Appunti di reti logiche

Ing. Luca Martini

11 aprile 2003

Capitolo 1

Reti combinatorie

Sommario

In questo breve documento mostreremo sia alcuni concetti base sulle reti combinatorie, che alcuni dei moduli combinatori più usati nella sintesi di reti logiche.

1.1 Il concetto di rete combinatoria

Useremo per i nostri scopi una logica a due stati: falso (che indicheremo con 0) e vero (che indicheremo con 1). Questo significa che dovremo utilizzare un tipo di algebra (*algebra booleana*), in cui le variabili logiche possono assumere solamente questi due valori.

Una rete combinatoria T può essere modellata come una struttura con n variabili di ingresso x_i e m variabili di uscita z_j, tale che il valore delle uscite dipende dal valore degli ingressi secondo una legge φ che associa ad ognuna delle 2^n possibili combinazioni degli ingressi una sola combinazione del valore delle uscite.

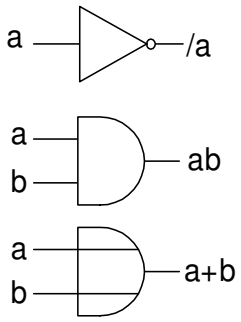
Poichè il dominio delle funzioni logiche con un numero finito di ingressi é finito, possiamo specificare φ attraverso una *tabella di verità* con 2^n righe, e per ognuna delle righe dovremo mostrare il valore delle m uscite.

Funzioni combinatorie elementari sono NOT, AND e OR (figura 1.1 nella pagina seguente).

1.1.1 Il costo di una rete combinatoria

Se vogliamo comparare l'economicità di più reti combinatorie, possiamo adottare vari criteri. In questo caso presenteremo il criterio di *costo a porte* e il criterio di *costo a ingressi*.

Il costo a porte C_g di una rete si può calcolare facendo la *somma* del numero di porte logiche elementari usate nella sintesi della rete, indipendentemente dalla loro complessità. Per porte logiche elementari intendiamo, in questo caso, porte AND, OR e i rispettivi complementi NAND e NOR. Le porte NOT non contribuiscono al calcolo di C_g. Nel caso di una rete combinatoria in forma SP



(a) Porte logiche elementari

a	$\overline{a}$
0	1
1	0

a	b	$a \cdot b$	$a + b$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

(b) Tabelle di verità

Figura 1.1: Funzioni logiche elementari

(somma di prodotti), il costo a porte C_g é dato dal numero di implicanti usati (le porte AND) più uno (la porta OR finale).

Il costo a ingressi C_i tiene parzialmente conto anche della complessità della porta utilizzata, assegnando ad ogni porta un costo pari al numero dei suoi ingressi.

Se, ad esempio abbiamo realizzato la rete descritta dalla seguente forma SP:

$$z = (\overline{a} \cdot c) + (a \cdot \overline{b} \cdot c) + (a \cdot b \cdot \overline{c})$$

avremo $C_g = 4$ (3 porte AND più una porta OR), e $C_i = 11$ (2 per il primo implicante, 3 per ciascuno degli altri due, più tre per la porta OR finale a tre ingressi).

1.2 I convertitori di codice

In questa sezione mostreremo significato e alcune implementazioni di:

- Codificatore
- Decodificatore

1.2.1 Codificatore $2^n \times n$

Un codificatore (o *coder*) é un circuito che presenta $p = 2^n$ ingressi x e n uscite z (simbolo in figura 1.2 nella pagina successiva).

Il comportamento della rete é specificato solo per quelle configurazioni con uno e un solo ingresso ad 1 e tutti gli altri a 0.

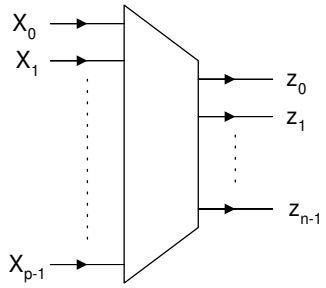


Figura 1.2: Coder $2^n \times n$

Funzionamento

Sia x_i l'ingresso posto ad 1*. Ovvero:*

$$\begin{cases} x_i = 1 & 0 \leq i \leq p-1 \\ x_j = 0 & \forall j \neq i \wedge 0 \leq j \leq p-1 \end{cases}$$

Allora le uscite sono tali da codificare *l'intero i in binario. Per esempio, se n vale* 4 *(coder* 16×4*) e i vale 9, abbiamo in uscita* $z = 0101$*, che é proprio la codifica di* 9 *in binario.*

Esempio: Codificatore $2^2 \times 2$

La tabella di verità é mostrata nella tabella 1.1.

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
	others			–	–

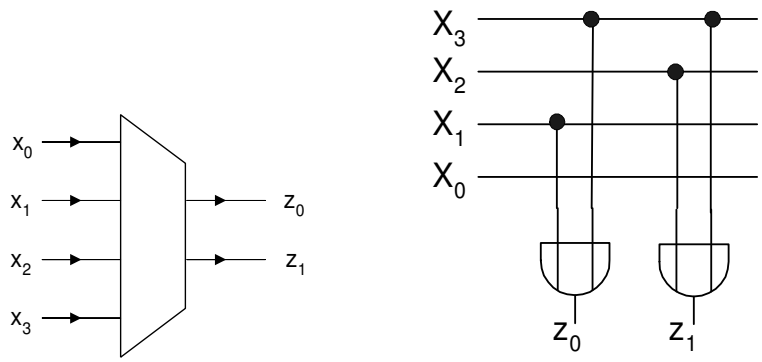
Tabella 1.1: Tabella di verità del coder 4×2

Avremo perciò:

$$\begin{cases} z_0 = x_1 + x_3 \\ z_1 = x_2 + x_3 \end{cases}$$

potendo così realizzare il coder solamente con due porte OR (vedi figura 1.3 nella pagina successiva).

Una tipica applicazione del coder si ha collegandolo in ingresso ad una serie di periferiche mutuamente esclusive, ed in uscita ad un sistema, che riconoscerà quale delle periferiche é attiva dalla codifica del suo indirizzo.



(a) (b)

Figura 1.3: Coder $2^2 \times 2$

Esempio: Codificatore $2^3 \times 3$

La tabella di verità é mostrata nella tabella 1.2.

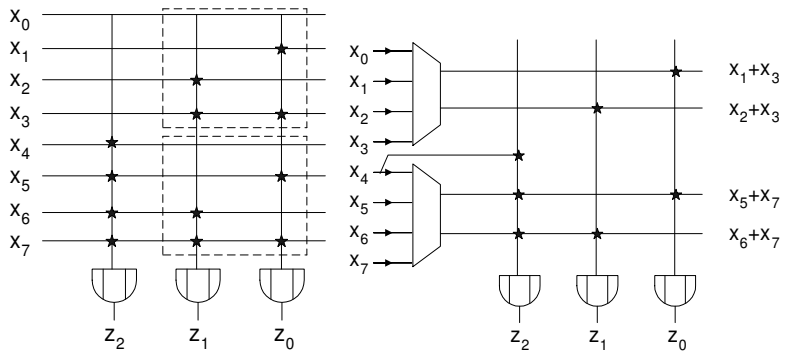
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	z_2	z_1	z_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1
			others					–	–	–

Tabella 1.2: Tabella di verità del coder 8×3

Dalla semplificazione otteniamo:

$$\begin{cases} z_0 = x_1 + x_3 + x_5 + x_7 \\ z_1 = x_2 + x_3 + x_6 + x_7 \\ z_2 = x_4 + x_5 + x_6 + x_7 \end{cases}$$

come si può notare, aumentando il numero degli ingressi possono sorgere dei problemi di fan-in. Per ovviare possiamo utilizzare una struttura modulare, riconoscendo all'interno della figura 1.4(a) due codificatori 4×2, ottenendo così la struttura in figura 1.4(b).



(a) (b)

Figura 1.4: Coder $2^3 \times 3$ realizzato modularmente

1.2.2 Decodificatore $n \times 2^n$

Un decodificatore (o *decoder*) é un circuito che presenta n ingressi x e $p = 2^n$ uscite z (simbolo in figura 1.5). Contrariamente al coder, tutte le configurazioni di ingresso sono possibili.

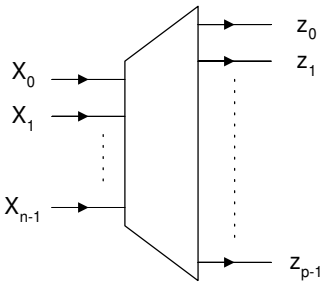


Figura 1.5: Decoder $n \times 2^n$

Funzionamento

Il decoder realizza la funzione inversa del coder; infatti interpreta l'insieme dei 2^n ingressi come un numero i espresso in codifica binaria, il cui valore é perciò compreso fra 0 e $n-1$, pone ad 1 la i-esima linea di uscita, e mantiene a 0 tutte le altre.

Esempio: Decodificatore 2×2^2

La tabella di verità é mostrata nella tabella 1.3 nella pagina seguente.

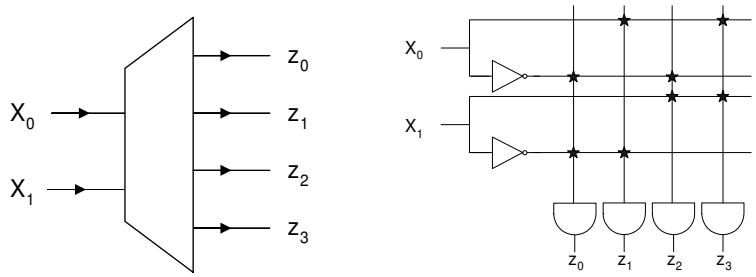
x_1	x_0	z_3	z_2	z_1	z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Tabella 1.3: Tabella di verità del decoder 2×4

Avremo perciò:

$$\begin{cases} z_0 = \overline{x_1 x_0} \\ z_1 = \overline{x_1} x_0 \\ z_2 = x_1 \overline{x_0} \\ z_3 = x_1 x_0 \end{cases}$$

Otteniamo la struttura di figura 1.6.



(a) Simbolo (b) Realizzazione

Figura 1.6: Decoder 2×2^2

Più generalmente, avremo un mintermine di ordine n per ogni uscita: il decoder $n \times 2^n$ può perciò essere realizzato con n porte AND, ciascuna ad n ingressi.

Abbiamo adesso tutti gli strumenti per poter costruire un semplice modello con cui realizzare qualsiasi funzione combinatoria. Sia nota la funzione combinatoria:

$$\begin{cases} z_0 = \varphi_0 \left(x_{n-1}, x_{n-2}, \cdots, x_1, x_0 \right) \\ z_1 = \varphi_1 \left(x_{n-1}, x_{n-2}, \cdots, x_1, x_0 \right) \\ \vdots \\ z_{m-1} = \varphi_{m-1} \left(x_{n-1}, x_{n-2}, \cdots, x_1, x_0 \right) \end{cases}$$

é possibile realizzare la rete combinatoria che la implementa grazie a:

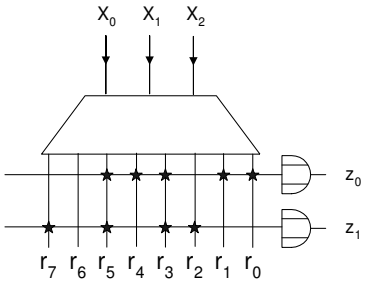
- un decoder $n \times 2^n$

- m porte OR con al più 2^n ingressi ciascuna.

Consideriamo la variabile di uscita z_j. Per ogni configurazione riconosciuta dalla funzione φ_j si collega l'uscita del decoder relativa con l'ingresso della j-esima porta OR. Nella figura 1.7 sono rappresentate una funzione combinatoria di esempio a 3 ingressi e 2 uscite e la sua implementazione con il modello sopra descritto. Questo modello realizza reti combinatorie effettuando però una sintesi

x_2	x_1	x_0	z_1	z_0
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	0

(a) Tabella di verità



(b) Realizzazione

Figura 1.7: Esempio di funzione combinatoria realizzata con decoder e porte OR

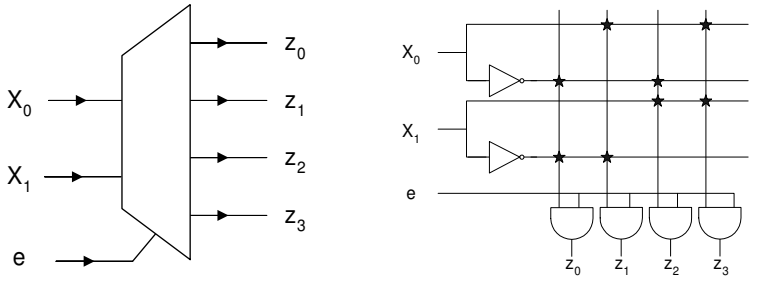
a costo non minimo. Per ottenere una rete a costo minimo si può sintetizzare la funzione combinatoria utilizzando il metodo delle mappe di Karnaugh.

Possiamo aggiungere al decoder $n \times 2^n$ un ulteriore ingresso e, detto abilitazione, il cui valore discrimina se il decoder deve avere il normale funzionamento ($e = 1$), oppure se tutte le uscite devono essere poste a 0 indipendentemente dal valore degli ingressi ($e = 0$).

Il decoder con abilitazione può essere realizzato a partire da un decoder semplice collegando l'ingresso e a tutte le porte AND (simbolo e realizzazione di un decoder 2×2^2 con abilitazione sono nella figura 1.8 nella pagina successiva).

Un tipico utilizzo dei decoder si ha nella logica di controllo delle memorie. Modellando una memoria da 2^n locazioni come un insieme di 2^n celle, possiamo indirizzare la i-esima cella semplicemente collegando ad ogni uscita del decoder una cella e presentando in ingresso il numero i espresso nella codifica binaria (indirizzo).

Aumentando il numero degli ingressi possono sorgere problemi di fan-out: é però semplice realizzare, con una struttura ad albero, un decoder $n \times 2^n$ a partire da due decoder $n - 1 \times 2^{n-1}$ con abilitazione. Basta collegare l'ingresso corrispondente alla cifra più significativa come abilitazione ai due decoder (una volta negata e una volta no), e collegare i restanti $n - 1$ ingressi agli ingressi dei decoder (nella figura 1.9 nella pagina seguente la realizzazione di un decoder 3×2^3 a partire da due decoder 2×2^4 con abilitazione).



(a) Simbolo (b) Realizzazione

Figura 1.8: Decoder 2×2^2 con abilitazione

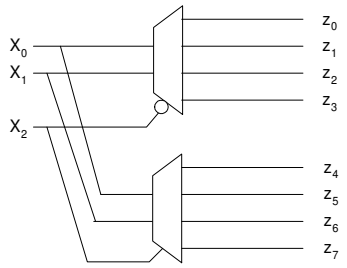


Figura 1.9: Decoder 3×2^3 realizzato modularmente

1.3 I selettori

In questa sezione mostreremo significato e alcune implementazioni di:

- Multiplexer (o *selettore di ingresso*)
- Demultiplexer (o *selettore di uscita*)

1.3.1 Multiplexer di ordine n

Un multiplexer di ordine n é un circuito combinatorio con n ingressi x, un'unica uscita z, e k variabili di comando b con:

$$k = \lceil \log_2 n \rceil$$

dove con $\lceil a \rceil$ indichiamo l'approssimazione all'intero superiore del numero reale a.

Il simbolo del multiplexer di ordine n é mostrato nella figura 1.10. Agli

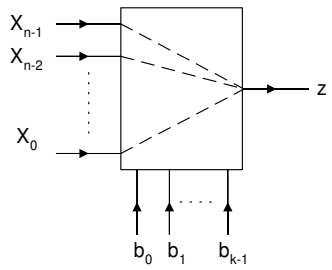


Figura 1.10: Multiplexer di ordine n

effetti pratici le k variabili di comando possono essere considerate come ingressi aggiuntivi della rete.

Funzionamento

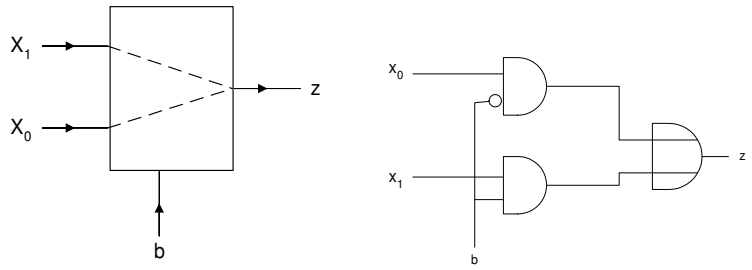
Il multiplexer seleziona l'ingresso x_i con i pari al valore delle variabili di comando, interpretate secondo la codifica binaria, e pone il valore dell'uscita pari al valore di x_i. In altre parole, le variabili di comando instradano uno degli ingressi verso l'unica uscita.

Esempio: Multiplexer di ordine 2

Implementando il multiplexer come una rete a due livelli di logica avremo(figura 1.11 nella pagina seguente):

$$z = x_0\overline{b} + x_1 b$$

Più generalmente, un multiplexer di ordine n, può sempre essere realizzato con n porte AND a $k+1$ ingressi (k per riconoscere la configurazione d'ingresso



(a) Simbolo (b) Realizzazione

Figura 1.11: Multiplexer di ordine 2

i-esima, più uno per instradare l'ingresso x_i) collegate in ingresso ad una porta OR. Ovvero:

$$z = x_0 \cdot \overline{b_{k-1}} \cdot \overline{b_{k-2}} \cdots \overline{b_0} + x_1 \cdot \overline{b_{k-1}} \cdot \overline{b_{k-2}} \cdots \overline{b_1} \cdot b_0 + \cdots + x_{n-1} \cdot b_{k-1} \cdot b_{k-2} \cdots b_0$$

Per risolvere eventuali problemi di fan-in é possibile realizzare il multiplexer modularmente utilizzando:

- un decoder $k \times 2^k$;
- n porte AND a 2 ingressi;
- una porta OR a n ingressi

Nella figura 1.12 un multiplexer a 8 ingressi sintetizzato con questa tecnica.

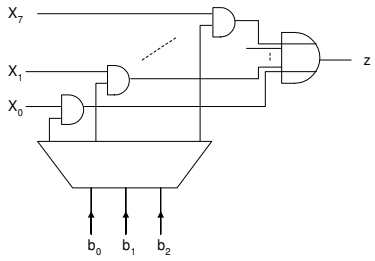


Figura 1.12: Multiplexer a 8 ingressi

Il multiplexer di ordine n può essere usato per sintetizzare una qualsiasi funzione combinatoria φ a k variabili. Basta infatti collegare alle variabili di comando del multiplexer gli ingressi di φ, e agli ingressi x_i del multiplexer costanti 1 e 0 a seconda che, rispettivamente, la configurazione rappresentata dall'intero

i sia riconosciuta o meno da φ. Nella figura 1.13 mostriamo l'implementazione della funzione combinatoria illustrata nella tabella di figura 1.7a, relativamente alla varibile di uscita z_0, con questo approccio.

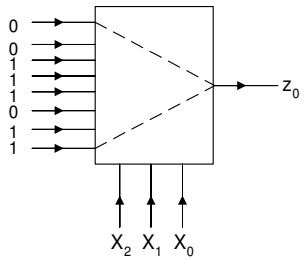


Figura 1.13: Esempio di uso del multiplexer per la generazione di funzioni combinatorie

Un'altra applicazione del multiplexer può essere la conversione di una linea parallela a n vie in una linea seriale. Ipotizziamo di avere $n = 8$ (e quindi $k = 3$). Collegando le 8 vie della linea parallela agli ingressi del multiplexer, e facendo variare sequenzialmente le variabili di comando da 000 a 111 si trasmette serialmente sull'uscita il contenuto delle 8 vie.

1.3.2 Demultiplexer 1×2^n

Ipotizziamo di avere un decodificatore $n \times 2^n$ con abilitazione. Consideriamo la variabile e come l'unico ingresso della rete (rinominato come x) e le vecchie variabili di ingresso x_i come variabili di comando (le rinominiano b_i). Otteniamo in questo modo un circuito che realizza la funzione inversa del multiplexer: lo chiameremo *demultiplexer* e lo indicheremo con il simbolo di figura 1.14.

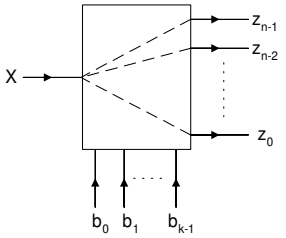


Figura 1.14: Demultiplexer 1×2^n

Funzionamento

Il funzionamento del demultiplexer consiste nell'instradare il valore dell'unico ingresso x sull'uscita specificata dalle variabili di comando, mentre

tutte le altre uscite rimangono a zero. L'ingresso viene instradato sull'uscita z_p se e solo se il valore delle variabili di comando $b_{k-1} \cdots b_0$ rappresenta la codifica binaria dell'intero p.

Caso $n = 1$

Abbiamo:

$$\begin{cases} z_0 = \overline{b} \cdot x \\ z_1 = b \cdot x \end{cases}$$

Nel caso più generale avremo:

$$z_i = x \cdot p_i \quad \text{con} \quad p_i = b^*_{k-1} \cdots b^*_0, \quad b^*_j \in \left\{b_j, \overline{b_j}\right\}$$

Esistono versioni di multiplexer e demultiplexer che lavorano smistando blocchi di dati anzichè singoli segnali logici. Le implementazioni si ottengono facilmente parallelizzando i circuiti visti finora.

1.4 Fenomeni transitori nelle reti combinatorie

Fino ad adesso abbiamo sorvolato sui fenomeni connessi al tempo di risposta di una rete combinatoria: in effetti, prima che una rete combinatoria fornisca l'uscita prevista può accadere che passi un tempo indefinito, durante il quale l'uscita assume valori in genere non desiderati. Questo puó accadere per vari motivi connessi al pilotaggio della rete: si fanno transire gli ingressi in maniera troppo veloce, senza dare il tempo ai circuiti di andare a regime, oppure si fa transire più di un'ingresso alla volta, presentando perciò alla rete degli ingressi spuri (sarà in generale impossibile far cambiare di valore due variabili esattamente nello stesso istante).

Possono però verificarsi oscillazioni non desiderate in uscita (*alee*) anche nel caso di reti ben pilotate. Prendiamo ad esempio una rete a due livelli di logica realizzata come circuito di tipo SP. Se due configurazioni di ingresso adiacenti non hanno un implicante che li copre entrambi, ma tutti e due danno in uscita 1, può accadere che per un breve tempo in ingresso alla porta OR finale si abbia una configurazione di tutti 0 producendo perciò in uscita un'alea sul livello basso. Per rimediare possiamo implementare la rete non a costo minimo, inserendo un'implicante che copra la transizione tra i due ingressi adiacenti.

Vedremo (paragrafo 2.3 a pagina 20) come progettare reti combinatorie prive di alee sia importante al fine di sintetizzare reti sequenziali che funzionino in maniera corretta.

1.5 Altri circuiti combinatori

1.5.1 Le ROM

Le ROM (*Read Only Memory*) sono delle *memorie* (dispositivi in grado di mantenere l'informazione) *non volatili* (possono mantenere l'informazione anche se

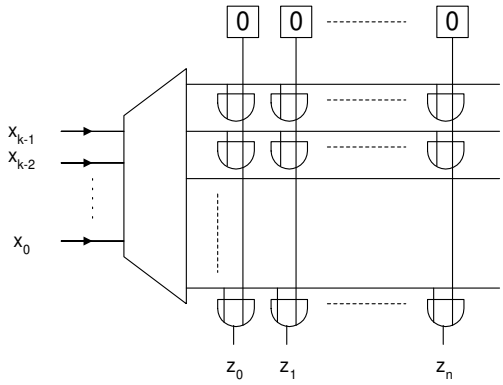


Figura 1.15: ROM $2^k \times n$ realizzata con decoder e porte OR

non alimentate) *di sola lettura*. Questo significa che il loro contenuto informativo é stabilito una volta per tutte all'atto della fabbricazione.

In una ROM $2^k \times n$ sono memorizzate 2^k stringhe binarie (dette *parole*) di n bit. É possibile accedere all'i-esima parola presentando come valore delle k variabili di ingresso la codifica binaria dell'intero i. In pratica possiamo considerare una ROM $2^k \times n$ come una rete combinatoria con k ingressi e n uscite. Una possibile implementazione é visibile in figura 1.15. Fa uso di un decoder e di una matrice di porte OR $2^k \times n$. Inizialmente tutte le porte OR sono presenti, ma durante la fabbricazione alcune vengono distrutte, per realizzare il particolare contenuto della memoria. In maniera piú specifica la presenza o meno della porta OR all'incrocio (i, j) determina se il bit j-esimo della parola i-esima vale, rispettivamente 1 o 0.

Con una ROM $2^k \times n$ opportunamente realizzata é possibile implementare fino a n qualsiasi funzioni combinatorie a k variabili. Oltre che per implementare funzioni combinatorie, le ROM sono anche utilizzate per contenere porzioni di codice (sola lettura), come per esempio i BIOS di alcuni dispositivi.

Altri tipi di memorie

Per ovviare al fatto che le ROM non sono affatto personalizzabili sono state create anche i sguenti tipi di memorie:

PROM Le PROM (*Programmable ROM*) sono delle ROM che però possono essere scritte (anche se una volta sola). Sono realizzate con un decoder e una matrice di fusibili. I fusibili della i-esima parola possono essere distrutti selezionando l'i-esima riga della matrice e applicando una tensione opportuna ad un terminale di programmazione.

EPROM Le EPROM (*Erasable PROM*) sono delle PROM che possono anche essere cancellate esponendo il chip (racchiuso in uno speciale case con una finestrella al quarzo) ad una forte luce ultravioletta.

EEPROM Le EEPROM (o E^2PROM - Electrically Erasable PROM) sono delle EPROM che hanno il vantaggio di poter essere cancellate applicando un opportuno voltaggio. In questo modo, rispetto alle EPROM, si aumenta la facilità d'uso e si diminuisce il costo del packaging del circuito.

1.5.2 Matrici logiche programmabili

Le matrici logiche programmabili (o *PLA - Programmable Logic Arrays*) sono dei circuiti combinatori a due livelli di logica. Nel caso di PLA and-or a n ingressi e ad m uscite il primo livello é costituito da un certo numero k di porte and ciascuna con al più $2n$ ingressi (gli ingressi del PLA più i loro complementi). L'uscita di ognuna delle porte and é collegata in ingresso a m porte or (figura 1.16).

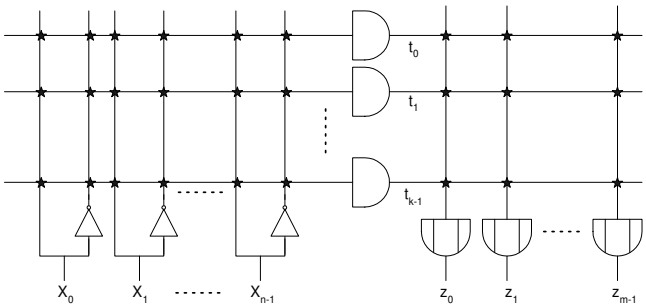


Figura 1.16: PLA and-or a n ingressi, m uscite e k termini

Ciascun collegamento (rappresentato in figura da un asterisco) é in realtà un fusibile, che può essere distrutto in fase di programmazione del PLA. In questa maniera è possibile realizzare la sintesi di funzioni combinatorie (compatibilmente col dimensionamento del PLA) non solo in forma canonica, come avevamo visto con il decoder (figura 1.7), con il multiplexer (figura 1.13) e con le PROM, ma anche in *forma minima*. Le condizioni da rispettare per implementare una funzione combinatoria generica φ con un PLA con le caratteristiche sopra mostrate sono le seguenti:

- Numero di ingressi di φ minore di n;
- Numero di uscite di φ minore di m;
- Somma del numero di implicanti distinti necessari per implementare φ minore di k;

Con un PLA come quello di figura 1.16 é facile realizzare reti combinatorie sintetizzandole come somme di prodotti (SP), mentre, dovendo effettuare una sintesi PS (prodotti di somme), é necessario utilizzare PLA or-and.

1.6 Oltre la logica

Fino ad ora abbiamo considerato le reti logiche esclusivamente come oggetti ideali, solamente da un punto di vista schematico, senza preoccuparci di come siano realizzate nella pratica. Pur senza scendere in dettagli che riguardano più propriamente l'elettronica digitale, é necessario spendere qualche parola in merito.

In realtà i livelli logici viaggiano sulle linee sotto forma di tensioni: se una tensione é compresa in un range di valori (fissato, per esempio, da 0 a 2V), considereremo quella linea al livello logico basso. Viceversa se la tensione ricade in un altro range (per esempio da 3 a 5 V), considereremo quella linea al livello logico alto.

Le porte logiche sono dispositivi elettronici che necessitano di una alimentazione elettrica. Quindi, oltre ai terminali logici, avranno anche due terminali aggiuntivi (elettrici): alimentazione (di solito indicata con V_{CC}), e massa (indicata con GND, *ground*).

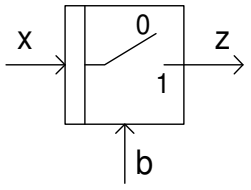
1.6.1 Le porte three-state

Le porte *three-state*, non rientrano nella classe delle reti logiche, in quanto introducono un 'terzo stato', che non é logico ma elettrico. Nella figura 1.17 sono mostrati simbolo e tabella di verità di una porta three-state.

Funzionamento

> *Quando la variabile di comando b é posto ad 1, l'uscita z si allinea all'ingresso x. Se la variabile di comando vale 0, l'uscita viene posta in* alta impedenza.

Lo stato di alta impedenza é identificato nella tabella dalla lettera Z. Se una porta mette una linea in alta impedenza, allora significa che quella porta *non impone alcun valore logico* su quella linea.



(a) Simbolo

x	b	z
0	1	0
1	1	1
–	0	Z

(b) Tabella di verità

Figura 1.17: La porta three-state

Le porte three-state sono utili quando si vogliano realizzare delle linee bidirezionali (vedi fig. 1.18 nella pagina successiva): con la variabile di comando si discrimina chi é ad imporre il livello di tensione (e quindi logico) sulla linea.

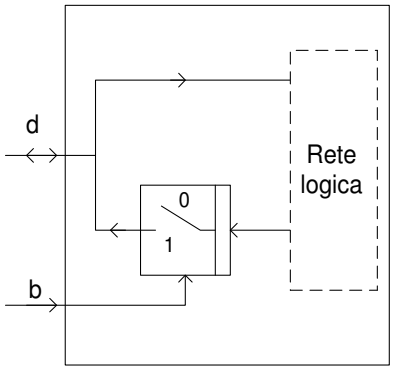


Figura 1.18: Linee bidirezionali

Capitolo 2

Reti sequenziali asincrone

Sommario

In questa sezione affronteremo lo studio delle reti sequenziali che, a differenza delle reti combinatorie, hanno capacitá di memoria. Mostreremo alcune delle reti sequenziali asincrone più usate.

2.1 Il concetto di rete sequenziale

Una rete sequenziale é una rete logica con n variabili di ingresso x_i, m variabili di uscita z_j, e uno stato interno t. Lo stato interno assume uno dei k possibili valori dell'insieme $\mathcal{S}$.

Sia $\mathcal{X}$ l'insieme dei possibili valori degli ingressi e $\mathcal{Z}$ l'insieme dei possibili valori delle uscite: allora t é calcolato attraverso una funzione F, mentre il valore delle uscite attraverso una funzione G. Ad ogni istante il valore dello stato interno viene aggiornato applicando la F allo stato interno precedente e agli ingressi, mentre l'uscita viene calcolata applicando la G.

La presenza dello stato interno differenzia *concettualmente* le reti combinatorie dalle reti sequenziali: mentre infatti le uscite delle prime dipendono solamente dal valore degli ingressi, per calcolare l'uscita delle seconde occorre conoscere anche il valore dello stato interno, ovvero un informazione che dipende dalla storia (*sequenza*) degli ingressi precedenti.

2.2 Il latch SR

Come primo esempio di rete asincrona illustriamo il *latch SR*: esso ha due ingressi denominati s e r, ed un'uscita q (simbolo figura 2.1 nella pagina seguente).

Funzionamento

*Il latch SR si comporta nel seguente modo: se la variabile s é alta e r bassa, l'uscita viene posta al livello alto (*set*), se la variabile s é bassa e r alta, l'uscita viene posta al livello basso (*reset*); se entrambi gli ingressi*

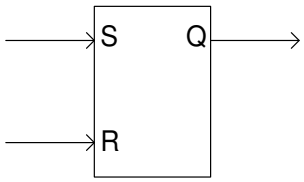


Figura 2.1: Simbolo del latch SR

sono bassi l'uscita mantiene *il valore precedente. Il comportamento della rete non é definito per* $s = 1, r = 1$.

Per descrivere il comportamento del latch SR (e in generale di una qualsiasi rete sequenziale) possiamo utilizzare un grafo (fig. 2.2(a) nella pagina successiva) oppure una tabella di flusso(fig. 2.2(b) nella pagina seguente).

Nel grafo i cerchi rappresentano gli stati, con le corrispondenti uscite, e le frecce le transizioni, in corrispondenza del valore degli ingressi. La tabella delle transizioni e quella delle uscite rappresentano la codifica della tabella di flusso.

Un'altra rappresentazione del funzionamento del latch SR si può dare con la *tabella di eccitazione* (tabella 2.1).

q_{old}	q_{new}	s	r
0	0	0	–
0	1	1	0
1	1	–	0
1	0	0	1
⏟ evoluzione		⏟ eccitazione	

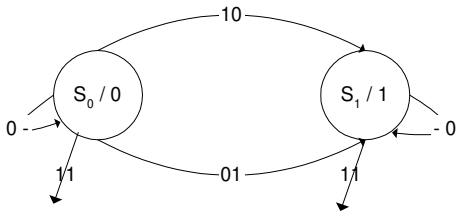
Tabella 2.1: Tabella di eccitazione del latch SR

Implementiamo adesso il latch SR usando un modello valido per qualsiasi rete sequenziale asincrona. Tale modello consiste nel codificare lo stato interno della rete con $w = \lceil \log_2 k \rceil$ bit e realizzare due reti combinatorie che sintetizzano le funzioni F e G, per poi interconnetterle come in figura 2.3 nella pagina seguente. I ritardi presenti possono essere omessi (se la rete per F introduce già un delay sufficiente).

In questo caso, codificando gli stati come nella figura 2.2(b), la rete per G non é niente altro che un'identità. Avremo perciò (sintetizzandolo rispettivamente in forma PS e SP e applicando le leggi di De Morgan):

$$y = \overline{r} \cdot (s + y) = \overline{r + \overline{s + y}}$$

$$y = s + \overline{r} \cdot y = \overline{\overline{s} \cdot \overline{\overline{r} \cdot y}}$$



(a) Grafo

	sr				q
	00	01	11	10	
S_0	S_0	S_0	–	S_1	0
S_1	S_1	S_0	–	S_1	1

y \ sr	00	01	11	10
0	0	0	–	1
1	1	0	–	1

y

Stato	y	q
S_0	0	0
S_1	1	1

(b) Tabella di flusso e tabelle delle transizioni e delle uscite di F

Figura 2.2: Il comportamento del latch SR

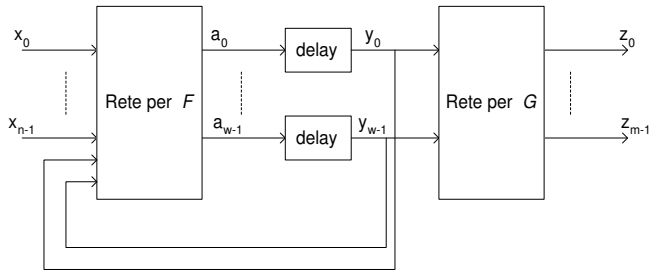
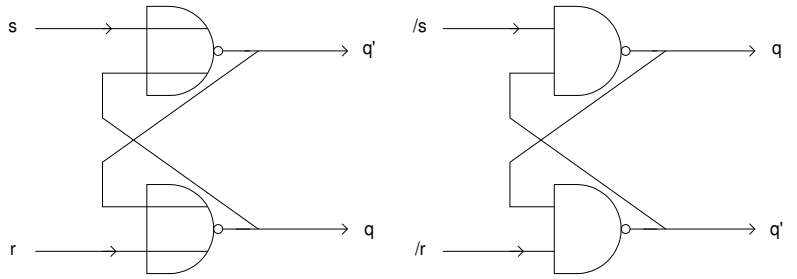


Figura 2.3: Modello per la sintesi della rete SR

Possiamo perciò realizzare il latch SR sia con il solo utilizzo di porte NOR, che con il solo utilizzo di porte NAND (fig. 2.4).



(a) Sintesi a porte NOR (b) Sintesi a porte NAND

Figura 2.4: Due implementazioni del latch SR

L'uscita aggiuntiva q', se non si presenta mai la configurazione di ingressi proibita 11, é il complemento di q.

2.3 Corretta progettazione delle reti sequenziali

Il modello di figura 2.3 necessita di ulteriori precisazioni. Infatti la rete per F deve essere priva di alee, in quanto stati spuri delle variabili a potrebbero portare la rete ad assumere degli stati interni non desiderati, *cambiandone completamente l'evoluzione*.

É necessario porre particolare attenzione anche nella scelta della codifica degli stati: poichè risulta impossibile far variare contemporaneamente due segnali logici, bisogna far sì che le codifiche di due stati comunicanti differiscano al più per un bit. Questo può essere effettuato anche aggiungendo degli stati fittizi di passaggio (che possono portare anche a rallentare la risposta della rete). A questo punto dovrebbe essere chiaro come puntare ad una soluzione a costo minimo sia nella sintesi della rete per F, che nella codifica degli stati interni, può portare la rete ad avere malfunzionamenti, pur se pilotata in modo corretto.

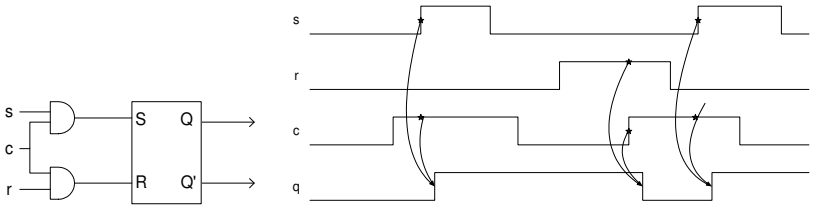
2.4 Gated latch

Esistono anche delle versioni del latch SR con un ingresso aggiuntivo c di controllo denominate *gated latch*: quando c é basso il latch é insensibile alle variazioni degli ingressi, mentre quando é alto il latch si comporta come un normale latch

SR. Avremo (fig 2.5(a)):

$$y = \overline{c \cdot r} \cdot ((c \cdot s) + y)^1$$

Come illustrato nella temporizzazione di figura 2.5(b), i latch sono sensibili *solamente al livello* e non alle variazioni dell'ingresso di controllo.

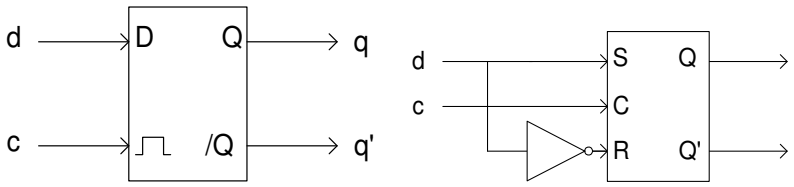


(a) Realizzazione (b) Esempio di Temporizzazione

Figura 2.5: Il gated latch

2.5 D-Latch

Un altro tipo di latch molto usato é il *D-latch* (figura 2.6(a)). Si può ottenere, ad esempio, ponendo agli ingressi di un gated latch due segnali di valore opposto (figura 2.6(b)). Poichè il gated latch non potrá mai avere in ingresso la configurazione proibita $sr = 11$, l'uscita q' é sempre il complemento di q.



(a) Simbolo (b)

Figura 2.6: Il D-latch

[1] Questa espressione é derivata direttamente dalla sintesi SP del latch SR sostituendo s e r rispettivamente con $c \cdot s$ e $c \cdot r$.

Funzionamento

*Quando l'ingresso c é alto l'uscita q insegue l'ingresso d (*fase di campionamento*), mentre, quando c é basso, q é insensibile alle variazioni di d e mantiene l'ultimo stato memorizzato prima che c andasse a zero (*fase di conservazione*).*

Un'implementazione più semplice si può ottenere sostituendo nell'espressione di sintesi del gated latch l'eguaglianza $s = \overline{r} = d$, ottenendo dopo alcune semplificazioni:

$$y = (c \cdot d) + (y \cdot \overline{c})$$

2.6 Il flip-flop JK

Per evitare di avere in ingresso al latch SR la configurazione proibita si puó modificarlo perchè, quando i due ingressi vanno entrambi ad uno, commuti il suo stato. Otteniamo così il *flip-flop JK* (simbolo in fig. 2.7(a) nella pagina seguente)

Funzionamento

Se $j = k = 0$ l'uscita q mantiene il suo valore precedente. Se $j = 1$, $k = 0$ l'uscita si porta sul livello alto. Se $j = 0$, $k = 1$ l'uscita si porta sul livello basso. Se $j = k = 1$ l'uscita cambia il suo valore (se 0 si porta ad 1, e viceversa).

La configurazione degli ingressi $j = k = 1$ é palesemente instabile e porta il flip-flop in un oscillazione fra due stati. Per questo é utilizzato esclusivamente nella sua versione sincronizzata[2].

La tabella 2.2 nella pagina successiva é la tabella delle transizioni del flip-flop JK.

Come utile esercizio implementiamo adesso il flip-flop JK utilizzando il modello esposto in figura 2.3, utilizzando però dei latch SR come ritardi nell'anello di retroazione.

Innanzitutto scriviamo la tabella di stato (fig. 2.7(b)), poi codifichiamo S_0 con 0 e S_1 con 1 per ottenere come rete per G un'identitá (sarà quindi necessario solamente un latch come marcatore). Ricordandoci la tabella delle transizioni del latch SR (tab. 2.1 a pagina 18), possiamo tradurre la tabella di stato nella tabella di fig. 2.7(c) nella pagina successiva.

Separando le uscite della rete per F (s ed r), e utilizzando poi le mappe di Karnaugh otteniamo:

$$s = j \cdot \overline{y}$$
$$r = k \cdot y$$

[2] Riguardo alle terminologie di *flip-flop* e di *latch*, alcuni autori usano i termini come sinonimi. In questo documento ci siamo invece attenuti alla convenzione del libro di testo, che indica con il nome di latch tutti quegli elementi di memoria sensibili solamente al livello degli ingressi (*level-triggered*), mentre con il nome di flip-flop indica gli elementi di memoria sensibili alle transizioni degli ingressi di sincronizzazione (*edge-triggered*, vedi capitolo successivo).

A stretto rigore di logica avremmo dovuto scrivere quindi *latch JK* e non *flip-flop JK*. In considerazione del fatto che le versioni asincrone di questi elementi vengono presentate solo a scopo didattico, utilizzeremo impropriamente il termine flip-flop, specificando però che si trattano di elementi asincroni.

q_{old}	q_{new}	j	k
0	0	0	–
0	1	1	–
1	1	–	0
1	0	–	1
evoluzione		eccitazione	

Tabella 2.2: Tabella delle transizioni del flip-flop JK

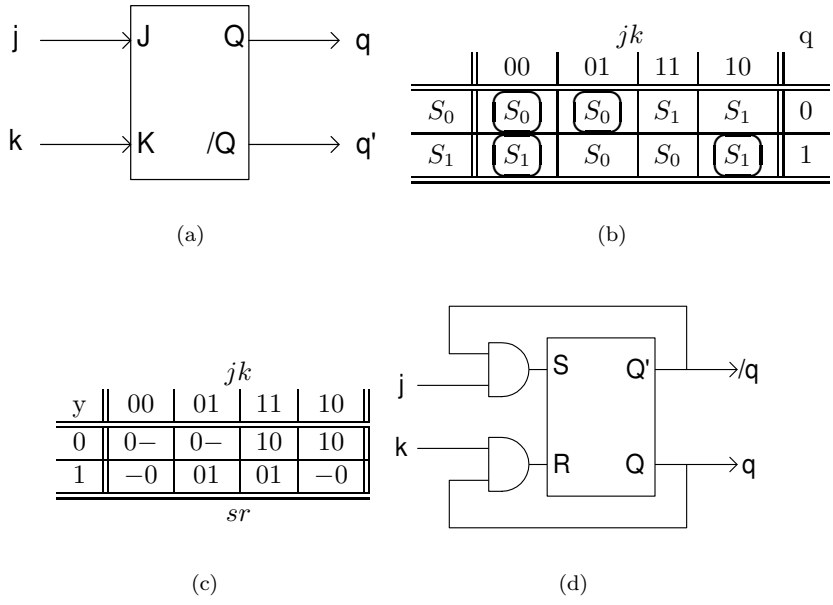


Figura 2.7: La realizzazione del flip-flop JK

Il risultato finale é mostrato in figura 2.7(d) nella pagina precedente

2.7 Il flip-flop T

Il *flip flop T* (*toggle*) si ottiene portando gli ingressi di un flip-flop JK ad avere sempre lo stesso valore (fig. 2.8(b)).

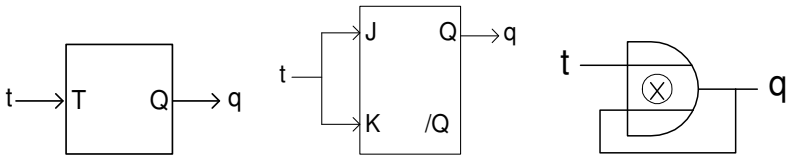
Funzionamento

Se l'unico ingresso t vale 0, l'uscita q mantiene il suo valore. Se $t = 1$ l'uscita commuta.

Per gli stessi motivi visti sopra, del flip-flop T é usata quasi esclusivamente la versione sincronizzata[3]. Poichè l'equazione caratteristica del flip-flop T é:

$$y = (y \cdot \overline{t}) + (\overline{y} \cdot t)$$

possiamo anche implementarlo attraverso una porta XOR (*exclusive OR*) posta in retroazione (fig. 2.8(c)).



(a) Simbolo (b) (c)

Figura 2.8: Il flip-flop T

[3] vedi nota 2 a pagina 22

Capitolo 3

Reti sincronizzate

3.1 Il flip-flop D edge-triggered

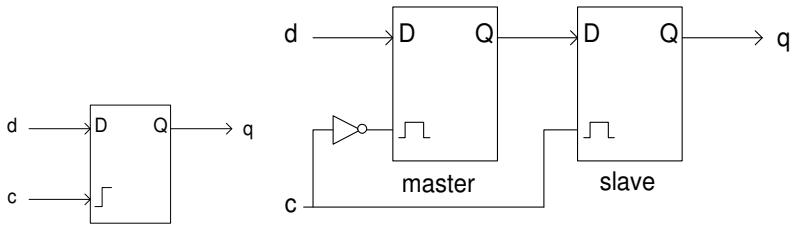
In molte situazioni c'é l'esigenza di campionare un segnale logico in un preciso istante e memorizzare il suo valore. Per tale necessità i latch non sono adatti in quanto, come già precisato, sono sensibili al livello dell'ingresso di controllo e non alle sue transizioni. Questo impedisce di poterli usare in circuiti in cui c'é retroazione delle uscite sugli ingressi. Il *flip-flop D edge-triggered* (fig. 3.1(a) nella pagina successiva) soddisfa questa esigenza, poichè campiona solamente sul fronte in salita (*positive*) o sul fronte in discesa (*negative*) dell'ingresso di sincronizzazione.

Funzionamento

> *Il flip-flop D positive edge-triggered ha due ingressi d e c, ed un'uscita q. Quando la variabile c transisce da* 0 *ad* 1*, per un piccolo intervallo di tempo (denominato T_{hold}) l'uscita q si allinea al valore di d. In tutte le altre situazioni l'uscita é insensibile alle variazioni di d (mantenimento).*

Per un corretto pilotaggio del flip-flop occorre quindi mantenere l'ingresso d costante per un tempo T_{setup} e per un tempo T_{hold} rispettivamente prima e dopo il fronte in salita di c. Definiamo anche $T_{propagation}$ il tempo necessario all'uscita per cambiare il suo valore, a partire dall'istante di salita di c.

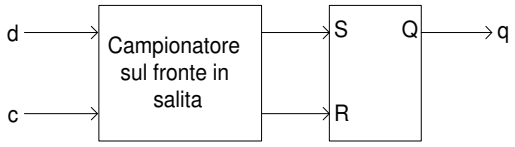
Una possibile implementazione del flip-flop D edge-triggered fa uso di due D-latch messi in configurazione master-slave (fig. 3.1(b) nella pagina seguente). Quando c vale zero il master é in campionamento e la sua uscita q_1 insegue d. Lo slave però é insensibile alle variazioni di q_1 e memorizza la vecchia uscita. Quando c va ad 1 lo slave comincia a campionare mettendo in uscita l'ultimo valore memorizzato dal master, il quale, essendo nella fase di mantenimento, é ormai insensibile alle variazioni di d. Perché questa configurazioni funzioni in modo corretto non deve mai accadere che sia il master che lo slave siano contemporaneamente nella fase di campionamento. Per questo i due segnali di clock che pilotano il master e lo slave non possono essere generati banalmente da un invertitore reale, ma occorrono tecniche leggermente più complesse, come la generazione a soglia.



(a) (b)

Figura 3.1: Il flip-flop D edge-triggered

Un'altra implementazione del flip-flop D edge-triggered può essere realizzata sfruttando il modello di figura 3.2(a); la rete campionatrice sul fronte é una rete sequenziale asincrona implementata utilizzando latch SR come ritardi. La tabella di flusso é riportata in figura 3.2(b). Codificando gli stati come evidenziato dalla tabella delle transizioni di figura 3.2(c), avremo implementato il flip-flop D-edge triggered con la rete di figura 3.3.



(a)

	$c\ d$ 00	01	11	10	$s\ r$
S_0	S_0	S_0	S_2	S_1	00
S_1	S_0	S_0	S_1	S_1	01
S_2	S_0	S_0	S_2	S_2	10

(b) Tabella di flusso della rete campionatrice

Stato	Codifica	
S_0	0	0
S_1	0	1
S_2	1	0

(c) Tabella delle transizioni

Figura 3.2: Una possibile implementazione del flip-flop D edge-triggered

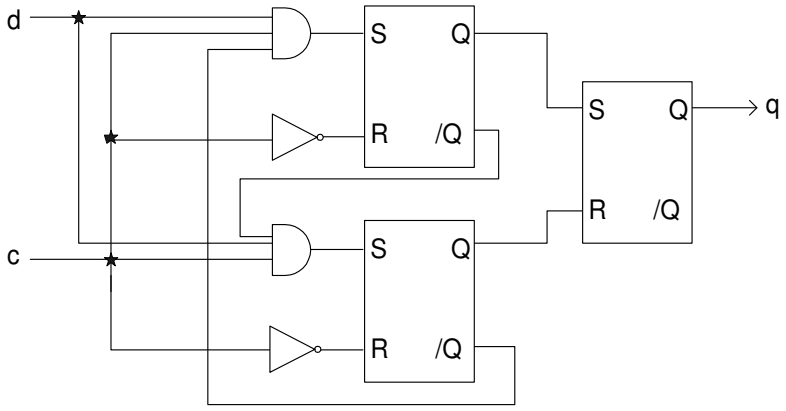


Figura 3.3: Implementazione del modello in fig. 3.2(a)

3.2 Differenze fra reti sequenziali sincrone e asincrone

Le reti sincrone hanno un'evoluzione differente rispetto a quelle asincrone. É infatti presente come ingresso una variabile logica, detta comunemente clock, di cui non ci interessa il livello, ma i fronti in salita (o in discesa). Mentre nelle reti asincrone le transizioni di stato venivano gestite autonomamente dalla rete, nelle reti sincrono sono pilotate dal clock.

Viene perciò introdotto il concetto di *ciclo di clock* ovvero la distanza temporale fra due fronti in salita successivi. Variazioni degli ingressi all'interno del ciclo non hanno effetti sullo stato della rete, ma quello che conta ai fini della generazione del nuovo stato é solamente il valore degli ingressi in prossimità dell'inizio del ciclo successivo.

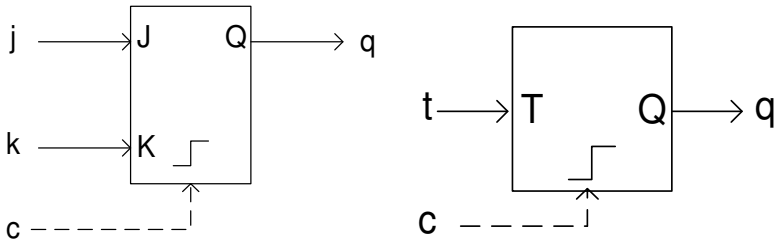
3.3 Il flip-flop JK sincronizzato

Vediamo adesso la versione sincronizzata del flip-flop JK (fig. 3.4(a) nella pagina seguente).

Funzionamento

*Il funzionamento é quello visto nel caso asincrono, con le differenze che esiste una variabile di ingresso aggiuntiva (*clock*) e che lo stato (e quindi l'uscita) può variare solamente in corrispondenza del fronte in salita del clock.*

Mostriamo un'implementazione del flip-flop JK edge-triggered basata sul modello master-slave (fig. 3.3 nella pagina successiva): fa uso di un flip-flop asincrono JK gated come master e di uno SR (sempre gated) come slave.



(a) Flip-flop JK (b) Flip-flop T

Figura 3.4: Versioni sincronizzate dei flip-flop JK e T

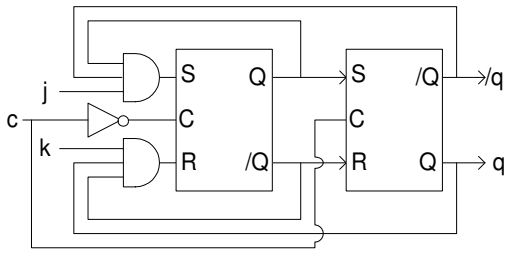


Figura 3.5: Una possibili implementazione del flip-flop JK sincronizzato

3.4 Il flip-flop T sincronizzato

Funzionamento

*Anche in questo caso il funzionamento é quello visto nel caso asincrono, con le differenze che esiste una variabile di ingresso aggiuntiva (*clock*) e che l'uscita può variare solamente in corrispondenza del fronte in salita del clock.*

Il flip-flop T sincronizzato (fig. 3.4(b) nella pagina precedente) può essere realizzato accoppiando gli ingressi di un flip-flop JK edge-triggered. Se si mantiene t permanentemente ad 1 il flip-flop agisce da divisore di frequenza.

3.5 I registri

Con la parola *registri* si comprende tutta una serie di circuiti sincronizzati, principalmente deputati alla memorizzazione dell'informazione.

Il registro più semplice possibile, che memorizza la minima quantità di informazione (un bit), può essere realizzato, per esempio, tramite un flip-flop D edge-triggered, o tramite un flip-flop JK.

Più generalmente, un registro a n bit ha la funzione di mantenere un dato, detta *parola*, che é in pratica un numero binario ad n cifre.

I registri si suddividono in paralleli e seriali, a seconda delle modalitá di ingresso/uscita dei dati.

3.5.1 I registri paralleli

I *registri paralleli* a n bit hanno n ingressi e n uscite.

Funzionamento

Sul fronte in salita del segnale di sincronizzazione memorizzano il nuovo valore della parola, prendendolo dal valore degli n ingressi. In uscita mantengono il valore dell'ultima parola memorizzata.

Normalmente sono realizzati con una batteria di n flip-flop D edge-triggered che hanno in comune il segnale di clock (figura 3.6(a) nella pagina successiva).

Possono presentare anche altri ingressi di controllo: *ld* (*load* - abilitazione degli ingressi) e *oe* (*output enable* - abilitazione delle uscite). Finchè *load* rimane non attivo, non é possibile memorizzare nuove parole nel registro. Se *oe* é basso le uscite vengono poste in alta impedenza. In figura 3.6(b) nella pagina seguente é mostrato un registro parallelo ad n bit, con ingressi di controllo, realizzato grazie a flip-flop JK.

3.5.2 I registri di traslazione

I *registri di traslazione* (shift registers) differiscono dai registri paralleli nel fatto che hanno un unico ingresso seriale che va dal bit meno significativo a quello più significativo (left-shift registers) o viceversa (right-shift registers).

Nel caso di left-shift registers, ad ogni fronte in salita del clock, il contenuto del registro viene traslato verso sinistra e l'ingresso viene memorizzato nel bit

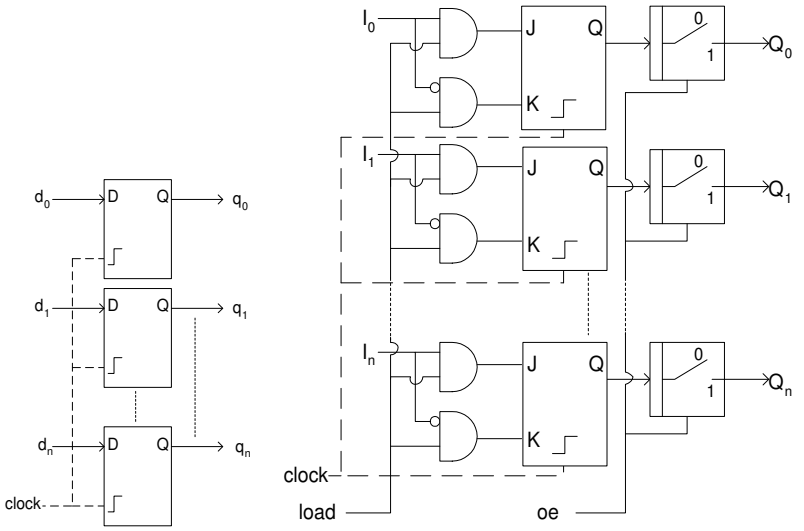


Figura 3.6: Due registri paralleli ad n bit

meno significativo del registro. Sono quindi necessari n cicli di clock per memorizzare una nuova parola nel registro. Nella figura 3.7 é illustrato invece un right-shift register a 4 bit, realizzato con flip-flop D edge-triggered.

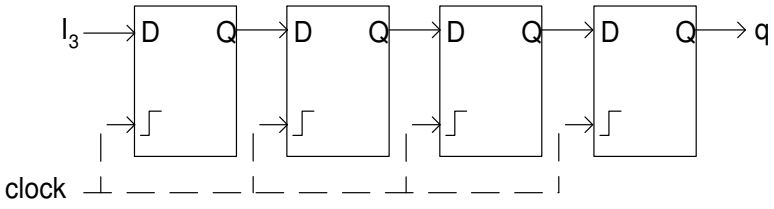
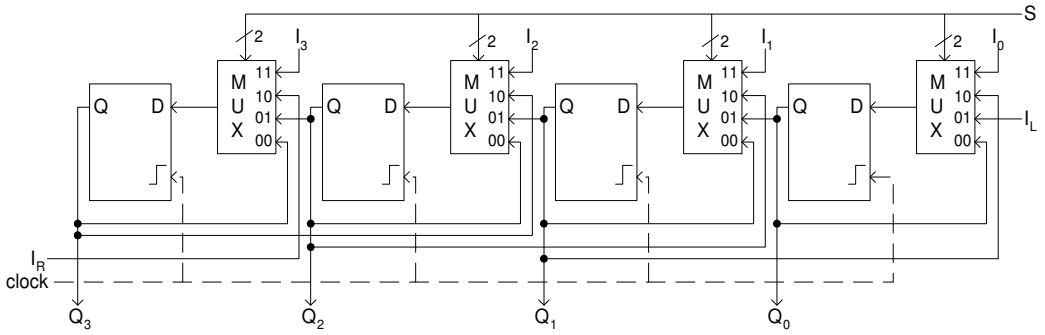


Figura 3.7: Right-shift register a 4 bit

In questa breve trattazione abbiamo supposto che l'uscita fosse sempre parallela, ma in realtá esistono anche registri con uscita seriale, e registri universali, che, in base al valore di alcune variabili di comando, possono avere uscita e ingressi seriale o parallela. Nella figura 3.8 nella pagina seguente é illustrato un registro universale a 4 bit. In dipendenza del valore della coppia di variabili indicate in figura con S, il registro può effettuare sia il caricamento parallelo, a partire dagli ingressi $I_3 \cdots I_0$, che il caricamento seriale (usando come ingressi

I_L per il left-shift, e I_R per il right shift).



(a)

S_1	S_0	**Operazione**
0	0	Nessuna operazione
0	1	Left-shift
1	0	Right-shift
1	1	Caricamento parallelo

(b)

Figura 3.8: Registro universale a 4 bit

3.6 Esempi di reti sincronizzate

Mostreremo adesso un modello strutturale per reti sequenziali sincronizzate e una implementazione che segue tale modello.

Nella figura 3.6 nella pagina successiva é visualizzato il *modello di Moore per reti sincrone*. Come si può notare é molto simile al modello visto per le reti asincrone, con la differenza che i ritardi sono sostituiti da un registro denominato *registro di stato*[1]. Questo comporta che lo stato interno della rete possa variare solo in prossimità del fronte in salita del clock. Anche in questo caso avremo una legge F e una legge G per calcolare lo stato interno e le uscite.

É importante notare come nella realizzazione della rete combinatoria per F non sia necessario porre attenzione né nel codificare stati adiacenti con parole che differiscano al piú per un bit, né nell'evitare la presenza di alee dinamiche. I transitori in uscita alla rete combinatoria, se eventualmente ci fossero, verrebbero infatti "filtrati" dalla presenza del registro di stato.

[1]Se ci sono k possibili stati interni, sarà necessario un registro a $w = \lceil \log_2 k \rceil$ bit.

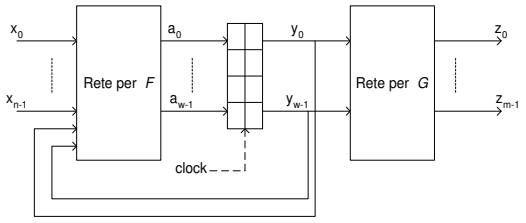


Figura 3.9: Il modello di Moore per le reti sincronizzate

Una variante del modello sopra presentato consiste nel sostituire il registro di stato con altri marcatori sincronizzati (es. flip-flop JK edge-triggered), in analogia a quanto visto con le reti asincrone allorchè si usavano latch SR.

3.6.1 Il flip-flop JK come rete sincrona

Per esercizio, realizziamo un flip-flop JK sincronizzato usando il modello per reti sincrone di Moore visto precedentemente. Nella figura 3.10(a) nella pagina seguente é riportata la tabella di stato del flip-flop JK. Codificando lo stato S_0 con 0 e lo stato S_1 con 1, otteniamo che la rete per G é un identità, ed é necessario un registro di stato ad un solo bit.

La figura 3.10(c) nella pagina successiva mostra il risultato finale del processo di sintesi.

3.6.2 Riconoscitore di sequenza

Implementiamo un riconoscitore della sequenza 01,10,11 come rete di Moore, con flip-flop JK in luogo del registro di stato. Avremo il grafo di flusso di figura 3.11(a) a pagina 34, che corrisponde alla tabella di stato in fig. 3.11(b).

Codificando lo stato S_i con il numero i espresso in binario, e ricordandoci la tabella di eccitazione del flip-flop JK, otteniamo la rete di figura 3.11(d).

3.7 Le memorie RAM

Le memorie RAM sono memorie *volatili* (mantengono l'informazione solo se alimentate) di lettura e scrittura.

Una memoria RAM può essere schematizzata come un insieme di *locazioni*, dette anche *celle*, ognuna delle quali é in grado di mantenere un'informazione (una parola).

Derivano il loro nome (*Random Access Memory*) dal fatto che, a differenza delle memorie ad accesso sequenziale (es. nastri magnetici), per leggere il contenuto di una locazione non é necessario leggere tutte le locazioni precedenti.

Una memoria $2^m \times n$ ha 2^m locazioni, ciascuna individuabile da un *indirizzo* ad m bit, che contengono una parola da n bit.

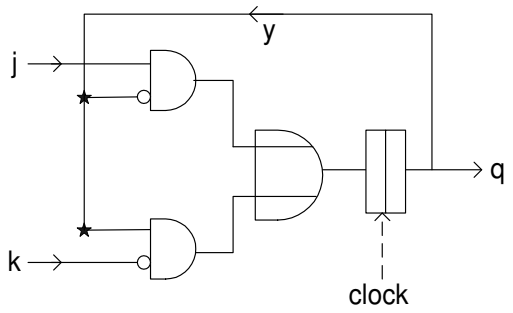
	jk				q
	00	01	11	10	
S_0	S_0	S_0	S_1	S_1	0
S_1	S_1	S_0	S_0	S_1	1

(a) Tabella di flusso

	jk			
y	00	01	11	10
0	0	0	1	1
1	1	0	0	1

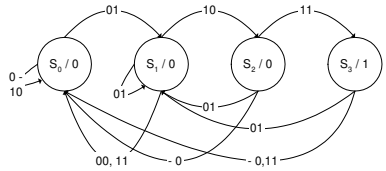
$$y = (\overline{y}j) + (y\overline{k})$$

(b) Tabella delle transizioni



(c) Rete finale

Figura 3.10: Il flip-flop JK implementato come rete di Moore



(a) Grafo di flusso

	x_1x_0 00	01	11	10	q
S_0	S_0	S_1	S_0	S_0	0
S_1	S_0	S_1	S_0	S_2	0
S_2	S_0	S_1	S_3	S_0	0
S_3	S_0	S_1	S_0	S_0	1

(b) Tabella di stato

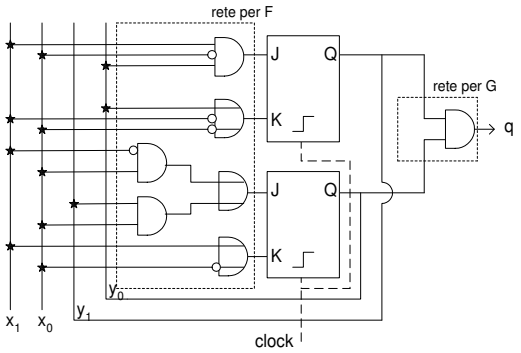
y_1y_0 \ x_1x_0	00	01	11	10
00	0–	1–	0–	0–
01	–1	–0	–1	–1
11	–1	–0	–1	–1
10	0–	1–	1–	0–

j_0k_0

y_1y_0 \ x_1x_0	00	01	11	10
00	0–	0–	0–	0–
01	0–	0–	0–	1–
11	–1	–1	–1	–1
10	–1	–1	–0	–1

j_1k_1

(c) Tabelle per la generazione degli ingressi dei flip-flop jk



(d) Rete finale

Figura 3.11: Riconoscitore della sequenza 01, 10, 11

Con una memoria RAM si possono eseguire due operazioni (lettura e scrittura) mutuamente esclusive, ovvero quando si sta scrivendo non si puó contemporaneamente leggere, e viceversa. Nella figura 3.12 é mostrato un modulo di RAM $2^m \times n$:

Il piedino $/s$, quando attivo, indica che il modulo di memoria é selezionato e può effettuare un'operazione.

Gli ingressi $/oe$ e $/we$, quando attivi indicano che é in corso un'operazione, rispettivamente, di lettura o di scrittura.

Gli ingressi $a_{m-1} - a_0$ sono interpretati, durante un operazione di lettura (scrittura), come l'indirizzo della cella da cui leggere (su cui scrivere).

Le linee $d_{n-1} - d_0$ sono bidirezionali. Durante un'operazione di scrittura fungono da ingressi e rappresentano la nuova parola da memorizzare nella cella indirizzata. Durante un'operazione di lettura sono le uscite da cui prelevare il valore delle parola da leggere. Se non é in corso un'operazione di lettura sono poste in alta impedenza.

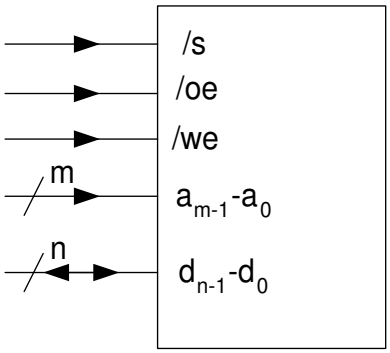


Figura 3.12: Simbolo e piedinatura di una RAM $2^m \times n$

É possibile realizzare un modulo di RAM $2^m \times n$ attraverso una matrice di $2^m \times n$ D latch e della logica di controllo (semplici reti combinatorie, mux e demux) che permette di selezionare un'intera riga della matrice, e fornire i corretti segnali di sincronizzazione ai latch.

3.7.1 Tecniche di espansione di parola e indirizzi

Combinando opportunamente due moduli di RAM identici é possibile ottenere un modulo di RAM che ha lo stesso numero di righe (celle), ma una dimensione doppia della parola. Nella figura 3.13 nella pagina seguente sono mostrati i collegamenti necessari (in sostanza basta far sì che i due moduli originari funzionino in parallelo).

A partire da due moduli di RAM identici si puó anche ottenere un modulo con la stessa dimensione della parola, ma con un numero doppio di celle

(espansione degli indirizzi). Basta collegarli come in figura 3.7.1, dove la rete combinatoria RC é tale da attivare al più un modulo dei due, a seconda del valore del bit piú significativo degli indirizzi.

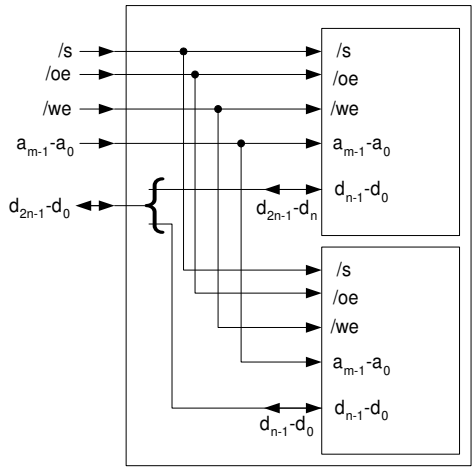
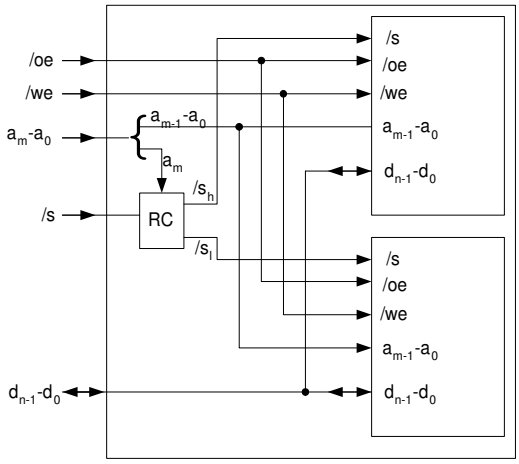


Figura 3.13: Modulo di RAM $2^m \times 2n$ a partire da due moduli $2^m \times n$



(a)

$/s$	a_m	$/s_l$	$/s_h$
1	-	1	1
0	0	0	1
0	1	1	0

(b) Tabella di veritá della rete combinatoria RC

Figura 3.14: Modulo di RAM $2^{m+1} \times n$ a partire da due moduli $2^m \times n$