CAPITOLO XI
INTEGRAZIONE A MEDIA E LARGA SCALA

11.1) Introduzione.

Molto spesso e' stato affermato che l'avvento di tecnologie LSI e MSI ha invalidato i tradizionali metodi di progetto logico e reso necessario il ricorso ad altre metodologie. Tale affermazione tuttavia non va intesa in modo esclusivo; e' bensi' vero che aspetti riguardanti costo, uso, costruzione e progetto di sistemi digitali sono stati rivoluzionati dalle tecnologie LSI e MSI. Malgrado cio' solo gli ultimi punti del processo di progetto, come la costruzione delle maschere, la realizzazione del layout e delle interconnessioni e i problemi orientati alla verifica del circuito sono stati profondamente modificati, mentre la parte iniziale del progetto, che e' sotto certi aspetti la piu' creativa, e' rimasta in sostanza inalterata. L'algebra booleana rimane ancora il metodo base per descrivere formalmente il funzionamento di circuiti logici combinatori; pertanto il progetto di logiche combinatorie da implementare con tecnologie LSI non differisce in sostanza dal progetto fatto per un'implementazione a logica sparsa o addirittura a componenti discreti. L'uso di parti MSI puo' tuttalpiu' limitare la liberta' d'azione del progettista. Cio' non significa tuttavia che in certi casi particolari i problemi non possano essere affrontati in maniera inusuale. Ad esempio e' interessante vedere come utilizzando componenti a media scala di integrazione possano essere realizzate funzioni combinatorie di qualsiasi tipo.

	A	B	C	D	y
1	0	0	0	0	y_0
	0	0	0	1	y_1
2	0	0	1	0	y_2
	0	0	1	1	y_3
3	0	1	0	0	y_4
	0	1	0	1	y_5
4	0	1	1	0	y_6
	0	1	1	1	y_7
5	1	0	0	0	y_8
	1	0	0	1	y_9
6	1	0	1	0	y_{10}
	1	0	1	1	y_{11}
7	1	1	0	0	y_{12}
	1	1	0	1	y_{13}
8	1	1	1	0	y_{14}
	1	1	1	1	y_{15}

figura 11.1.1

Si supponga infatti di essere in possesso di un multiplexer con n bit di indirizzo, cioe' di un dispositivo in grado di smistare un ingresso verso una di 2^n linee di uscita in funzione del

valore presente sulle n linee di indirizzo. Con l'uso di tale dispositivo e' molto semplice realizzare qualsiasi funzione combinatoria di n+1 variabili.

Si consideri per semplicita' un multiplexer con tre bit di indirizzo. Con il suo uso e' possibile realizzare qualsiasi funzione combinatoria da quattro variabili. Si ricordi che le funzioni di quattro variabili sono in numero di $2^{2^4} = 2^{16}$.

La tavola di verita' di una generica funzione di 4 variabili e' riportata in fig. 11.1.1, con la particolarita' che le righe sono stata suddivise in 8 gruppi di due righe ciascuno. Si noti che la variabile D assume in ciascun gruppo il valore 0 e rispettivamente 1 in corrispondenza alla prima e alla seconda riga del gruppo.

Si considerino ora i valori che una generica funzione puo' assumere in corrispondenza all'i-esimo gruppo. Poiche' i soli possibili valori sono 00, 01, 10 e 11 si deduce che:

$$y_{2(i-1)}, y_{2i-1} = \begin{cases} 0 \\ D \\ \overline{D} \\ 1 \end{cases}$$

Questa osservazione permette di realizzare la funzione nella forma illustrata in fig. 11.1.2, connettendo semplicemente i vari ingressi alle linee $0, D, \overline{D}, 1$, secondo quanto specificato nella tavola di verita'.

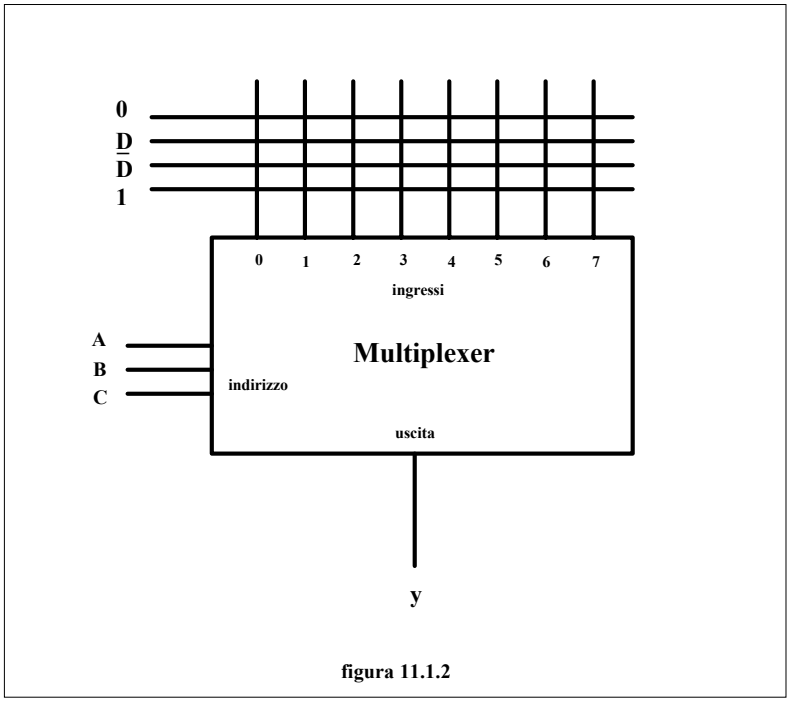


figura 11.1.2

Si e' ottenuta in tal modo una realizzazione particolarmente semplice, che non ha richiesto alcuna minimizzazione e che inoltre permette con la stessa configurazione circuitale di sintetizzare una qualsiasi tra le 65536 funzioni di quattro variabili. E' evidente che sarebbe estremamente interessante essere in possesso di una sorta di macroalgebra, ma

sfortunatamente nessuno a tutt'oggi l'ha inventata. D'altra parte va osservato che molto spesso un progetto utilizza contemporaneamente gate individuali e parti LSI e MSI.

Per quanto riguarda i sistemi digitali complessi diventa estremamente importante capire perfettamente il comportamento delle parti sequenziali; infatti i vincoli associati alle temporizzazioni dei circuiti sequenziali sono molto stretti e, poiche' le modifiche apportate a un progetto sono di solito molto costose, diviene quasi indispensabile avere a disposizione uno strumento di indagine per verificare le temporizzazioni di un sistema. Uno degli strumenti piu' efficaci a questo scopo e', come si vedra' in seguito, la simulazione.

E' abbastanza probabile che chi si occupa di progettazione di circuiti sequenziali con parti standard MSI abbia poche occasioni di utilizzare i metodi classici; egli si rivolgera' piuttosto a linguaggi del tipo di quello illustrato al capitolo X. Al contrario chi si occupa del progetto di parti MSI sara' costretto ad utilizzare i metodi classici, considerando anche che, in relazione al gran numero di elementi che dovranno poi essere prodotti, il progetto minimo assume grande importanza.

La parte finale di questo capitolo sara' poi dedicata a una breve introduzione ai problemi del "testing" dei circuiti. Per evidenziare infatti eventuali malfunzionamenti di un circuito LSI o MSI e' necessario mettere a punto delle configurazioni di ingresso tali che in uscita appaia l'effetto del malfunzionamento stesso. Cio' e' vero in particolare per i circuiti LSI nei quali il numero dei terminali di uscita e' piccolo rispetto al numero delle connessioni interne tra i gate. C'e' da notare tuttavia che, specie per circuiti sequenziali complessi, l'individuazione delle sequenze di test puo' essere un processo molto difficoltoso. Per i circuiti sequenziali infatti non esiste alcuno schema completamente automatico per la generazione dei test; si puo' anzi affermare che oggi la difficolta' di verifica del circuito e' uno dei maggiori, se non il maggior ostacolo, all'aumento della densita' di integrazione e della complessita' del singolo integrato.

11.2) Definizioni.

Si definisce circuito integrato un circuito, montato in un unico contenitore, nel quale gli elementi attivi e passivi sono fabbricati a partire da un unico substrato di materiale semi-conduttore. Piu' difficile e' definire la complessita' di un circuito integrato, cioe' dove termina la categoria dei circuiti integrati ordinari ed inizia quella dei circuiti MSI e LSI. Il diagramma di fig. 11.2.1 puo' aiutare ad inquadrare la questione.

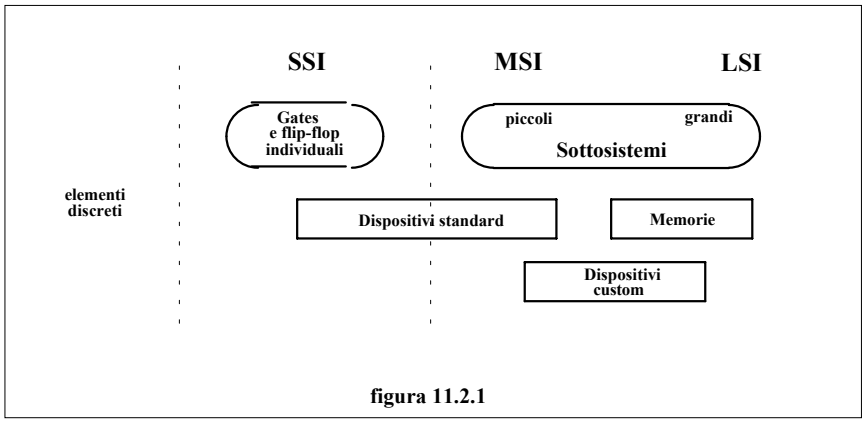


figura 11.2.1

A stretto rigore il termine circuito integrato comprende anche gli elementi LSI e MSI. Tuttavia nell'uso comune con circuiti integrati (detti anche elementi SSI - small scale

integration) si intendono quei dispositivi con un basso numero di gate o di flip-flop in cui tutti gli ingressi e le uscite degli elementi sono disponibili.

Piu' difficile e' fissare un confine tra gli elementi MSI (medium scale integration) e quelli LSI (large scale integration); usualmente gli elementi MSI comprendono piccoli sottosistemi montati in un unico involucro, come ad esempio contatori, decodificatori, circuiti per la verifica della parita' ecc. Sistemi piu' complessi, ad esempio parti non trascurabili di un calcolatore, quali sommatori, sezioni di memoria, unita' aritmetiche complete o addirittura unita' centrali di calcolo, montati in un unico involucro sono di solito considerati elementi LSI.

Una distinzione piu' importante che non quella tra elementi MSI e LSI e' quella tra elementi integrati standard e elementi integrati "custom". Dalla fig. 11.2.1 si vede che gli elementi standard contengono mediamente meno gate che i dispositivi "custom".

Qualsiasi circuito che un costruttore consideri potenzialmente redditizio sara' costruito e distribuito come elemento standard, mentre i circuiti che via via perdono la loro redditivita' vengono scartati. Gli elementi che divengono popolari vengono poi copiati dagli altri costruttori. Non bisogna considerare tale fatto negativo o illecito, in quanto qualsiasi produttore di apparecchiature sara' sempre riluttante ad adottare elementi LSI e MSI a meno che non vi sia piu' di una fonte di approvvigionamento. La tendenza attuale e' quindi quella che viaggia verso una standardizzazione sempre piu' larga.

Gli elementi standard MSI comprendono decodificatori binari e BCD, contatori binari e decimali, registri a scorrimento, comparatori, generatori di parita', ecc. La maggior parte delle realizzazioni LSI comprendono invece memorie RAM e ROM.

Si ricorre all'uso di elementi "custom" solo quando la realizzazione con elementi standard non appare conveniente. E' necessario in tal caso fornire al fabbricante una qualche descrizione del circuito. Alcuni fabbricanti accettano descrizioni in termini di diagramma a blocchi, ma il problema potrebbe essere semplificato sia per il committente che per il fabbricante adottando qualche forma di linguaggio atto alla descrizione di sistemi a larga scala del tipo di quello illustrato al capitolo X.

11.3) Il progetto con parti MSI standard.

L'uso di parti standard piu' complesse che non i singoli gate tende a semplificare la procedura di progetto anziche' complicarla.

In effetti una parte dell'attivita' del progettista si riduce a cercare sul catalogo del venditore quegli elementi che possono essere adattati all'obiettivo che egli si propone. Di conseguenza, quando una consistente porzione della funzione totale del sistema e' stata realizzata con parti standard, le manipolazioni booleane necessarie vengono corrispondentemente ridotte.

E' interessante inoltre notare che, anche quando le parti standard sono sequenziali, il problema della loro interconnessione e' molto spesso un problema completamente combinatorio. In tal caso l'approccio da usare non differisce in alcunche' da quello usato nelle realizzazioni a logica sparsa o a componenti discreti.

Per convincersi della potenza e della semplicita' del progetto con parti standard MSI e' opportuno prendere in considerazione alcuni esempi.

ESEMPIO 1

Si voglia progettare un circuito in grado di confrontare due numeri binari da 8 bit A e B. Le due uscite Z_2Z_1 devono assumere valore 10 se A > B, 11 se A = B o 01 se A < B. Siano disponibili come parti standard MSI comparatori da 4 bit. Si indichino con A_2 e B_2 i due numeri binari costruiti con i quattro bit piu' significativi di A e B, mentre A_1 e B_1 siano quelli costruiti con i quattro bit meno significativi. Gli 8 bit di A_2 e B_2 possono essere usati come ingressi di un comparatore MSI da 4 bit. Le corrispondenti uscite siano f_1 e f_2 , mentre con g_1 e g_2 verranno indicate le uscite del comparatore cui vengono applicati A_1 e B_1.

Il significato delle uscite dei due comparatori e le tavole necessarie per il progetto del comparatore a 8 bit sono riportati in fig. 11.3.1.

f_2	f_1	Significato
0	0	---
0	1	$A_2 < B_2$
1	1	$A_2 = B_2$
1	0	$A_2 > B_2$

g_2	g_1	Significato
0	0	---
0	1	$A_1 < B_1$
1	1	$A_1 = B_1$
1	0	$A_1 > B_1$

g_2g_1 \ f_2f_1	00 ($A_2<B_2$)	01 ($A_2=B_2$)	11 ($A_2>B_2$)	10
00	---	---	---	---
01 ($A_1<B_1$)	---	<	<	>
11 ($A_1=B_1$)	---	<	=	>
10 ($A_1>B_1$)	---	<	>	>

figura 11.3.1

E' evidente che l'unica operazione necessaria e' quella di costruire le uscite Z_2Z_1 in funzione di f_2 f_1 e g_2 g_1 come illustrato nella mappa di Karnaugh di fig. 11.3.2.

g_2g_1 \ f_2f_1	00	01	11	10
00	---	---	---	---
01	---	01	01	10
11	---	01	11	10
10	---	01	10	10

Z_2Z_1

figura 11.3.2

In genere per progettare un circuito e' necessario prendere in considerazione i singoli bit; in questo caso tuttavia cio' non e' necessario in quanto, come illustrato in fig. 11.3.1, la relazione tra A e B e' completamente definita dalle relazioni tra i due insiemi separati di

quattro bit. Pertanto per ottenere una rappresentazione sotto forma di mappa di Karnaugh delle uscite Z_2 e Z_1 e' sufficiente rimpiazzare nella tabella di fig. 11.3.1 le relazioni tra A e B con la loro forma codificata. A partire dalla mappa di Karnaugh cosi' ottenuta, la realizzazione minima e' allora:

$$Z_1 = \overline{f_2} + f_1 . g_1$$
$$Z_2 = \overline{f_1} + f_2 . g_2$$

e il relativo circuito e' riportato in fig. 11.3.3.

Si noti che per progettare il circuito non vi e' stato alcun bisogno di speciali tecniche MSI, tuttavia il risultato e' probabilmente piu' semplice di qualsiasi progetto che si dovesse avvalere unicamente di gate individuali.

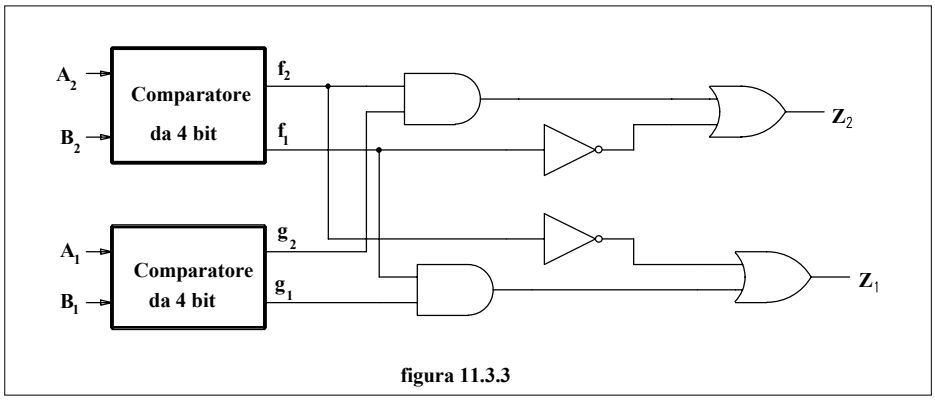


figura 11.3.3

L'esempio che segue si riferisce ad un circuito sequenziale e, come accade di solito in questi casi, non fa uso della tavola degli stati dell'intero circuito.

A tutt'oggi, d'altra parte, rimane un problema di notevole interesse quello di individuare la tavola di stato di una parte standard entro la tavola di stato del circuito sequenziale da progettare.

ESEMPIO 2

Si debba progettare per un determinato processo produttivo un semplicissimo controllore di sequenza. Su 25 linee, da Z_0 a Z_{24}, debbono comparire sequenzialmente dei segnali a livelli in modo che una e una sola delle linee vada a 1 a ciascun periodo di clock. La sequenza di controllo deve inoltre essere ripetitiva, di modo che Z_0 vada a 1 immediatamente dopo Z_{24}. Sono disponibili dei contatori da 4 bit e dei decoder da 4 a 16 linee.

Il progetto deve evidentemente venir impostato attorno ai contatori da 4 bit, dotati di un ingresso di clock e di un ingresso di reset. Le quattro uscite siano C_8, C_4, C_2 e C_1; il contatore venga azzerato quando l'ingresso di reset viene portato a 1. Il reset sia sincrono, cioe' l'azzeramento avvenga in coincidenza con il successivo impulso di clock.

Una sequenza di lunghezza 16 puo' venir generata molto facilmente connettendo le uscite di un contatore esadecimale agli ingressi di un decoder. Per questi ultimi e' molto frequente che l'uscita eccitata vada a zero mentre le altre rimangono a 1; si indicheranno pertanto tali uscite con $\overline{d}_0, \overline{d}_1, ..., \overline{d}_{15}$.

Per contare fino a 24 e' necessario aggiungere un flip-flop la cui uscita y sara' il bit piu' significativo del conteggio. La tavola di stato di questo flip-flop e' riportata in fig. 11.3.4. Ogni volta che il conteggio arriva a 15 ($\bar{d}_{15} = 0$) con il successivo impulso di clock il flip-flop y passa a 1, mentre il contatore viene resettato a zero.

Il 24-esimo impulso di clock pone evidentemente $d_8 = 1$ mentre il 25-esimo azzera y, come illustrato in fig.11.3.4 nella casella d_8=y=1; contemporaneamente viene azzerato il contatore.

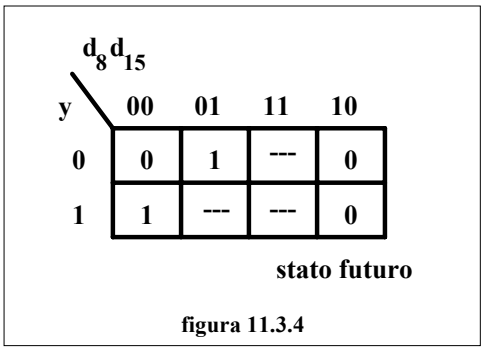


figura 11.3.4

Da tale tavola di stato si ricavano facilmente le equazioni di eccitazione del flip-flop.

$$S_y = d_{15} \qquad R_y = d_8$$

mentre il contatore, che si azzera comunque dopo che il conteggio ha raggiunto il 15, verra' resettato semplicemente connettendo il suo ingresso R di reset alla funzione $d_8 . y$.

I primi 16 segnali di controllo sono dati da:

$$Z_0 = d_0 . \bar{y} = \overline{\overline{d_0} + y}$$
$$Z_1 = d_1 . \bar{y} = \overline{\overline{d_1} + y}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_{15} = d_{15} . \bar{y} = \overline{\overline{d_{15}} + y}$$

mentre i restanti 9 sono:

$$Z_{16} = d_0 . y = \overline{\overline{d_0} + \bar{y}}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$Z_{24} = d_8 . y = \overline{\overline{d_8} + \bar{y}}$$

Il progetto completo dell'unita' di controllo e' riportato in fig. 11.3.5.

E' evidente che usando gate individuali i 25 NOR d'uscita non sarebbero necessari, in quanto potrebbero essere combinati nella funzione di decodifica. Nel caso in esame invece il decoder e' una parte standard MSI e non puo' essere modificato. Tuttavia la configurazione raggiunta e' senz'altro piu' economica di quella che si otterrebbe con gate individuali.

Quanto piu' e' complesso un sistema, tanto piu' e' probabile che esso venga realizzato usando parti MSI. Nel capitolo precedente d'altra parte si e' visto che i sistemi piu' complessi possono essere descritti mediante un opportuno linguaggio di progetto. Si puo' pertanto

sperare di riuscire a realizzare uno specifico progetto realizzando una sequenza di controllo che descriva elementi a media scala di integrazione.

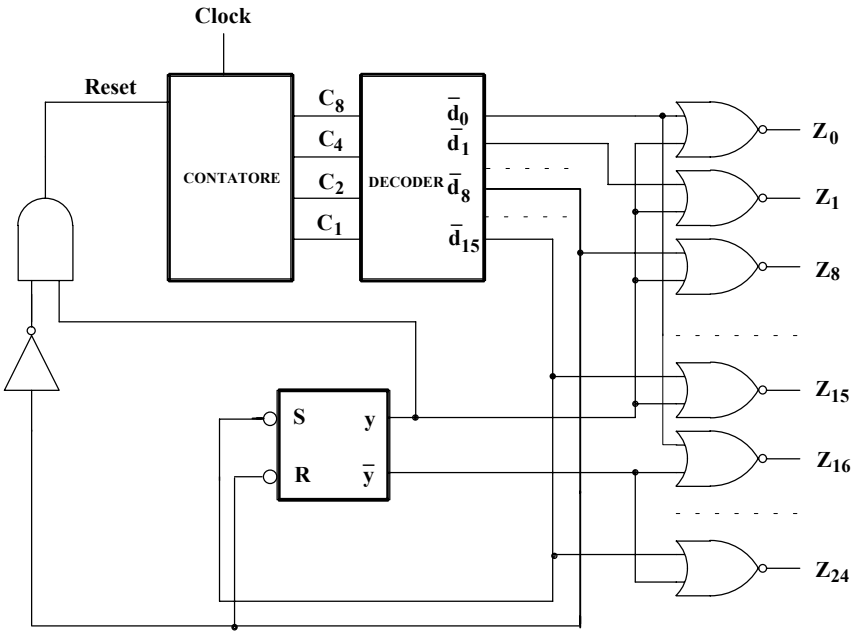


figura 11.3.5

C'e' tuttavia da tener presente che il numero di terminali di ciascun elemento MSI e' limitato. Ora, poiche' ogni flip-flop di un registro richiede al minimo 2 e possibilmente 4 collegamenti di input/output, ne consegue che in un contenitore da quattordici piedini potranno trovar posto da 2 a 5 flip-flop. Pertanto registri di una certa dimensione e per estensione subroutine logiche combinatorie di tipo vettoriale non possono essere implementate come parti singole.

Da tutto cio' ovviamente non deriva che i sistemi a larga scala non possano essere formulati in termini di sequenza di controllo o che non sia opportuno utilizzare, per quanto possibile, parti MSI. Semplicemente le realizzazioni MSI risultano di solito molto meno eleganti di quanto si sarebbe potuto sperare. In sostanza esse consisteranno in poche parti MSI interconnesse da un gran numero di gate individuali.

Nell'esempio che segue si formulera' una sequenza di controllo in modo che la realizzazione finale utilizzi il piu' possibile parti standard MSI.

ESEMPIO 3

Si voglia sviluppare un'interfaccia tra un calcolatore e una stampante. Le informazioni sono trasmesse in parallelo su sei linee, individuate dal vettore X. Ogni combinazione rappresenta un carattere da stampare con l'eccezione di $\overline{\varepsilon(6)}$ il cui apparire fa si' che venga eseguito un ritorno carrello e un avanzamento di riga.

Come illustrato in fig. 11.3.6, l'interfaccia deve generare un segnale sulla linea indicata con **ready** ogni volta che e' pronta a ricevere un carattere, mentre ogni volta che un carattere e' pronto in X il calcolatore fornira' il segnale **read**.

Nel progetto dovra' essere incluso un sincronizzatore che generi il segnale **sread**, in concomitanza con il clock e della sua stessa durata, ogni volta che sulla linea **read** compare un segnale.

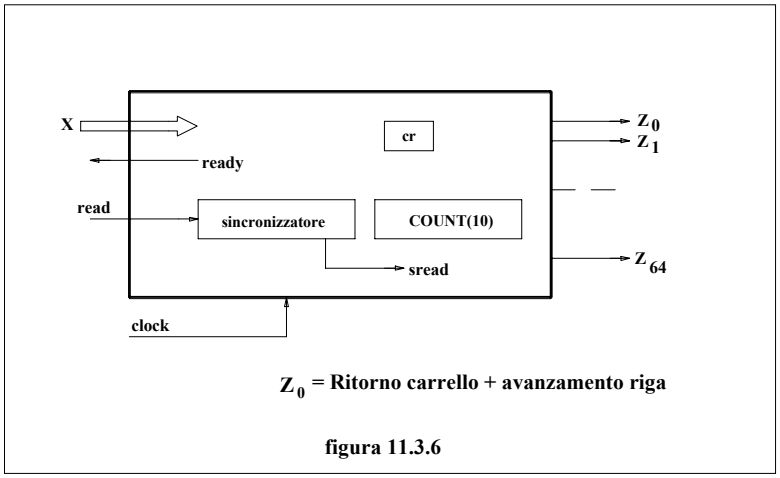


figura 11.3.6

Le uscite verso la stampante consistono in 64 linee, individuate dal vettore Z; sulla linea Z_k comparira' un impulso di durata pari a quello di clock ogni volta che all'ingresso X viene applicato in numerazione binaria il numero k. In particolare poi il segnale della linea Z_0 fara' si' che venga eseguito un ritorno carrello e un avanzamento riga.

Dopo l'inizio della stampa di un carattere l'interfaccia dovra' attendere un tempo pari a 16 msec. prima di generare il successivo segnale di **ready**, mentre nel caso di un ritorno carrello l'attesa dovra' essere di un secondo.

Si voglia infine realizzare l'interfaccia con parti standard MSI.

Soluzione

Si supponga che come parti standard siano disponibili decoders da 4 bit, mentre non siano disponibili quelli da 6 bit.

Un efficiente metodo per temporizzare l'attesa prima di generare il segnale **ready** richiede l'uso di un contatore; se si usa un clock relativamente lento da 1 kHz, un contatore da 4 bit terminera' il suo conteggio in 16 msec., mentre uno da 10 bit terminera' il suo ciclo in poco piu' di un secondo, che e' l'intervallo di tempo da far seguire ad un ritorno carrello. Si supponga che un contatore da 10 bit sia disponibile come parte MSI standard. In fig. 11.3.6 tale parte e' identificata con **COUNT** mentre **cr** e' un flip-flop che viene posizionato a 1 in presenza di un ritorno carrello.

Prima di tentare di costruire un diagramma logico a blocchi per l'unita' di interfaccia, e' necessario darne una formulazione quale sequenza di controllo.

Si tenga presente che il vettore di uscita Z deve coincidere con $\overline{\varepsilon(64)}$ in ogni istante, eccetto che per un periodo di clock in corrispondenza ad ogni carattere ricevuto. Durante tale periodo una e una sola linea d'uscita deve essere posta a 1. Il controllore di sequenza quindi

presentera' un segnale di nome **typ**, della durata di un periodo di clock, quando apparira' tale uscita. L'uscita **z** viene pertanto definita preliminarmente con :

$$\textbf{OUTPUT: } Z = \left(\text{typ}.\overline{X}_0.\overline{X}_1.\text{DECODE}\left(\omega^4/X\right)\right), \left(\text{typ}.\overline{X}_0.X_1.\text{DECODE}\left(\omega^4/X\right)\right),$$
$$\left(\text{typ}.X_0.\overline{X}_1.\text{DECODE}\left(\omega^4/X\right)\right), \left(\text{typ}.X_0.X_1.\text{DECODE}\left(\omega^4/X\right)\right)$$

Questa espressione logica combinatoria, che e' la concatenazione di quattro vettori da 16 bit, rappresenta l'uscita istante per istante. Si noti inoltre che tutti i bit di questo vettore saranno 0 a meno che **typ** non valga 1; in quest'ultimo caso solamente un bit varra' 1. La subroutine logica combinatoria **DECODE** altro non e' che il decoder MSI da 4 bit. Si osservi che per ricavare un decoder da 6 bit da uno da 4 bit sono necessari 64 gate AND e quattro invertitori.

I passi rimanenti della sequenza di controllo sono:

1. $\text{COUNT} \leftarrow \overline{\varepsilon(10)}; \text{cr} \leftarrow 0$
$\rightarrow \left((\text{sread}.2)+\left(\overline{\text{sread}}.1\right)\right)$

2. $\text{cr} \leftarrow \overline{X}_0.\overline{X}_1.\text{DECODE}\left(\omega^4/X\right)_0; \text{COUNT} \leftarrow \text{INC}(\text{COUNT}); \text{typ} = 1$

3. $\text{COUNT} \leftarrow \text{INC}(\text{COUNT})$
$\rightarrow \left(\left(+/\left(\left(\text{cr}.\alpha^6/\text{COUNT}\right), \omega^4/\text{COUNT}\right)\right).4\right)+$
$\left(+/\left(\left(\text{cr}.\alpha^6/\text{COUNT}\right), \omega^4/\text{COUNT}\right).3\right)$

4. $\text{ready} = 1$
$\rightarrow (1)$

Si supponga che il contatore sia progettato in modo da incrementarsi ad ogni impulso di clock a meno che al suo ingresso di reset non sia applicato un 1. L'ingresso di reset pertanto verra' connesso alla linea di controllo $\text{COUNT} \leftarrow \overline{\varepsilon(10)}$.

Dopo che un carattere sara' stato fornito, il passo 2. porra' **cr** a 1 qualora tale carattere fosse il ritorno carrello e porra' **typ** a 1 per abilitare l'uscita. Si noti che, sebbene non vi sia alcun segnale di controllo che corrisponda a:

$$\text{COUNT} \leftarrow \text{INC}(\text{COUNT})$$

il contatore viene incrementato sia al passo 2. che al passo 3. finche' gli ultimi 4 bit di **COUNT** non si azzerano oppure, se **cr**=1, finche' tutti i 10 bit non sono contemporaneamente nulli. Solo successivamente, al passo 4., viene generato il segnale **ready** ed il controllo viene restituito al passo 2. in attesa del prossimo carattere.

Uno schema semplificato sia del controllore di sequenza che della parte di memorizzazione dei dati e logica e' riportata in fig. 11.3.7.

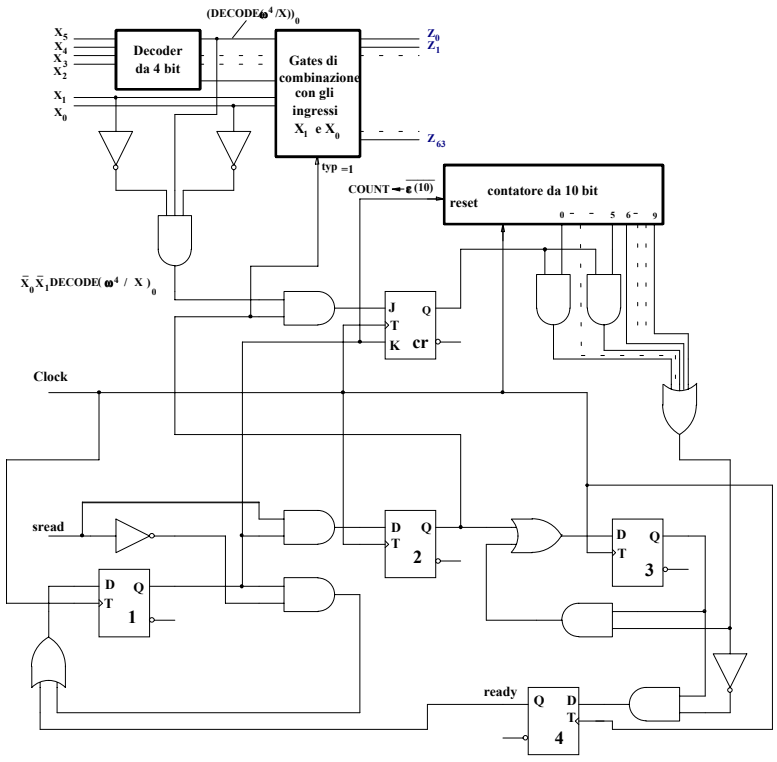


figura 11.3.7

Sebbene l'uso di contatori MSI e di decoder da 4 bit permetta una significativa riduzione del numero di interconnessioni richiesto, puo' destare qualche perplessita' il numero di gate individuali che rimangono nella rete logica. Tale situazione e' tuttavia tipica di tutti quei sistemi che hanno un elevato numero di ingressi e di uscite.

11.4) Considerazioni economiche sui circuiti integrati.

Sulla scelta della tecnologia con cui implementare un determinato sistema digitale pesa evidentemente in maniera determinante il rapporto costo - velocita'. Tuttavia per ragioni di semplicita' nel seguito si fara' l'ipotesi che la velocita' non sia un fattore critico.

Si consideri pertanto la situazione nella quale un'adeguata realizzazione di un circuito possa essere ottenuta sia utilizzando gate individuali ed elementi di memoria, sia usando gate individuali e parti standard MSI, sia ricorrendo ad una realizzazione LSI specializzata. E' ovvio che la scelta tra i tre possibili approcci e' determinata in questo caso unicamente da considerazioni economiche.

E' necessario come primo passo sviluppare un'espressione che permetta di valutare il costo di produzione di n parti MSI o LSI.

Un costo considerevole e' certamente quello associato alla produzione del layout, delle maschere, alla simulazione e alla generazione delle sequenze di test. Questi costi, chiamati costi di progetto, sono indipendenti dal numero di dispositivi prodotti e variano con la complessita' della parte progettata. Un'espressione approssimata per il costo di progetto, quando g sia il numero di gate contenuti nel dispositivo, e':

costo di progetto = A + B.g

A e B sono due costanti che dipendono dalla particolare tecnologia impiegata e che vanno via via diminuendo nel tempo. L'unita' di misura normalmente adottata e' il migliaio di dollari.

Il costo totale di produzione di n parti puo' pertanto essere espresso in via approssimata dell'espressione:

$$\textbf{Costo di n parti} = A + B.g + (C + D.g).n \qquad (11.4.1)$$

dove C + D.g e' un costo associato con ciascuna parte prodotta che copre i costi del materiale, del lavoro, di esecuzione dei test e gli imprevisti.

Per un circuito LSI specializzato, il cui volume di produzione e' al massimo di alcune migliaia di pezzi, il costo di progetto e' il termine preponderante. Al contrario, per una parte standard MSI, il cui volume di produzione e' di decine e decine di migliaia di pezzi, il costo di progetto diviene una parte trascurabile del costo totale di un singolo esemplare. Il costo di una parte standard e' pertanto:

Costo di una parte standard = C + D.g

E' forse opportuno a questo punto approfondire quanto esposto attraverso un esempio.

ESEMPIO

Si supponga che un progettista abbia sviluppato una logica combinatoria composta da 85 gate.

Si supponga inoltre che vi siano due possibili approcci per realizzare il progetto. Il primo faccia ricorso ad una parte MSI standard da 90 gate molto simile alla rete logica voluta e che costi K dollari per elemento. Sia inoltre necessaria una rete speciale da 30 gate per realizzare l'interconnessione con la porzione utilizzabile del dispositivo MSI in modo da ottenere quanto desiderato. In tale situazione e' abbastanza logico chiedersi se il dispositivo da costruire non sia realizzabile con maggior economia utilizzando un circuito integrato di tipo "custom".

Si supponga infine che il costo dei gate individuali sia di 0.05 dollari/gate e che si debbano realizzare m esemplari del circuito.

Soluzione

Il costo dei componenti per n esemplari di una rete ibrida e':

$$\textbf{Costo dei componenti della rete ibrida} = (K + 30 \,.\, 0{,}05).m =$$

$$= (K + 1.5).m \qquad (11.4.2)$$

E' necessario poi tenere in conto un costo addizionale non trascurabile per la realizzazione delle interconnessioni. Assumendo che il costo totale di interconnessione sia di W dollari si ha:

Costo della realizzazione ibrida = (K + W + 1.5).m (11.4.3)

Nel caso di una realizzazione totalmente "custom" la (11.4.1) si riduce per m esemplari a:

Costo della realizzazione "custom" = p + Q.m

quando il numero di gate sia fissato (85).

Le relazioni ricavate sono riportate in grafico in fig. 11.4.1 e per facilitare il confronto tra le due possibili soluzioni su tale grafico sono riportate sia la relazione (11.4.2) che la (11.4.3)

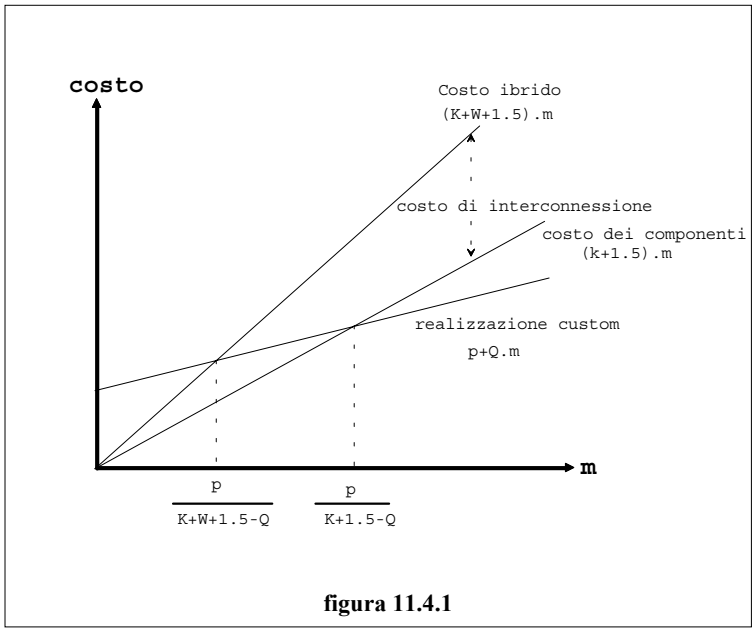


figura 11.4.1

Risulta evidente che la realizzazione "custom" e' conveniente solo per volumi produttivi molto elevati, mentre la realizzazione ibrida va utilizzata solo per volumi produttivi bassi. Il punto di incrocio e' determinato dal costo di interconnessione, che nel processo totalmente "custom" non ha alcun peso.

Si supponga ad esempio che un particolare produttore di semiconduttori quoti la realizzazione "custom" 2000 dollari piu' 2 dollari per parte prodotta. Si supponga inoltre che questo stesso produttore possa fornire parti MSI standard a 2 dollari a esemplare (K = 2 dollari). Il costo di interconnessione per la realizzazione ibrida sia poi di 1.5 dollari ad esemplare.

In tali condizioni l'approccio "custom" sara' preferibile per livelli produttivi in cui m soddisfa la diseguaglianza:

$$2000 + 2.m < 5.m$$

cioe' per

$$m > 667$$

L'esempio che si e' preso in considerazione si riferisce ovviamente ad una situazione estremamente semplice; d'altra parte puo' servire come base per l'analisi di casi piu' realistici.

In genere un sistema complesso puo' essere partizionato in diversi sottosistemi. Di solito vi sono parti MSI standard che sono un'accettabile approssimazione di parecchi, ma non di tutti questi sottosistemi. La stessa partizione in effetti puo' dipendere dal fatto se sia economicamente conveniente o meno una realizzazione di tipo "custom".

Infine poiche' in generale vi e' un significativo interesse nei confronti delle memorie ROM, e poiche' le considerazioni economiche ad esse relative si differenziano per qualche aspetto da quelle finora viste, ad esse si fara' riferimento nel seguito del presente paragrafo.

Tali memorie possono venir utilizzate per immagazzinarvi le sequenze di controllo di unita' microprogrammabili o per rimpiazzare logiche combinatorie. E' opportuno pero' ricordare che il ritardo di una ROM puo' essere dell'ordine della decina di volte il ritardo attraverso due livelli di un circuito realizzato con componenti veloci. Pertanto l'uso di ROM nella realizzazione di logiche combinatorie e' limitato a quelle applicazioni in cui la velocita' non e' determinante.

L'uso di ROM infine indirizza verso realizzazioni ibride, con parti combinatorie ed elementi di memoria realizzati con chip separati. Se tutte queste considerazioni non portano a escludere a priori l'uso di ROM, allora l'opportunita' o meno del loro impiego diventa una questione puramente economica.

Esistono due tipi di ROM utilizzabili; il primo e' programmabile direttamente dall'utente, solitamente con mezzi elettrici; il secondo tipo invece viene programmato direttamente in fase di costruzione registrando permanentemente i dati durante il processo di fabbricazione.

Le **ROM programmabili** dall'utente possono essere considerate parti LSI standard ed il costo della loro programmazione e' analogo al costo di interconnessione di cui si e' parlato in precedenza. Il costo unitario di queste ROM e' notevolmente maggiore di quelle del secondo tipo; tuttavia non vi e' alcun costo aggiuntivo da prendere in considerazione. Come suggerito dal grafico di fig. 11.4.1 le ROM programmabili saranno la scelta piu' opportuna quando la produzione e' di pochi esemplari.

Per quanto riguarda le **ROM** del secondo tipo, dette **a maschera**, vi sono due passi per prepararne la produzione. Il primo passo consiste nel predisporre per la produzione un chip standard completo dei circuiti di indirizzamento. Il secondo passo consiste nell'utilizzare una lista fornita dall'utente, in cui e' descritto il contenuto desiderato della memoria, in modo da sviluppare la maschera necessaria al processo di metallizzazione finale.

Il costo della ROM pertanto puo' essere spezzato in tre parti e cioe' nel costo per la preparazione del passo 1 e del passo 2 e nel costo addizionale associato con la produzione di ciascun esemplare.

Una parte del costo del passo 1 puo' essere indicata, come gia' e' stato detto, con A. Un'altra parte del costo andra' riferita al decodificatore degli indirizzi e potra' essere considerata approssimativamente proporzionale al numero (2^n) di linee di uscita del decodificatore stesso. La parte rimanente del costo sara' proporzionale al numero totale di bit della ROM, in quanto per ciascuno di essi si rende necessario un transistore. Indicando con b il numero totale di bit per parola, il costo del passo 1 sara':

$$\textbf{Costo del passo 1} = A + B.2^n + C.(2^n.b)$$

dove normalmente B >> C.

Il costo di produzione della maschera necessaria alla metallizzazione finale contiene una parte fissa e una parte proporzionale al numero totale di bit. Si ottiene:

Costo del passo 2 = D + E.(2^n.b)

Il costo incrementale associato ad ogni ROM prodotta e' funzione essenzialmente del costo dell'involucro e del costo di connessione tra i terminali del circuito e i piedini dell'involucro. Questo costo e' approssimativamente proporzionale al numero totale di piedini e pertanto vale:

Costo incrementale = F.(n + b)

Non si tentera' in questa sede di dare dei valori alle sei costanti delle espressioni che precedono, in quanto esse sono troppo influenzate dall'evolvere della tecnologia. Tuttavia queste espressioni possono essere utilmente impiegate dal progettista per decidere quando una ROM e' conveniente per una data applicazione, stimando, quando necessario, dai listini prezzi tali costanti .

Le ROM sono normalmente disponibili in un certo numero di configurazioni standard. Alcuni esempi possono essere le dimensioni 16 x 16, 256 x 16, 512 x 8, ma la lista potrebbe essere molto piu' grande ed in pratica una memoria si puo' ottenere come parte standard in qualsiasi configurazione in cui il numero delle parole e il numero di bit per parola (detto anche parallellismo) siano potenze di 2.

Se una ROM standard puo' essere adattata ad una data applicazione, gran parte del costo del passo 1 puo' essere evitato, in quanto tale costo puo' essere distribuito sull'elevatissimo numero di utilizzatori della medesima configurazione circuitale.

Il costo del passo 2 e il costo incrementale devono invece essere sostenuti dall'utente individuale. Se sono necessarie p copie identiche di una particolare ROM, il costo unitario C sara' allora:

$$C = \frac{D + E.(2n.b)}{p} + F.(n + b)$$

E' interessante notare che il costo del decodificatore degli indirizzi, parte non trascurabile del costo del passo 1, non compare in quest'ultima espressione.

Per valori di p molto piccoli il progettista dovra' mettere a confronto il costo di ROM a maschera con quello di realizzazione di una rete di circuiti integrati SSI e con quello di una ROM programmabile. E' evidente che si devono attentamente considerare anche i costi degli zoccoli, talvolta superiori addirittura a quello degli integrati, dei circuiti stampati e i costi di assemblaggio.

L'utilizzo di una ROM permette comunque di evitare un gran numero di connessioni. La metallizzazione finale infatti compie tale funzione, in una maniera forse brutale, ma sicuramente sistematica ed efficace. Sfortunatamente quando si abbia a che fare con funzioni a singola uscita il punto di incrocio tra i vari tipi di realizzazione si pone a valori di p molto elevati.

L'uso piu' comune di ROM con alti valori di parallellismo si ha in unita' di controllo microprogrammate. In tale applicazione infatti per ogni indirizzo e' necessario un numero di bit sufficiente a specificare l'origine e la destinazione di un trasferimento tra registri e l'indirizzo della successiva istruzione. Un numero di bit particolarmente alto e' richiesto quando si vuol ottenere una notevole flessibilita', utilizzando salti condizionati. In queste

particolari condizioni l'uso di una ROM puo' essere conveniente anche quando p sia uguale a 1.

11.5) La simulazione.

Tutte le volte che l'implementazione di un sistema digitale include una parte MSI o LSI di tipo "custom", il sistema deve operare correttamente fin dalla prima volta. Non e' infatti possibile pensare di apportare una serie di modifiche al progetto del circuito o di costruire un prototipo da sottoporre a test. Ogni modifica al progetto del circuito richiede perlomeno la sostituzione di una costosa maschera di diffusione o di metallizzazione e la fabbricazione di qualsiasi parte nuova puo' comportare ritardi di giorni o addirittura di settimane. Il tradizionale progetto a prova e errore non e' pertanto praticabile e porterebbe di sicuro ad una rapida bancarotta la compagnia che lo praticasse.

Poiche' gli errori di progetto sono inevitabili, e' necessario dotarsi di metodi per verificare un sistema prima che questo venga costruito. La risposta sta nel ricorso alla simulazione. Nel presente paragrafo verranno dati dei cenni sui problemi connessi alla simulazione di circuiti sequenziali. E' opportuno rilevare infatti che la necessita' di simulare logiche puramente combinatorie e' molto scarsa e si limita il piu' delle volte al calcolo dei valori funzionali per tutte le possibili combinazioni degli ingressi. Vi sono parecchi modi di affrontare la simulazione dei circuiti sequenziali. Il metodo che si sceglie e' funzione del livello di dettaglio con cui si vuol conoscere il funzionamento del circuito. In termini generali si puo' dire che quanto maggiore e' il livello di dettaglio o la precisione della simulazione, tanto maggiore sara' il tempo di calcolo. Tale parametro assume un notevole interesse quando si vogliono simulare reti che contengono un numero di gate superiore al centinaio.

Esistono sostanzialmente quattro diversi modi per affrontare la simulazione, ciascuno con il suo livello di dettaglio.

A. **Analisi della rete**.
B. **Modo asincrono con uscita analogica**.
C. **Modo asincrono con uscita binaria**.
D. **Modo sincrono.**

In fig. 11.5.1 e' riportato un grafico qualitativo del contenuto informativo che puo' essere ottenuto in funzione del tempo di calcolo per ciascun tipo di simulazione.

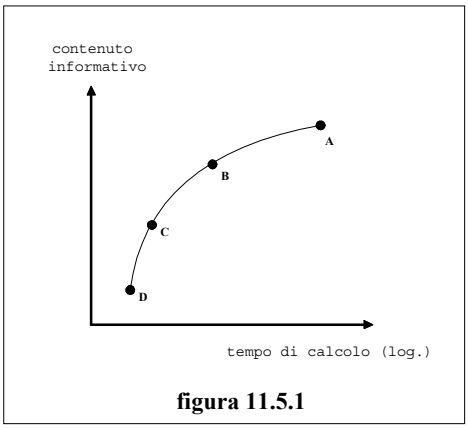


figura 11.5.1

Le simulazioni piu' accurate sono evidentemente basate sui programmi di analisi circuitale (punto A di fig. 11.5.1), quali ad esempio lo SCEPTRE e lo SPICE. In tal caso per ottenere la massima precisione nel simulare il funzionamento interno dei singoli elementi circuitali e nel ricavare i valori delle uscite non si pongono vincoli al tempo di calcolo. Sfortunatamente per grosse reti logiche il costo della simulazione condotta per tale via diviene proibitivo .

All'estremo opposto della curva di fig. 11.5.1 (punto D) si trova la simulazione sincrona. E' il tipo di simulazione meno costosa e va usato non appena sia in grado di fornire il dettaglio richiesto. C'e' tuttavia da rilevare che non tutti i dispositivi MSI o LSI sono progettati per operare in modo sincrono. Inoltre molto spesso le informazioni che si vogliono ottenere riguardano la possibilita' o meno che un sistema ha di operare ad una determinata frequenza di clock e tale informazione puo' essere ottenuta solo ricorrendo ad una simulazione asincrona in modo fondamentale.Per definizione, infatti, una simulazione sincrona implica un'iterazione per ciascun periodo di clock. I valori delle uscite e degli stati futuri vengono cioe' calcolati dalla rete combinatoria e memorizzati in corrispondenza ad ogni istante di clock.

Un numero molto maggiore di iterazioni viene richiesto quando la simulazione viene condotta in modo asincrono, anche se il risultato in output dovesse risultare sincrono. Si devono cioe' avere diverse iterazioni tra successive variazioni del clock per permettere ai segnali di propagarsi correttamente e lo stesso clock deve venir considerato come un qualsiasi altro ingresso. Un tal modo di procedere tuttavia permette di ottenere una messe di informazioni notevolmente maggiore che non operando nel modo sincrono. Per ridurre il tempo di calcolo alcune simulazioni di tipo sofisticato usano degli intervalli di tempo variabili tra un'iterazione e l'altra.

Un'importante differenza tra la simulazione sincrona e asincrona consiste anche nel fatto che la descrizione della rete nel secondo caso dev'essere notevolmente piu' dettagliata. Nella simulazione sincrona infatti un flip-flop puo' essere considerato un unico elemento, mentre nella simulazione asincrona deve essere trattato come un insieme di 8-10 gate.

Per tutte le ragioni esposte il tempo macchina in una simulazione asincrona e' da 10 a 100 volte superiore di quello di una simulazione sincrona condotta sulla stessa rete logica.

Per la simulazione asincrona sono possibili vari approcci; se si ha un accurato modello per i gate, simile a quello illustrato schematicamente in fig. 11.5.2 (a), si puo' in pratica ottenere l'andamento della tensione di uscita tra i due livelli logici in modo continuo.

L'utilizzo di modelli di questa struttura, di natura certamente complicata, si rivela particolarmente utile quando vi e' la possibilita' di corse e di alee. In caso contrario ci si puo' rivolgere a modelli semplificati del tipo di quello illustrato in fig. 11.5.2 (c).

Con riferimento alla fig. 11.5.2 (b) il blocco indicato come matrice di interconnessione altro non e' se non una rete di connessioni e in un programma di calcolo tale funzione viene realizzata descrivendo in qualche maniera il collegamento tra la matrice che contiene l'uscita dei vari gate e la matrice degli ingressi degli stessi gate. Non si tentera' in questa sede di descrivere come cio' venga effettivamente fatto, in quanto le varie soluzioni fino ad oggi proposte sono molto varie e non riconducibili ad un'unica descrizione.

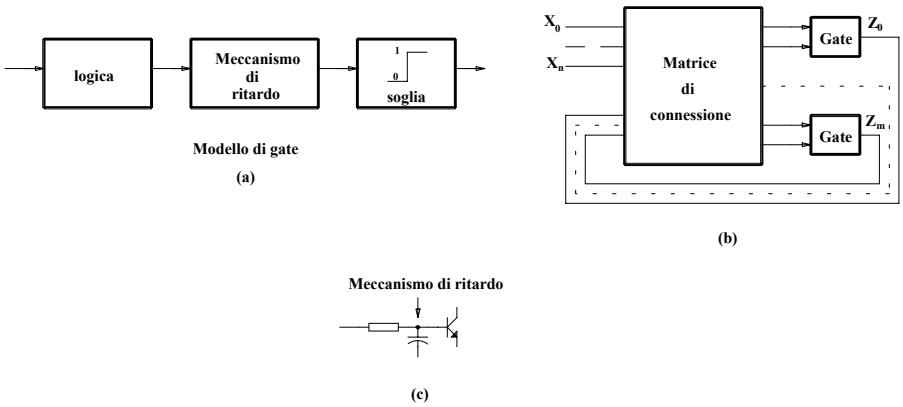


figura 11.5.2

Un possibile modello per il singolo gate, che pur non essendo particolarmente complicato, fornisce tuttavia dei risultati apprezzabili verra' ora descritto a scopo esemplificativo. Esso richiede tre passi di calcolo durante ciascuna iterazione. Durante il primo passo viene calcolata la funzione logica specificata per il gate in corrispondenza al valore attuale degli ingressi. Nel secondo passo viene usato un opportuno modello di ritardo, mentre nel terzo passo viene calcolato il nuovo valore dell'uscita.

Si ricordi che il ritardo e il tempo di salita di qualsiasi circuito reale e' influenzato da un certo numero di costanti di tempo. Nel modello che si sta considerando, e che, come si e' detto, e' particolarmente semplice, il tempo di salita viene completamente trascurato. Cio' corrisponde a dire che l'uscita del gate passa da 0 a 1 e viceversa in un'unica iterazione. Si suppone inoltre che il ritardo sia controllato da un'unica costante di tempo, come avviene ad esempio per il modello di fig. 11.5.2 (c), in cui e' previsto un condensatore connesso tra base del transistore e massa.

Una discreta approssimazione di questo modo di funzionamento, che puo' essere considerato un'integrazione non lineare, e' il meccanismo di conteggio avanti-indietro con saturazione illustrato in fig. 11.5.3.

y	S^n	S^{n+1}
0	S_{min}	S_{min}
0	$>S_{min}$	-1
1	$<S_{max}$	+1
1	S_{max}	S_{max}

contatore di ritardo

S^{n+1}	z
$S^{n+1} < 0$	0
$S^{n+1} \geq 0$	1

soglia di uscita

figura 11.5.3

S_{min} e S_{max} sono rispettivamente due interi di piccolo valore, l'uno negativo e l'altro positivo. L'effetto combinato del conteggio di ritardo e della soglia sull'uscita puo' essere spiegato come segue. Se la funzione logica y e' all'iterazione n, per gli ingressi correnti, zero il conteggio S viene diminuito , mentre se e' al valore 1 viene incrementato. Come e' specificato nella fig. 11.5.3 il conteggio S non puo' superare S_{max} o assumere valori inferiori a S_{min}.

Per quanto riguarda l'elemento a soglia presente in uscita, esso opera in modo che quest'ultima e' 0 se S e' minore di zero, 1 in caso contrario.

Si supponga da esempio che S sia inizialmente saturato a S_{min} e di conseguenza che l'uscita z sia 0. Quando y passa a 1 saranno necessarie ancora S_{min} iterazioni prima che z commuti. Un ragionamento analogo si puo' condurre a partire da S_{max}.

S_{min} e S_{max} rappresentano quindi i valori nominali del ritardo a partire dai due livelli logici dell'uscita, espressi in numero di passi di iterazione.

Il modello di gate realizzato nel modo appena illustrato possiede parecchie tra le piu' importanti proprieta' di un gate reale. Ad esempio, gli impulsi molto stretti vengono cancellati dal circuito. Il suo principale vantaggio e' la semplicita' e l'efficienza; puo' infatti essere implementato su macchine di piccole dimensioni con aritmetica in virgola fissa operante su parole da 1 byte e non richiede moltiplicazioni e/o divisioni.

Ciascuna iterazione operante sull'intera rete si puo' ritenere caratterizzata dal diagramma di flusso di fig. 11.5.4. E' importante osservare che ciascun passo deve essere terminato per ciascun gate prima che si possa passare al passo successivo. Pertanto tutti gli ingressi di tutti i gate devono rimanere inalterati finche' tutte le uscite non sono state ricalcolate. I gate quindi possono essere presi in considerazione in un ordine qualsiasi senza tenere in alcun conto le interconnessioni della rete stessa.

Dopo un certo numero K di iterazioni il programma attiva una routine di supervisione che opera la variazione degli ingressi x_1 ..... x_n ed esegue tutta una serie di operazioni utili ai fini informativi.

Si tenga tuttavia presente che la simulazione in modalita' asincrona puo' essere affrontata in molti modi. Quanto e' stato descritto e' solamente un tentativo di illustrare come tale simulazione possa essere condotta anche usando un modello di gate estremamente semplice.

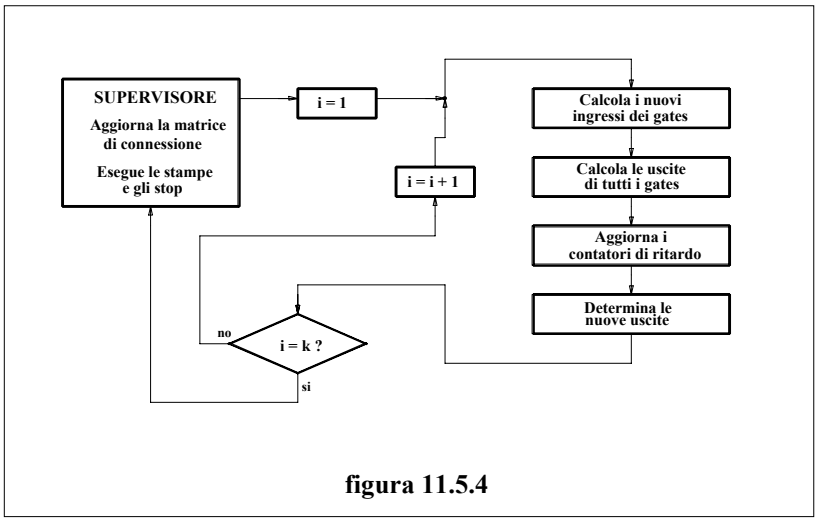
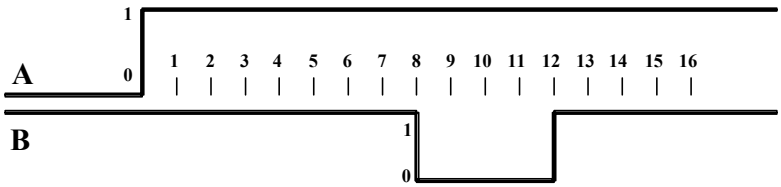


figura 11.5.4

A titolo di esempio si prenda in considerazione la simulazione asincrona di un gate AND a due ingressi A e B, supponendo che sia $S_{max} = 2$ e $S_{min} = -3$. Gli ingressi siano quelli illustrati di seguito.



Si assuma che inizialmente sia $S = S_{min}$. Il valore di S inizia a cambiare quando A passa a 1 all'iterazione iniziale. L'uscita commutera' invece all'iterazione 3 come illustrato in fig. 11.5.5, mentre S continuera' ad aumentare fino a giungere alla saturazione all'iterazione 5. Si noti che la variazione dell'uscita non avra' alcun effetto su alcun altro gate della rete fino alla quarta iterazione. All'ottava iterazione l'impulso stretto presente all'ingresso B inizia a farsi sentire. E' interessante notare che il corrispondente impulso di uscita e' piu' stretto di quello di ingresso, cosa che nella realta' avviene normalmente per impulsi piu' stretti che non due volte il tempo di ritardo di un gate.

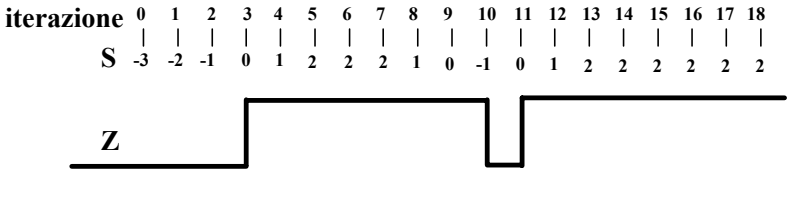


figura 11.5.5

11.6) Generazione delle sequenze di test.

La fase di test per circuiti a componenti discreti o per circuiti integrati SSI e' di solito considerata un problema collaterale e poco importante nell'ambito del processo di produzione. Il problema si presenta invece notevolmente piu' complesso quando si tratta di verificare circuiti MSI o LSI.

Si consideri infatti che in reti anche solo di 100 gate le interconnessioni possono essere diversissime, che il circuito puo' essere puramente combinatorio oppure sequenziale e in quest'ultimo caso puo' essere sincrono o asincrono. Infine che nella maggior parte dei casi il numero degli ingressi e' maggiore del numero di uscite.

La rappresentazione a "black box" di fig. 11.6.1 di un circuito con 10 ingressi e 4 uscite puo' sembrare poco significativa, ma somiglia moltissimo agli involucri dei circuiti integrati cosi' come si presentano alla fase di test. In particolare risulta evidente che le uniche misure che si possono fare sono quelle sui terminali di uscita z_1 ..... z_4.

Pertanto, se vi e' un guasto nella rete, e' necessario far si' che il suo effetto compaia su una di queste linee di uscita. In altre parole si deve far si' che l'uscita di una rete difettosa sia diversa da quella che si ha in corrispondenza agli stessi ingressi su una rete che non presenti

guasti. I segnali possono evidentemente essere introdotti nella rete solamente attraverso le linee di ingresso x_1, ... ,x_{10}.

Si supponga per semplicita' che la rete di fig. 11.6.1 sia puramente combinatoria, rappresentabile cioe' con quattro funzioni booleane da 10 variabili. Il modo piu' completo per verificare la correttezza delle uscite sarebbe quello di applicare alla rete tutte le 1024 possibili configurazioni degli ingressi. Definendo come singolo test l'operazione consistente nell'applicare una particolare combinazione all'ingresso e verificare la corrispondente uscita, sarebbero pertanto necessari 1024 test.

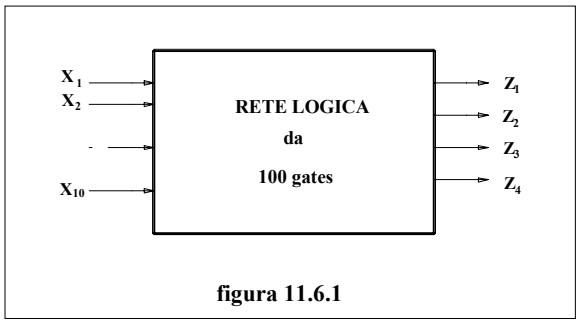


figura 11.6.1

Si supponga ora che in un qualche modo sia stato possibile determinare che nella rete di fig. 11.6.1 non si possano verificare piu' di 500 guasti diversi. Poiche' la rete e' combinatoria, per ogni possibile guasto e' determinabile una configurazione di ingresso che lo mette in evidenza. Non dovrebbero pertanto essere necessari piu' di 500 test per verificare completamente il circuito. C'e' inoltre da dire che una configurazione di ingresso e' di solito in grado di mettere in evidenza non un solo guasto, ma un'intera classe e quindi il numero di test strettamente necessari dovrebbe essere considerevolmente inferiore a 500.

Varie metodologie sono state messe a punto per trovare, sia pure in maniera approssimata, l'insieme minimo dei test necessari per la verifica completa di una rete combinatoria [(*)]. Uno dei metodi migliori a tutt'oggi conosciuto e' l'algoritmo D sviluppato da J.P.Roth. Non si ritiene opportuno presentare dettagliatamente il metodo in questa sede. Per maggiori chiarimenti si rimanda il lettore a [(**)]

ESEMPIO

Si supponga che il gate AND piu' a sinistra di fig. 11.6.2 abbia l'ingresso B fissato a 0 a causa di un guasto (Stuck at zero, SA0). L'uscita di tale gate sara' quindi in ogni condizione 0.

L'uscita dello stesso gate in una rete priva di guasti sarebbe 1 solamente per A = B = 1. Pertanto A e B dovranno essere uguali a 1 in ogni test che voglia mettere in luce questo guasto. Assumendo quindi A = B = 1, si fissi ad un valore indeterminato, rappresentato con X, l'uscita del gate soggetto al guasto, indicando in tal modo che il valore in tale punto differisce nella rete difettosa e in quella priva di guasti.

(*) **H.Y.Chang - E.G.Manning - G. Metze** " Fault diagnosis of digital systems"
Wiley - Interscience - N.Y. 1970

(**) **J.P.Roth - W.G.Bouricius - P.R.Schnieder** " Programmed Algorithms to compute test to detect and distinguish between Failures in Logic Circuits"
IEEE Transactions on Electronic Computers - EC 16. - 567 - 579 (ottobre 1967)

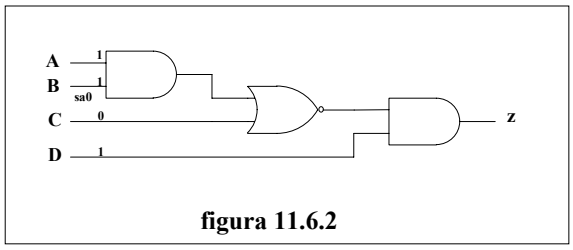


figura 11.6.2

E' necessario ora scegliere per C e D dei valori che permettono la propagazione del valore indeterminato all'uscita. Usando la logica a tre valori introdotta al capitolo IX si vede che l'uscita del gate NOR sara' indeterminata solo se C = 0 e che l'uscita z del circuito sara' indeterminata solo se D = 1.

Si e' in tal modo individuata la configurazione di test per il guasto preso in considerazione. Si noti tuttavia che la stessa configurazione di test puo' mettere in evidenza anche i seguenti guasti:

1) **SA0 dell'altro ingresso del gate all'estrema sinistra**
2) **SA0 dell'ingresso superiore del gate NOR**
3) **SA1 dell'uscita del gate NOR**
4) **SA1 dell'ingresso superiore del gate AND piu' a destra**.

Nell'esempio considerato sono stati presi in esame solo guasti il cui modello puo' essere realizzato fissando un ingresso o un'uscita a uno dei due livelli logici (stack at ...). E' questo che in pratica ancor oggi si fa. In piu' si suppone nello sviluppo di un set di configurazioni di test che non vi sia mai piu' di un guasto al medesimo tempo. Quest'ultima assunzione puo' in parte essere giustificata dalle seguenti considerazioni:

1) L'insieme completo dei guasti multipli e' comunque meno probabile di un particolare guasto singolo.

2) Massicci guasti multipli derivanti da problemi nel processo di fabbricazione possono venir facilmente rivelati da un qualsiasi insieme di configurazioni di test.

Tuttavia la piu' convincente giustificazione dell'ipotesi di guasto singolo e' che questa e' la piu' semplice e forse l'unica maniera di procedere. Infatti in un circuito che puo' avere 500 guasti singoli vi sono 2^{500} possibilita' di avere un guasto multiplo e quindi l'enumerazione completa non e' praticabile.

Ancora piu' complessa e' la situazione quando si prendono in considerazione reti sequenziali. Non e' possibile infatti provare tutte le combinazioni di ingresso realizzabili, in quanto l'uscita non e' solo funzione dell'ingresso, ma anche dello stato e per condurre un circuito sequenziale in un particolare stato puo' essere necessaria una sequenza anche molto lunga di ingressi. Al giorno d'oggi ancora non e' stata determinata una procedura algoritmica per calcolare le sequenze ottime di test per un circuito sequenziale. Il problema e' molto complesso e molto probabilmente il miglior approccio e' quello della simulazione interattiva. Le sequenze di ingresso potrebbero essere suggerite dal progettista sulla base della sua completa conoscenza del funzionamento del circuito. La rete priva di guasti e quelle con

guasti singoli vengono simulate sulla base di queste sequenze; molto probabilmente la prima sequenza simulata non riuscira' a riportare in uscita l'effetto di tutti i guasti ipotizzati. Il progettista usera' allora il risultato di questa prima simulazione assieme alla sua conoscenza del funzionamento per suggerire la continuazione della sequenza. A questo punto la simulazione verra' continuata e cosi' via finche' non si sara' ottenuta la copertura di tutti i guasti singoli o di una percentuale di guasti che possa essere ritenuta accettabile. La simulazione puo' evidentemente essere sia sincrona che asincrona. Quella sincrona puo' far uso della logica a tre valori per manipolare livelli indeterminati, quali sono quelli che si possono presentare nella simulazione di elementi di memoria difettosi.

E' bene infine osservare che all'inizio di un processo di test lo stato iniziale della rete e' indeterminato. Questa situazione puo' essere simulata assegnando alle uscite degli elementi di memoria il valore indeterminato X.