'utensile.

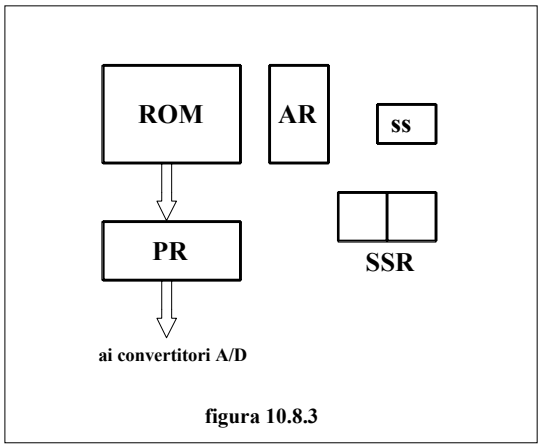

figura 10.8.3

Si noti che non vi sono specificazioni di ingresso poiche', come si e' gia' accennato, non vi e' alcun ingresso diretto.

Il passo 1 realizza una condizione di attesa finche' **ss** non passa a 1; in tal caso si procede verso il passo 2 che carica il primo indirizzo della sequenza voluta in **AR**.

OUTPUT : PR

1. $\rightarrow \left((ss.2)+\left(\overline{ss}.1\right)\right)$

2. $AR \leftarrow \left(\overline{e(10)} * \left(\overline{SSR_0}.\overline{SSR_1}\right)\right)+\left((0,1,\overline{e(8)}) * \left(\overline{SSR_0}.SSR_1\right)\right)+$
$+\left((1,0,\overline{e(8)}) * \left(SSR_0.\overline{SSR_1}\right)\right)+\left((1,1,\overline{e(8)}) * \left(SSR_0.SSR_1\right)\right)$

3. $PR \leftarrow SELECT(ROM, DECODE(AR))$

4. $AR \leftarrow INC(AR)$
$\rightarrow ((ss.5)+(ss.7))$

5. $\rightarrow \left(\left(\left(+/\omega^8/AR\right)3\right)+\left(\left(+/\omega^8/AR\right)6\right)\right)$

6. $ss \leftarrow 0$

7. $PR \leftarrow \overline{e(18)}$

$$\rightarrow (1)$$

Il passo 2 e' un esempio di trasferimento condizionale, la cui notazione generale e':

$$A \leftarrow (B^* f_1) + (C^* f_2)$$

dove f_1 e f_2 sono due funzioni logiche e l'asterisco acquista il significato di "se". In sostanza l'istruzione dice "trasferisci in A il contenuto di B se $f_1 = 1$ oppure trasferisci in A il contenuto di C se $f_2 = 1$". Si noti che in un'istruzione di questo tipo le funzioni f_i devono essere mutuamente esclusive, cioe' una sola di esse puo' valere uno ad un determinato istante.

L'espressione del passo 2 pertanto carica nel registro **AR** il valore 0, 256, 512 o 768 a seconda del contenuto di **SSR**.

Il passo 3 e' un esempio di istruzione di lettura da una memoria. Il contenuto della locazione della **ROM** specificata in **AR** viene trasferito in **PR**.

L'indirizzo contenuto in **AR** viene poi incrementato al passo 4, in modo da predisporre il sistema a leggere la parola successiva. Il passo 4 tuttavia contiene anche un esame del flip-flop **ss**. Se l'operatore ha azzerato tale flip-flop si passa al passo 7, nel quale viene azzerato **AR** e si ritorna poi al passo 1.

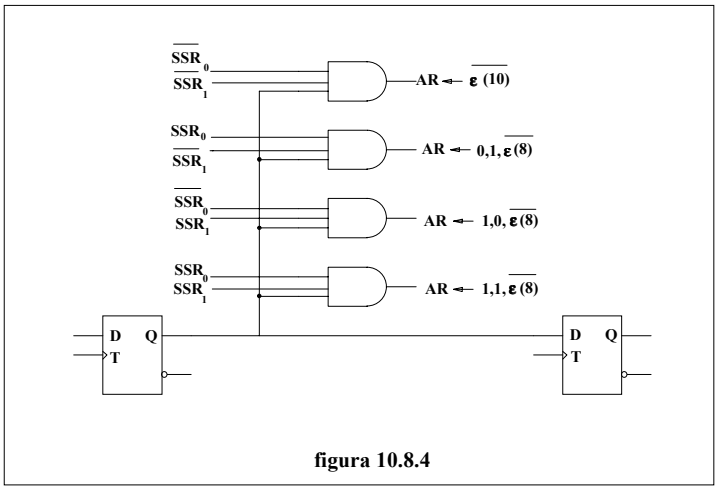

figura 10.8.4

Il passo 5 rimanda al passo 3 finche' non sono state lette 256 parole. Questa condizione e' rivelata dal fatto che alla fine di una sequenza gli ultimi 8 bit di **AR** sono sempre nulli. Alla fine della sequenza i passi 6 e 7 azzerano **ss** e **PR** e rimandano al passo 1.

In fig. 10.8.4 e' riportata la porzione di circuito di controllo che implementa il passo 2. Si puo' facilmente vedere perche' si e' parlato in questo caso di trasferimento condizionale; infatti la logica ha la stessa struttura di quella introdotta in precedenza per implementare salti condizionali. In questo caso tuttavia l'istruzione non serve per alterare l'ordine con cui viene eseguita la sequenza di controllo, ma semplicemente per eseguire un opportuno trasferimento.

10.9) Conclusioni.

La crescente affidabilita' ed economicita' della realizzazione di funzioni complesse in tecnica integrata fa si' che al giorno d'oggi i sistemi da progettare contengano piu' elementi di quanti sia conveniente manipolare con i tradizionali metodi di progetto, che fanno uso della tavola di stato.

In questo capitolo e' stato presentato un tipo di approccio che puo' essere usato con sistemi di qualsivoglia dimensione. Non vi e' tuttavia alcuna garanzia che la realizzazione ottenuta sia minima, ma il linguaggio introdotto presenta diversi vantaggi rispetto ai metodi intuitivi che in caso contrario il progettista sarebbe costretto ad adottare. Una grande importanza inoltre ha il rigore formale del linguaggio, che permette il trasferimento non equivoco di informazioni.

Gli esempi presentati in questo capitolo hanno preso in esame sistemi relativamente semplici in modo da dare semplicemente un'idea della metodologia. Maggior approfondimento si potra' avere riferendosi ad esempio a "Digital System: Hardware Organization and Design" di F.J.Hill e G.R.Peterson edito dalla Wiley e Sons, New York, 1973.