

Capitolo 1

Sistemi Logici

1.1 Sistemi di numerazione

1.1.1 Il sistema binario

Nella vita quotidiana siamo abituati ad usare il sistema decimale. In tale sistema la "base" é 10, ciò che vuol dire l'uso di 10 simboli diversi per rappresentare i numeri da 0 a 9. I calcolatori elettronici, e con essi tutti i moderni sistemi di elettronica digitale, sono basati sull'uso di "celle di memoria" che possono memorizzare solo due simboli diversi, convenzionalmente chiamati 0 ed 1. Per vedere come, a partire da questi due soli simboli, sia possibile "costruire" un intero generico, torniamo un attimo al sistema decimale e vediamo di analizzare le operazioni mentali che istintivamente effettuiamo nello scrivere un numero intero decimale. Vediamo ciò con un esempio. Prendiamo il numero 23. Esso implica:

a scrivere la cifra 2

b moltiplicarla per 10

c aggiungere al risultato un 3

Se aggiungiamo al 23, a destra, un'ulteriore cifra, ad esempio un 7, ciò che facciamo é:

d moltiplicare il risultato delle operazioni a), b), c) precedenti per 10

e aggiungere al risultato un 7

Nel sistema adoperato nei calcolatori, detto sistema binario, il meccanismo adoperato per costruire un numero é il medesimo appena descritto, con la differenza che ora i simboli sono solo 0 ed 1 e che l'operazione "moltiplicazione per 10" diviene "moltiplicazione per 2". Vediamo un esempio. Prendiamo il numero 101. Effettuiamo le operazioni a), b), c), d), e) descritte, con la modifica menzionata:

a' scriviamo un 1

b' moltiplichiamo questo per 2

c' aggiungiamo al risultato la seconda cifra (0)

d' moltiplichiamo il risultato per 2

e' aggiungiamo al risultato la terza cifra

Quindi il numero binario scritto é, in decimale, un 5.

E' immediato vedere che, aggiungere al numero precedente uno 0 (a destra) equivale a moltiplicare per 2 il 5 ottenuto, ottenendo cosí un 10. Vediamo quindi che, come nel sistema decimale l'aggiunta di uno 0 a destra di un numero equivale a moltiplicare questo per 10, cosí nel sistema binario l'aggiunta di uno 0 a destra di un numero equivale a moltiplicare questo per 2. Proviamo a convertire in decimale il seguente numero binario:

1 0 1 1 0 1 0 1 1

Seguiremo la procedura mostrata nella tabella 1.1:

1×2	=	2
$2 + 0$	=	2
2×2	=	4
$4 + 1$	=	5
5×2	=	10
$10 + 1$	=	11
11×2	=	22
$22 + 0$	=	22
22×2	=	44
$44 + 1$	=	45
45×2	=	90
$90 + 0$	=	90
90×2	=	180
$180 + 1$	=	181
181×2	=	362
$362 + 1$	=	363

Tabella 1.1:

Seguendo il procedimento indicato riusciamo a convertire in decimale, mentalmente ed in modo rapido, un numero binario non eccessivamente grande.

Possiamo, in modo equivalente, esprimere il numero decimale (o binario) dato, facendo uso del metodo delle potenze di 10 (o di 2). Cosí ad esempio, il numero decimale 314159 é esprimibile come mostrato nella tabella 1.2:

Vale a dire che ogni cifra che compone il numero dato viene ad essere moltiplicata per una potenza di 10 pari alla posizione che essa occupa nel numero dato, contando le posizioni da destra a sinistra ed attribuendo alla cifra posta all'estrema destra la posizione 0. Il numero é la somma di tutti i termini in tal modo ottenuti.

In modo analogo, nel sistema binario si scrive il dato numero come somma delle singole cifre binarie, moltiplicate ciascuna per una potenza di 2, corrispondente alla posizione che essa occupa nel numero. Ad esempio, il numero binario 100111010 si esprime come mostrato in tabella 1.3:

Questo metodo, facile da implementare in un computer, é meno agevole per effettuare il calcolo mentalmente in modo rapido.

9×10^0	(9)	+	
5×10^1	(50)	+	
1×10^2	(100)	+	
4×10^3	(4000)	+	
1×10^4	(10000)	+	
3×10^5	(300000)	=	314159

Tabella 1.2:

0×2^0	(0)	+	
1×2^1	(2)	+	
0×2^2	(0)	+	
1×2^3	(8)	+	
1×2^4	(16)	+	
1×2^5	(32)	+	
0×2^6	(0)	+	
0×2^7	(0)	+	
1×2^8	(256)	=	314

Tabella 1.3:

1.1.2 Il sistema ottale

Pur essendo il sistema binario quello adoperato nei sistemi digitali, altri sistemi diversi da quello binario sono spesso adoperati nello sviluppo del software che utilizza i sistemi digitali. I piú comuni sono quello ottale (base 8) e quello esadecimale (base 16). Nel sistema ottale si dispone di 8 simboli (da 0 a 7) e si "costruiscono" i numeri maggiori di 7 seguendo i passi indicati ai punti a) e), con la modifica del 10 in 8. Vediamo un esempio:

$$72_8 = 58_{10}$$

Cioé, come mostrato nella tabella 1.4:

7×8	=	56
$56 + 2$	=	58

Tabella 1.4:

Ovviamente, in tale sistema il numero 8 si scriverá "10", cioè 1×8 .

1.1.3 Il sistema esadecimale

Nel sistema esadecimale si dispone di 16 simboli (i numeri decimali da 0 a 9, e delle lettere A, B, C, D, E, F). Abbiamo la situazione mostrata nella tabella 1.5:

A	→	10 ₁₀
B	→	11 ₁₀
C	→	12 ₁₀
D	→	13 ₁₀
E	→	14 ₁₀
F	→	15 ₁₀

Tabella 1.5:

I numeri maggiori di 15 vengono "costruiti" seguendo la ricetta delineata ai punti a) e), con la sostituzione del 10 con il 16. Ad esempio il numero esadecimale "1A" in decimale é quello mostrato nella tabella 1.6:

1	×	16	=	16
16	+	A	=	26

Tabella 1.6:

Il numero esadecimale "FF" é uguale a $(15 \times 16) + 15 = 255$.

1.1.4 Conversioni tra sistemi diversi

É facile convertire in ottale un numero binario. Basta tener presente che un insieme di tre bit binari può rappresentare i numeri base del sistema ottale, cioè i numeri tra 0 e 7. Possiamo allora raggruppare tre a tre i bit che costituiscono il numero binario dato (a partire dalla destra) per ottenere il numero in ottale. Così ad esempio:

$$\underbrace{[110]}_6 \underbrace{101}_5 \underbrace{001}_1 \underbrace{111}_7]_2 = [6517]_8$$

In modo analogo, é possibile convertire in esadecimale un generico numero binario. Questa volta si dovranno raggruppare quattro alla volta i bit che costituiscono il numero binario dato e sostituire a ciascun gruppo il corrispondente numero esadecimale. Ad esempio, il numero prima convertito in ottale, diventa:

$$\underbrace{[1101]}_D \underbrace{0100}_4 \underbrace{1111}_F]_2 = [D4F]_{16}$$

E' facile vedere poi come si possa agevolmente convertire un numero ottale o esadecimale in un numero binario. A tale scopo é sufficiente convertire in binario ciascuna delle cifre che costituiscono il numero ottale o esadecimale.

Ci si può chiedere ora come si possa convertire viceversa un numero decimale dato in una base 2. La procedura é la seguente:

- (a) si divide per 2 il numero decimale assegnato, ottenendo un quoziente ed un resto. Notiamo che il resto sarà 0 (se il numero era pari) o 1 (se era dispari)
- (b) si divide per 2 il quoziente ottenuto. Si ottiene un nuovo quoziente ed un nuovo resto

- (c) si continua l'operazione, fino a che il quoziente non sia nullo
- (d) la serie dei resti ottenuti, letta in ordine inverso, fornirà il numero binario richiesto.

Vediamo ciò con un esempio

$$[374]_{10} = [?]_2$$

Le successive operazioni sono quelle mostrate nella tabella 1.7.

Divisione	Quoziente	Resto
374/2	187	0
187/2	93	1
93/2	46	1
46/2	23	0
23/2	11	1
11/2	5	1
5/2	2	1
2/2	1	0
1/2	0	1

Tabella 1.7:

Il numero binario é quindi:

$$1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0$$

1.1.5 Numeri binari frazionari

Cosí come nel sistema decimale le cifre dopo la virgola decimale (o il punto decimale) vengono ad essere moltiplicate per potenza negative e crescenti di 10, cosí nel sistema binario le cifre dopo la virgola (binaria) vengono ad essere moltiplicate per potenze negative e crescenti di 2. Ad esempio:

$$[101.1101]_2 = [5. \left(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} \right)]_{10}$$

Se vogliamo convertire un numero decimale frazionario in un numero binario, potremo procedere secondo le linee del seguente esempio.

Supponiamo di dover convertire il numero decimale 0.8125. Moltiplichiamo il numero per 2 e chiediamoci se il risultato é ≥ 1 .

$$0.8125 \times 2 = 1.625 \geq 1$$

Poiché la disuguaglianza é soddisfatta, scriviamo un 1 e sottraiamo un 1 dal risultato. Se non lo é scriveremo 0 e non effettueremo la sottrazione:

$$1.625 - 1 = 0.625 \Rightarrow \boxed{1}$$

Moltiplichiamo ora il numero ottenuto (0.625) per 2 e verifichiamo nuovamente se il risultato é ≥ 1 . Se lo é scriviamo un altro 1 e sottraiamo 1 dal numero dato.

$$0.625 \times 2 = 1.25 \geq 1 \Rightarrow \boxed{1}$$

$$1.25 - 1 = 0.25$$

Moltiplichiamo ancora 0.25 per 2 e verifichiamo:

$$0.25 \times 2 = 0.5 < 1 \Rightarrow \boxed{0}$$

Poiché questa volta la disuguaglianza non é soddisfatta, abbiamo scritto uno zero anziché un 1.

Moltiplichiamo nuovamente per 2:

$$0.5 \times 2 = 1 \geq 1 \Rightarrow \boxed{1}$$

$$1 - 1 = 0 \quad \textit{fine del processo}$$

La sequenza di 1 e 0 scritta, letta dall'alto verso il basso, rappresenta il numero frazionario desiderato. Nel nostro caso esso é:

$$0.1101$$

Verifichiamo:

$$0.1101 = \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} = 0.8125$$

1.1.6 Codifica di numeri binari in un computer

In una macchina a 32 bit, il piú grande numero intero che é possibile rappresentare, tenendo conto del fatto che un bit é usato per il segno, é:

$$N = 2147483648$$

Si tratta di un numero grande, ma non abbastanza per tutte le situazioni. Per rappresentare numeri molto piú grandi (o anche numero molto minori di 1) si ricorre alla rappresentazione che fa uso della caratteristica e della parte frazionaria. Ciò é in fondo analogo a quanto facciamo nel sistema decimale per rappresentare numeri molto grandi o molto piccoli, ricorrendo alla notazione esponenziale. Ad esempio, per il numero di Avogadro scriviamo:

$$N_A = 6.022045 \cdot 10^{23}$$

e per la costante di Boltzmann:

$$K = 1.380662 \cdot 10^{-23}$$

Cosí, nei computer, un numero puó essere rappresentato dal prodotto di un numero binario per una potenza di 16, dove l'esponente é specificato da un secondo numero binario. Ad esempio, nel vecchio sistema IBM/370, un "parola" di 32 bit era divisa in due gruppi di bits, come mostrato nella tabella 1.8.

Il bit 0, all'estrema sinistra, é adoperato per il segno. I successivi 7 bit sono adoperati per la caratteristica. I rimanenti 24 costituiscono la parte frazionaria.

S	caratteristica (7 bits)	parte frazionaria (24 bits)
0	1	7
		8
		31

Tabella 1.8:

Il punto decimale va immaginato alla sinistra del bit piú significativo della parte frazionaria.

La caratteristica é ottenuta in realtà aggiungendo 64 all'esponente vero e proprio. Ciò vien fatto allo scopo di riuscire a rappresentare anche esponenti molto minori di zero.

Notiamo che i 7 bits della caratteristica possono rappresentare i numeri decimali che vanno da 0 a 127. Ad esempio, la rappresentazione di un numero che in notazione decimale fosse:

$$N_D = F_D 10^{C_D}$$

dove F_D é la parte frazionaria (.6022045 nel caso del numero di Avogadro) e C_D la caratteristica nello stesso esempio (cioé 24), si otterrebbe nel modo seguente:

- a) si converte in binario la parte frazionaria, conservando solo 24 bits
- b) si converte 10^{C_D} in 16^{C_H} , cioè in una potenza di 16
- c) al valore di C_H ottenuto si somma 64 (decimale). Questa sarà la caratteristica del numero in esame.

La rappresentazione descritta é quella dei numeri detti "floating point" (ovvero, in italiano, "in virgola mobile") quando si adotti la "singola precisione". Per calcoli piú accurati é possibile usare "parole" costituite da gruppi di 2 parole singole (ovvero 64 bit) o anche di 4 parole singole (ovvero 128 bit).

Poiché un gruppo di 8 bits costituisce quello che é chiamato un "byte", una parola "singola precisione" é costituita da 4 bytes, una in "doppia precisione" da 8 bytes, ed una in "precisione estesa" da 16 bytes.

1.1.7 Il codice Gray

Sempre restando nell'ambito di una logica di tipo binario, occorre notare che la costruzione che abbiamo descritto della corrispondenza tra numeri binari e numeri decimali, suggerita dall'analogia con il sistema decimale, non é unica. Infatti si può altrettanto facilmente ipotizzare una generica corrispondenza tra numeri binari e numeri decimali ed adottarla come "sistema di numerazione binaria". Tale corrispondenza deve essere implementabile tramite una precisa "ricetta". Ciò é fatto ad esempio dal codice Grey. In tale codice, i numeri binari 0 ed 1 corrispondono allo 0 ed 1 decimale rispettivamente, come mostrato nella tabella 1.9.

Per "costruire" la configurazione binaria corrispondente al 2 ed al 3 decimali, effettuiamo una "riflessione" delle prime due righe nello specchio rappresentato dalla prima linea tratteggiata e contemporaneamente cambiamo in 1 gli zeri che vengono a comparire nella seconda colonna a partire dalla destra. Per "costruire" ora la configurazione binaria corrispondente al 4, 5, 6, 7 decimali, effettuiamo una nuova riflessione delle prime quattro righe nello specchio rappresentato dalla seconda linea tratteggiata, e contemporaneamente cambiamo in 1 gli zeri che compaiono nella terza colonna, a partire dalla destra, delle quattro nuove righe.

0000	→	0	
0001	→	1	
— — — —	— — — —	— — — — —	prima riflessione
0011	→	2	
0010	→	3	
— — — —	— — — —	— — — — —	seconda riflessione
0110	→	4	
0111	→	5	
0101	→	6	
0100	→	7	

Tabella 1.9:

Il processo può esser proseguito quanto si vuole. Notiamo che una caratteristica del sistema Grey é che nel passare da un numero al successivo, uno solo dei bit cambia. Così, nel passare dal 2 al 3 cambia solo il bit meno significativo, nel passare dal 3 al 4 cambia solo il terzo bit, etc. Ciò non accade nel sistema binario "convenzionale" dove, nel passaggio da un valore a quello adiacente possono cambiare un numero arbitrario di bits. Ciò é un indubbio vantaggio del codice Grey in applicazioni elettroniche, dove i tempi di commutazione degli elementi che memorizzano i diversi bit possono differire leggermente.

1.2 Porte logiche

L'informazione elementare alla base del funzionamento di un qualsiasi sistema digitale é il bit. I due valori che questo può acquistare, che convenzionalmente chiamiamo 0 ed 1, sono nella pratica associati a due possibili valori di tensione (ad esempio 0V e 5V) o di corrente. La convenzione che può essere adottata nello stabilire la corrispondenza tra stato logico (0 o 1) e livello di tensione (0V o 5V) stabilisce il tipo di logica. Se allo 0 logico facciamo corrispondere il più basso dei valori di tensione ed all'1 il più alto, diremo che stiamo lavorando in logica positiva. Viceversa, possiamo far corrispondere allo 0 logico il livello alto di tensione ed all'1 logico il livello basso; diremo che in tal caso stiamo lavorando in logica negativa.

Una variabile binaria può acquistare o il valore 0 o il valore 1. A partire da una o più variabili binarie possiamo poi, come vedremo tra un attimo, costruire delle funzioni di questa/e variabile/i. Se una variabile, o funzione di variabili, acquista il valore 1, diremo anche che essa é vera (True = T). Se acquista il valore logico 0, diremo che é falsa (False = F). Ovviamente, come una variabile, così anche una funzione logica di variabili logiche può assumere solo i valori 1=T o 0=F.

La più semplice delle funzioni (logiche) di variabile logica é il NOT, o invertitore. Se A é la variabile, l'applicazione della funzione NOT fornisce una nuova variabile che é il complemento (logico) di A. Cioé, se A = 0, il NOT di A vale 1; se A = 1, il NOT vale 0. Il simbolo per indicare la negazione (cioé il NOT) é:

$$NOT(A) = \overline{A}$$

Il simbolo del circuito elettronico che implementa tale operazione é quello mostrato in figura 1.1:

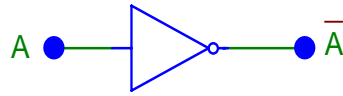


Figura 1.1:

Una funzione logica può esser caratterizzata scrivendo esplicitamente il valore assunto dalla funzione per ogni possibile valore del/degli ingressi. In genere si fa ciò sotto forma di una tabella, detta "tabella delle verità". Nel caso della funzione NOT questa é mostrata in figura 1.10:

A	$Y = \overline{A}$
0	1
1	0

Tabella 1.10:

La funzione OR a due ingressi é definita dalla tabella delle verità mostrata nella 1.11.

A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Tabella 1.11:

Cioé l'OR di A e B assume il valore 1 se almeno uno dei due ingressi A e B assume il valore 1. L'operatore che indica la funzione OR é il +:

Il simbolo del circuito logico che implementa la funzione OR é mostrato in figura 1.2:

La funzione AND a due ingressi é definita dalla tabella delle verità mostrata nella 1.12.

Vediamo dalla tabella che l'AND vale 1 se e solo se entrambi gli ingressi valgono 1. Il simbolo del circuito logico che implementa la funzione AND é mostrato in figura 1.3:

L'operatore che indica la funzione AND é quello della moltiplicazione: $A \cdot B$.

Porte NOR e NAND a più di due ingressi sono possibili. L'uscita di una funzione OR a più ingressi vale 1 se almeno uno degli ingressi vale 1. L'uscita di una funzione AND a più ingressi vale 1 se tutti gli ingressi valgono 1.

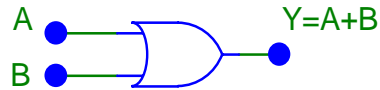


Figura 1.2:

A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Tabella 1.12:

E' ovviamente possibile combinare due o piú funzioni logiche per definire nuove funzioni. Cosí ad esempio, un AND seguito da un NOT realizza la funzione nota come NAND (= NOT·AND). La tabella delle veritá di un NAND é quella data nella 1.13

A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Tabella 1.13:

Il simbolo del circuito elettronico che implementa la porta NAND é mostrato in figura 1.4:

Un OR seguito da un NOT realizza la funzione nota come NOR (= NOT·OR), la cui tabella delle veritá é quella mostrata nella 1.14

A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Tabella 1.14:

Il simbolo del circuito elettronico che implementa la porta NOR é mostrato in figura 1.5:

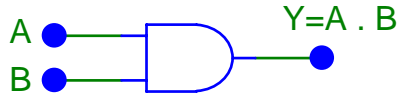


Figura 1.3:

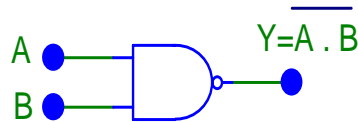


Figura 1.4:

La funzione "OR esclusivo", indicata con XOR, ha la medesima tabella delle verità della funzione OR, ad eccezione della riga con $A = B = 1$, in cui l'XOR vale 0. Possiamo pensare all'XOR come un "OR con esclusione dell'AND". Infatti la riga con $A = B = 1$ è quella per la quale è soddisfatto l'AND di A e B. Il simbolo del circuito che implementa la funzione XOR è mostrato in figura 1.6:

Il simbolo per l'XOR è il \oplus .

Sono disponibili in commercio integrati che effettuano le operazioni elementari descritte: AND, OR, NOT, NAND, XOR etc. I relativi chips sono distinti da sigle che variano a seconda della casa produttrice e che specificano caratteristiche diverse del relativo dispositivo. Esiste tuttavia una convenzione in base alla quale le ultime due cifre numeriche nella sigla indicano il tipo di funzione eseguita dall'integrato. Ad esempio, la sigla SN74LS00 indica un chip che esegue la funzione NAND a 2 ingressi. Le cifre numeriche che indicano la funzione NAND sono le ultime due (00). Più in particolare si tratta di quattro porte NAND realizzate sul medesimo chip (*QUAD* 2 input NAND gate). Analogamente, il codice 02 indica una porta NOR etc. Alcuni altri chip standard disponibili in commercio sono elencati nella tabella 1.15.

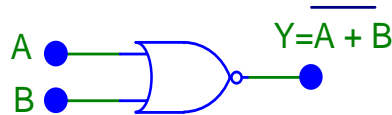


Figura 1.5:

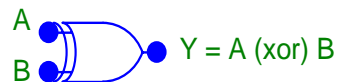


Figura 1.6:

Dispositivo	Funzione
00	Quad 2 input NAND
02	Quad 2 input NOR
04	Hex (6 in un chip) NOT
08	Quad 2 input AND
10	Triple 3 input NAND
20	Dual 4 input NAND
27	Triple 2 input NOR
30	8 input NAND
32	Quad 2 input OR
86	Quad exclusive OR
135	Quad exclusive OR/NOR
136	Quad exclusive OR

Tabella 1.15:

1.3 Algebra di Boole

1.3.1 Identit  Booleane

A partire da due o pi  variabili logiche A, B, C etc.,   possibile costruire espressioni logiche di varia complessit . Tali espressioni possono essere semplificate facendo uso delle regole dell'algebra di Boole. Tali regole sono nella pi  gran parte dei casi immediatamente ovvie; in altri possono essere dimostrate costruendo la relativa tabella delle verit  per evidenziare l'uguaglianza dei due membri dell'equazione.

Elenchiamo le pi  frequenti:

1. $A + B + C = (A + B) + C = A + (B + C)$ (proprietá associativa)
2. $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$ (proprietá associativa)
3. $A + B = B + A$ (proprietá commutativa)
4. $A \cdot B = B \cdot A$ (proprietá commutativa)
5. $A + A = A$ (identitá)
6. $A \cdot A = A$ (identitá)
7. $A + 1 = 1$
8. $A + 0 = A$
9. $A \cdot 1 = A$
10. $A \cdot 0 = 0$
11. $A \cdot (B + C) = A \cdot B + A \cdot C$ (proprietá distributiva)
12. $A + (A \cdot B) = A$ (ridondanza)
13. $A \cdot (A + B) = A$ (ridondanza)
14. $A + (B \cdot C) = (A + B) \cdot (A + C)$
15. $\overline{\overline{A}} = A$
16. $A \cdot \overline{A} = 0$
17. $A + \overline{A} = 1$
18. $A + (\overline{A} \cdot B) = A + B$
19. $A \cdot (\overline{A} + B) = A \cdot B$

Dimostriamo, facendo uso del metodo della tabella della verità, la (14). Il procedimento é mostrato nella tabella 1.16.

A	B	C	$(B \cdot C)$	$A + (B \cdot C)$	$(A + B)$	$(A + C)$	$(A + B) \cdot (A + C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Tabella 1.16:

\overline{A}	\overline{B}	A	B	$A \cdot B$	$\overline{A \cdot B} = \overline{A} + \overline{B}$
1	1	0	0	0	1
1	0	0	1	0	1
0	1	1	0	0	1
0	0	1	1	1	0

Tabella 1.17:

Notiamo che se $A = 1$ in logica positiva, allora $A = 0$ in logica negativa. Consideriamo ora la tabellina dell'AND (in logica positiva) (tabella 1.17).

Cambiare logica vuol dire complementare tutto, cioè scambiare 0 con 1 nella tabellina. Abbiamo indicato ciò nelle prime due colonne e nell'ultima. Notiamo che l'ultima colonna coincide con l'OR delle prime due. Possiamo allora dire che: "un AND in logica positiva diviene un OR in logica negativa". Questo risultato è anche conseguenza delle leggi di De Morgan, che esamineremo tra poco.

Esaminiamo ora la tabella delle verità (tabella 1.18) della funzione XOR ($Y = A \oplus B$).

A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 1.18:

La tabella ci permette di scrivere diverse espressioni della funzione XOR, facendo uso di funzioni OR, NOT ed AND. Una prima espressione si può ottenere esaminando le due righe in corrispondenza delle quali è $Y = 1$. La prima di queste corrisponde a A falso e B vero, cioè all'AND: $\overline{A} \cdot B$; la seconda corrisponde ad $A \cdot \overline{B}$. Poiché Y vale 1 sia nel primo caso che nel secondo, potremo scrivere:

$$1) A \oplus B = \overline{A} \cdot B + A \cdot \overline{B} \quad (1.1)$$

Una seconda espressione dell'XOR può essere ottenuta notando che l'XOR è vero quando è vero l'OR a condizione che sia falso l'AND. Cioè:

$$2) A \oplus B = (A + B) \cdot (\overline{A \cdot B}) \quad (1.2)$$

Una terza espressione si può ottenere notando che l'XOR è vero se è vero A o B : $(A + B)$ purché uno dei due sia falso ($\overline{A + B}$):

$$3) A \oplus B = (A + B) \cdot (\overline{A + B}) \quad (1.3)$$

Infine notiamo che il complemento dell'XOR è vero (cioè l'XOR è falso) nella prima e quarta riga della tabella delle verità.

Queste corrispondono a $\overline{A} \cdot \overline{B}$ (la prima) e a $A \cdot B$ l'ultima. Quindi:

$$\overline{A \oplus B} = \overline{A} \cdot \overline{B} + A \cdot B$$

cioé:

$$4) A \oplus B = \overline{\overline{A} \cdot \overline{B}} + A \cdot B \quad (1.4)$$

L'implementazione dei quattro modi indicati per realizzare la funzione XOR é indicata in figura 1.7, 1.8, 1.9 ed 1.10.

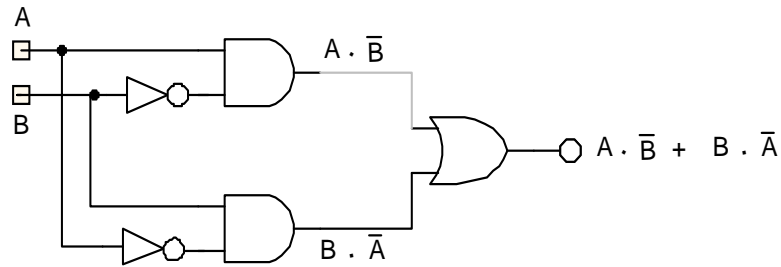


Figura 1.7:

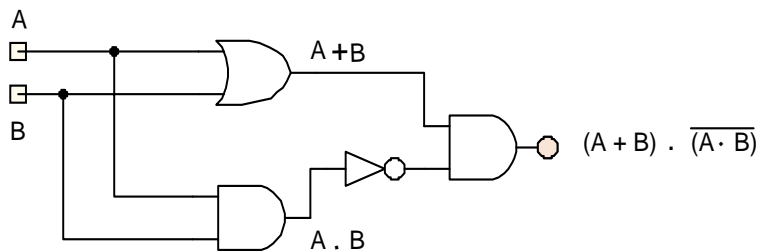


Figura 1.8:

1.4 Leggi di De Morgan

L'equivalenza delle quattro diverse implementazioni della funzione XOR, mostrata nella sezione precedente, può esser dimostrata in modo rigoroso facendo uso delle leggi di De Morgan. Esse sono:

$$\overline{A \cdot B \cdot C \cdots} = \overline{A} + \overline{B} + \overline{C} + \cdots$$

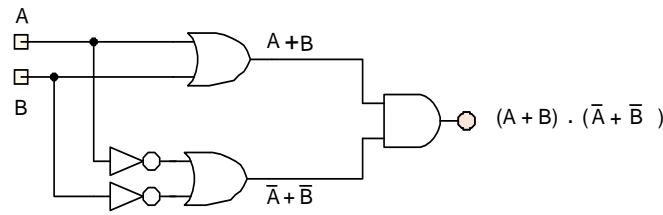


Figura 1.9:

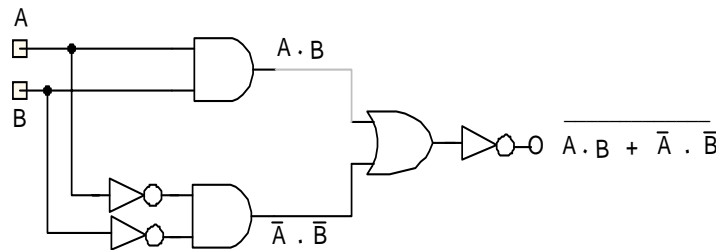


Figura 1.10:

$$\overline{A + B + C + \dots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \dots$$

La prima di queste ci dice che il complemento dell'AND di più variabili logiche é uguale all'OR dei complementi delle singole variabili. La seconda dice che il complemento dell'OR di più variabili logiche é uguale all'AND dei complementi delle singole variabili. Le leggi di De Morgan sono utilissime nel semplificare o trasformare espressioni logiche in più variabili.

Vediamo ora come, facendo uso di tali leggi, si possa facilmente dimostrare l'equivalenza delle quattro diverse espressioni che abbiamo fornito per la funzione XOR. Partiamo ad esempio dall'espressione 1.1:

$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

Complementandola due volte, otteniamo:

$$\dots \overline{\overline{\overline{A} \cdot B + A \cdot \overline{B}}} = \overline{\overline{\overline{A} \cdot B} \cdot \overline{A \cdot \overline{B}}} = \overline{(A + \overline{B}) \cdot (B + \overline{A})} = \overline{A \cdot B + \overline{A} \cdot \overline{B}}$$

ció troviamo la 1.4 data in precedenza.

Sviluppando quest'ultima troviamo poi:

$$\overline{A \cdot B + \overline{A} \cdot \overline{B}} = \overline{A \cdot B} \cdot \overline{\overline{A} \cdot \overline{B}} = (\overline{A} + \overline{B}) \cdot (A + B)$$

cioé la forma 1.3 dell'XOR.

Infine, complementando due volte il primo termine nell'ultimo membro dell'espressione appena scritta:

$$\overline{\overline{\overline{A} + \overline{B}}} \cdot (A + B) = (\overline{A \cdot B}) \cdot (A + B)$$

che é la forma 1.2 dell'XOR. E' cosí dimostrata l'equivalenza delle quattro forme.

Facendo uso dell'algebra di Boole e delle leggi di De Morgan é possibile semplificare proposizioni logiche complesse. E' inoltre possibile ottenere delle realizzazioni concrete di funzioni logiche, facendo uso di pochi componenti base. Ad esempio, é facile vedere come la funzione OR possa esser ottenuta facendo uso di sole porte NAND. Consideriamo infatti un OR con due ingressi A e B:

$$A + B$$

Negando due volte otteniamo:

$$\overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$$

cioé il simbolo circuitale mostrato in figura 1.11, che usa solo porte NAND.

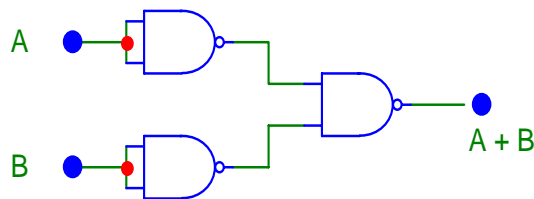


Figura 1.11:

1.5 Forme standard di funzioni logiche

Nel seguito, per semplificare la notazione, ometteremo il simbolo \cdot che indica il prodotto logico di due variabili.

Esistono due diverse forme standard per le funzioni logiche. Esse sono rispettivamente la *somma di prodotti* ed il *prodotto di somme*. Un esempio del primo tipo di funzione é:

$$XY + \overline{W}X + \overline{W}YZ + XYZ$$

dove compaiono le quattro variabili X,Y,Z,W.

Facciamo vedere ora come questa possa essere riscritta nella forma di un *prodotto di somme*. L'espressione può essere riscritta nella forma:

$$\begin{aligned}
XYX + \overline{W}X + \overline{W}YZ + XYYZ &= \\
(\overline{W} + XY)X + (\overline{W} + XY)YZ &= \\
(\overline{W} + XY)(X + YZ) &
\end{aligned}$$

che é un prodotto di somme.

Si dice che una somma di prodotti é in forma estesa se ciascun termine della somma contiene tutte le variabili. Tale forma emerge in modo naturale se la funzione logica é ottenuta da una tabella delle veritá. Vediamo ciò con un esempio (tabella 1.19).

X	Y	Z	L
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tabella 1.19:

cioé:

$$L = \overline{X}Y\overline{Z} + \overline{X}YZ + X\overline{Y}\overline{Z} + XY\overline{Z} + XYZ$$

Ciascuno dei termini della somma va sotto il nome di "minterm". Notiamo che l'espressione scritta può esser semplificata:

$$\begin{aligned}
L &= \overline{X}Y(\overline{Z} + Z) + X\overline{Y}\overline{Z} + XY(Z + \overline{Z}) = \overline{X}Y + X\overline{Y}\overline{Z} + XY = \\
&= Y(X + \overline{X}) + X\overline{Y}\overline{Z} = Y + \overline{Y}X\overline{Z} = Y + X\overline{Z}
\end{aligned}$$

Un esempio di prodotto di somme é:

$$L = (\overline{W} + XY)(X + YZ)$$

Ciascuno dei termini del prodotto é un "maxterm". La funzione scritta può facilmente esser trasformata in una nuova espressione dove ciascun termine del prodotto contiene solo due delle variabili. Facendo uso della relazione:

$$A + (BC) = (A + B) \cdot (A + C)$$

troviamo:

$$L = (\overline{W} + X)(\overline{W} + Y)(X + Y)(X + Z)$$

Diremo che il prodotto di somma é in *forma estesa* se ciascuna somma contiene tutte le variabili. Tale forma può facilmente essere ottenuta nel caso in cui la funzione L sia definita da una tabella della verità. Ad esempio, facendo uso della medesima tabella usata sopra, calcoliamo \overline{L} :

$$\overline{L} = \overline{XYZ} + \overline{XYZ} + \overline{XYZ}$$

da cui:

$$\begin{aligned} \overline{\overline{L}} &= L = \overline{\overline{XYZ} + \overline{XYZ} + \overline{XYZ}} = \overline{\overline{XYZ}} \cdot \overline{\overline{XYZ}} \cdot \overline{\overline{XYZ}} = \\ &= (X + Y + Z)(X + Y + \overline{Z})(\overline{X}\overline{Y}\overline{Z}) \end{aligned}$$

che é un prodotto di somme in forma estesa.

1.6 Operazioni aritmetiche

Vediamo ora come, facendo uso di circuiti elementari che implementano funzioni logiche, sia possibile effettuare la somma o la differenza di numeri binari. Immaginiamo di voler effettuare la somma di due bit a e b . Indichiamo con Σ la somma e con R il riporto. É facile verificare che essi sono dati dalla tabella delle verità mostrata nella 1.20.

a	b	Σ	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 1.20:

Vediamo facilmente che Σ coincide con $a \oplus b$, mentre R coincide con $a \cdot b$.

Immaginiamo di avere due numeri, di n bit ciascuno e contiamo i bit di ciascuno dei numeri a partire dal bit meno significativo (il piú a destra di tutti) da 0 ad $(n-1)$. Cioé:

$$\underline{a} = a_{n-1}a_{n-2} \cdots a_3a_2a_1a_0$$

$$\underline{b} = b_{n-1}b_{n-2} \cdots b_3b_2b_1b_0$$

La somma del bit i^{emo} di \underline{a} e del corrispondente di \underline{b} potrà essere ottenuto tramite un circuito che calcoli l'OR esclusivo e l'AND di a_i e b_i . Un tale circuito va sotto il nome di "semisommatore" o "Half Adder" (HA). In termini circuitali avremo quanto mostrato in figura 1.12.

Per effettuare la somma dei numeri a e b occorrerà un circuito che sommi alla somma dei due bit a_i e b_i , il riporto R_{i-1} dalla somma dei due bit precedenti (meno significativi). Tale circuito può in linea di principio essere ottenuto da due H.A., nel modo indicato in figura 1.13.

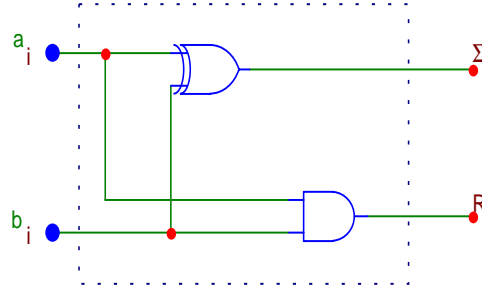


Figura 1.12:

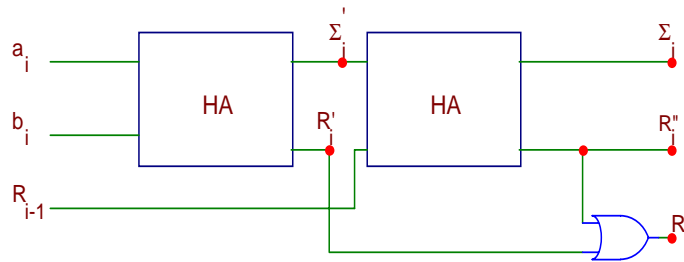


Figura 1.13:

a_i	b_i	R_{i-1}	Σ_i	R_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Tabella 1.21:

E' facile verificare che il riporto finale R_i é l'OR del riporto R'_i dal primo semisommatore e di quello R''_i dal secondo.

Un modo piú pratico di effettuare la somma dei due bit e del riporto della somma dei due bit precedenti é ottenibile analizzando la tabella delle verità per i termini $a_i, b_i, R_{i-1}, \Sigma_i, R_i$ mostrata nella tabella 1.21.

Dalla tabella delle verità si ottiene:

$$\Sigma_i = \bar{a}_i b_i \bar{R}_{i-1} + a_i \bar{b}_i \bar{R}_{i-1} + \bar{a}_i \bar{b}_i R_{i-1} + a_i b_i R_{i-1}$$

$$R_i = a_i b_i \bar{R}_{i-1} + \bar{a}_i b_i R_{i-1} + a_i \bar{b}_i R_{i-1} + a_i b_i R_{i-1}$$

Notiamo che R_i può essere semplificato in:

$$R_i = b_i R_{i-1} + a_i b_i + a_i R_{i-1} = a_i b_i + R_{i-1} \cdot (a_i + b_i)$$

La somma Σ_i é esprimibile come:

$$R_{i-1} \cdot (a_i b_i + \bar{a}_i \bar{b}_i) + \bar{R}_{i-1} \cdot (a_i \bar{b}_i + \bar{a}_i b_i)$$

Il termine nella prima parentesi é: $\overline{a_i \oplus b_i}$, quello nella seconda é: $a_i \oplus b_i$. Ne segue:

$$\Sigma_i = R_{i-1} \cdot \overline{a_i \oplus b_i} + \bar{R}_{i-1} \cdot (a_i \oplus b_i)$$

É evidente allora che questa altro non é che:

$$\Sigma_i = R_{i-1} \oplus (a_i \oplus b_i)$$

La somma ed il riporto sono allora entrambi ottenibili dal circuito di figura 1.14.

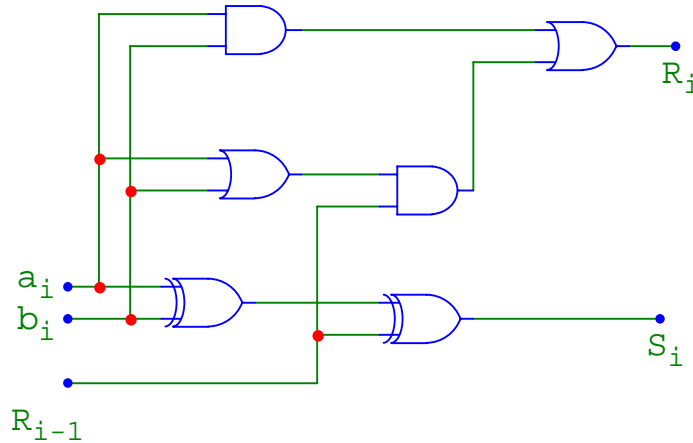


Figura 1.14:

Vediamo in tal modo che Σ_i ed R_i possono facilmente essere realizzati utilizzando porte logiche elementari.

1.7 Sottrazione binaria

Ammettiamo di voler sottrarre un numero binario A da un numero binario B, con $B > A$. Possiamo illustrare il procedimento da seguire se fissiamo il numero di bit di cui A e B sono costituiti. Ammettiamo ad esempio di dover adoperare numeri di 4 bit. Notiamo intanto che la somma di A e del suo complemento é:

$$A + \bar{A} = 1111$$

ció che di solito si esprime dicendo che \bar{A} é il *complemento ad 1* di A.

Nel seguito di questa sezione, per evitare di confondere il simbolo di somma con quello (+) adoperato per la funzione OR, adopereremo la lettera greca Σ per indicare la somma e la lettera Δ per indicare la differenza.

Notiamo che la relazione:

$$A\Sigma\bar{A} = 1111 \quad (1.5)$$

implica:

$$A\Sigma(\bar{A}\Sigma 1) = 10000 \quad (1.6)$$

Si dice anche che $(\bar{A}\Sigma 1)$ é il *complemento a 2* di A. Dalla 1.6 segue che:

$$A = 10000\Delta\bar{A}\Delta 1$$

e quindi:

$$B\Delta A = (B\Sigma\bar{A}\Sigma 1) \Delta 10000 \quad (1.7)$$

Dalla 1.7 segue che per sottrarre A da B occorre:

- a** complementare A, ottenendo \bar{A}
- b** sommare B ad \bar{A}
- c** sommare 1 al risultato
- d** lasciar cadere il quinto bit (il piú significativo)

Vediamo quindi che, con il metodo suggerito, riusciamo a realizzare l'operazione differenza effettuando operazioni che abbiamo già appreso, cioè il complemento (NOT) di ogni bit di un numero e la somma Σ . E' facile verificare che, con il procedimento indicato, se $B > A$, il quinto bit c'è sempre (cioé il numero ottenuto effettuando la somma: $B\Sigma\bar{A}\Sigma 1$ ha il quinto bit sempre uguale ad 1).

Possiamo ora verificare che, nel caso in cui il quinto bit sia 0, B é minore di A. Verificheremo anche che in tal caso la differenza (negativa) ha un valore assoluto pari al complemento di $(B\Sigma\bar{A})$.

Riscriviamo infatti la 1.7 nella forma:

$$B\Delta A = B\Sigma\bar{A}\Delta 1111 = \Delta [1111\Delta (B\Sigma\bar{A})] = \Delta [\overline{B\Sigma\bar{A}}]$$

Poiché, se n é un numero binario di quattro bit, $1111-n$ é il complemento di n.

Vediamo un esempio, cominciando dal caso $B > A$:

$$B = 12 = 1100 ; A = 9 = 1001$$

$$B\Delta A = B\Sigma\bar{A}\Sigma 1$$

dove: $\bar{A} = 0110$.

Esaminiamo la tabella 1.22.

Lasciamo ora cadere il bit piú significativo (il quinto) ed otteniamo:

$$B\Sigma\bar{A}\Sigma 1 - 10000 = 10011 - 10000 = 0011 = 3$$

B	1100
\overline{A}	0110
$B \Sigma \overline{A}$	10010
	1
$B \Sigma \overline{A} \Sigma 1$	10011

Tabella 1.22:

Come abbiamo già notato, se B è maggiore di A , la somma di B e di \overline{A} ha sempre il bit più significativo uguale ad 1. Ciò suggerisce di usare tale bit come addendo (cioè come l'1 da aggiungere nella terza delle operazioni sopra descritte (la (c))). Infatti, nel caso in cui B sia maggiore di A , il quinto bit varrà 1 e ciò è quel che occorre per realizzare la sequenza di operazioni sopra descritte. Se invece B è minore di A , la somma $B \Sigma \overline{A}$ avrà il bit più significativo uguale a 0 ed aggiungerlo quindi alla soma $B \Sigma \overline{A}$ non cambierà nulla.

Vediamolo nel caso concreto in cui:

$$B = 9 = 1001 \quad ; \quad A = 12 = 1100$$

Avremo la situazione mostrata nella tabella 1.23.

B	1001	
\overline{A}	0011	
$B \Sigma \overline{A}$	01100	il 5 bit è zero
	0	
$B \Sigma \overline{A} \Sigma 0$	01100	

Tabella 1.23:

Se complementiamo tale numero, troviamo:

$$\overline{B \Sigma \overline{A}} = 0011$$

da interpretare come numero negativo (cioè dobbiamo immaginare di aver aggiunto un segno meno).

Quanto detto ci suggerisce un metodo generale per effettuare la differenza di due numeri B ed A valido sempre (cioè per B maggiore, uguale o minore di A). Il metodo consiste nel:

- a** complementare A
- b** sommare B ad \overline{A}
- c** sommare al risultato il quinto bit (b5) della somma
- d** se $b5=1$, il risultato ottenuto al punto (c), dopo aver lasciato cadere il bit più significativo, rappresenta la differenza (positiva) $B \Delta A$

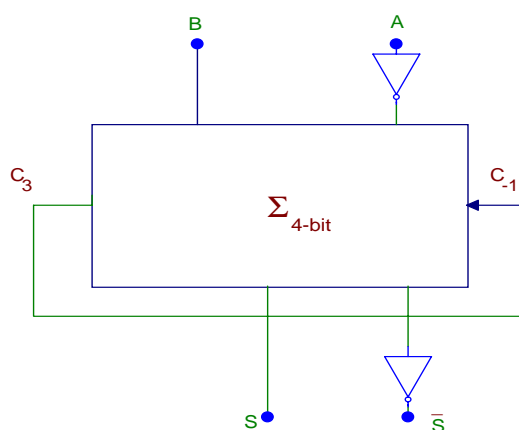


Figura 1.15:

e se invece $b_5=0$, il complemento del risultato ottenuto al punto (c) rappresenta il valore assoluto della differenza (negativa) $B \Delta A$.

Tutto ciò é riassunto schematicamente in figura 1.15.

Il circuito contenuto nella scatola riceve in ingresso \overline{A} , B ed il quinto bit della somma di B ed \overline{A} . Se tale bit vale 0, la differenza $B \Delta A$ é negativa ed il suo valore assoluto é \overline{S} . Se invece $c_3=1$, la differenza $B \Delta A$ (positiva) é data da S.

1.8 Rappresentazione dei numeri negativi

Come visto, per ottenere il complemento ad 1 di un numero binario, si complementano tutti i suoi bits. Una regola altrettanto semplice può essere adoperata per ottenere il complemento a 2: si esaminano i bit di cui il numero é costituito, a partire dalla destra. Si copiano i bit fino ad incontrare un bit uguale ad 1. A partire da quel punto si ricopia l'1 ed i complementi dei bit che seguono. Esempio:

$$C_2(100110100) = 01001100$$

Verifichiamo che $C_2(N) + N = 100000000$ (tabella 1.24).

10110100	+
01001100	=
100000000	

Tabella 1.24:

Cosí, se il primo bit sulla destra é un 1, il complemento a 2 può essere ottenuto inserendo alla sinistra di tale 1 i complementi di tutti i rimanenti bit.

E' ovvio da quanto detto che per rappresentare un numero negativo occorrerà aggiungere ai bit che rappresentano il numero, un ulteriore bit che ne rappresenti il segno. Le due convenzioni comunemente adoperate sono quella del "complemento a due" e quella del "complemento ad uno". Esaminiamo la prima di queste, ammettendo, per semplicitá, di voler rappresentare numeri il cui valore assoluto sia al

massimo il decimale 15 (1111 in binario). I numeri da 0 a 15 saranno i numeri binari di cinque bit, con il bit piú significativo (il quinto, che ora indicherá il segno) uguale a 0 e con i quattro meno significativi che vanno da 0000 a 1111.

Il numero negativo corrispondente ad un dato numero binario positivo si costruisce facendo il complemento a 2 del numero dato. Cosí:

$$+11 = 01011$$

$$-11 = 10101$$

Notiamo che, in tale sistema, la differenza di due numeri si ottiene prendendo la somma del primo e del complemento a 2 del secondo in tutti i casi! Cioé sia nel caso in cui A sia maggiore di B che in quello in cui A é minore di B. Vediamo ciò con un esempio, cominciando dal caso in cui sia A che B sono positivi. La somma é ricavata come mostrato nella tabella 1.25.

	binario	decimale
A =	00110	6 +
B =	00101	5 =
$A \Sigma B$ =	01011	11

Tabella 1.25:

Calcoliamo ora la differenza $A \Delta B$ (tabella 1.26).

	binario	decimale
A =	00110	6 +
$-B = C_2(B)$ =	11011	-5 =
$A \Sigma C_2(B)$ =	10001	1

Tabella 1.26:

Vediamo che ora il quinto bit é zero, mentre un 1 compare in sesta posizione. Ciò non crea alcun problema poiché nel caso specifico i nostri registri hanno solo 5 bit. La differenza é positiva ed uguale a 1. Calcoliamo ora $B \Delta A$ (tabella 1.27).

	binario	decimale
B =	00101	5 +
$-A = C_2(A)$ =	11010	-6 =
$B \Sigma C_2(A)$ =	11111	-1

Tabella 1.27:

Vediamo che il risultato é negativo (1 in quinta posizione) e che il modulo vale:

$$C_2(11111) = 00001 = 1$$

Un sistema analogo é quello della rappresentazione in "complemento ad 1".

Esamineremo piú avanti l'implementazione delle altre operazioni aritmetiche elementari.

1.9 Comparatori digitali

Ammettiamo di voler comparare due semplici bit A e B. Le possibili configurazioni sono quelle mostrate nella tabella 1.28:

A	B	Test	Y
0	0	$A = B$	$\overline{A} \cdot \overline{B}$
0	1	$A < B$	$\overline{A} \cdot B$
1	0	$A > B$	$A \cdot \overline{B}$
1	1	$A = B$	$A \cdot B$

Tabella 1.28:

La variabile Y nell'ultima colonna definisce in modo univoco lo stato di A e B. Notiamo che l'uguaglianza $A = B$, é espressa da:

$$Y = A \cdot B + \overline{A} \cdot \overline{B}$$

cioé, facendo uso delle leggi di De Morgan:

$$\begin{aligned} Y &= \overline{\overline{Y}} = \overline{\overline{A \cdot B + \overline{A} \cdot \overline{B}}} = \overline{\overline{A \cdot B} \cdot \overline{\overline{A} \cdot \overline{B}}} = \\ &= \overline{(\overline{A} + \overline{B}) \cdot (A + B)} = \overline{\overline{A} \cdot \overline{B} + B \cdot \overline{A}} \end{aligned}$$

In definitiva abbiamo la situazione mostrata in tabella 1.29.

$A > B$	$Y = A \cdot \overline{B} = 1$
$A < B$	$Y = B \cdot \overline{A} = 1$
$A = B$	$E \equiv Y = \overline{A \cdot \overline{B} + B \cdot \overline{A}} = 1$

Tabella 1.29:

Un circuito che fornisce in uscita la variabile Y é quello di figura 1.16.

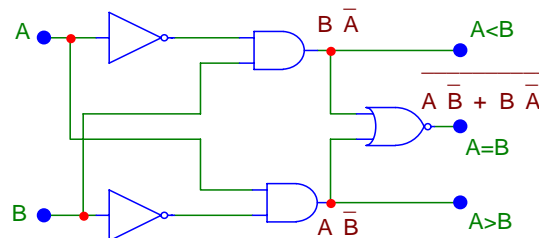


Figura 1.16:

Se vogliamo comparare dei numeri costituiti da piú bit, il circuito dovrà comparare i singoli bit ed un po' di logica aggiuntiva sarà necessaria per ottenere la risposta.

Ammettiamo ad esempio di voler comparare due numeri di 4 bit. L'uguaglianza dei due numeri richiederà ovviamente l'uguaglianza due a due dei bit corrispondenti. Così, indicando con A_3, A_2, A_1, A_0 i bit del primo numero e con B_3, B_2, B_1, B_0 quelli del secondo, l'uguaglianza richiederà:

$$A_3 = B_3; A_2 = B_2; A_1 = B_1; A_0 = B_0$$

La prima di queste uguaglianze è espressa da $E_3 = 1$, la seconda da $E_2 = 1$, etc.. Avremo quindi l'uguaglianza richiesta se è:

$$E_3 \cdot E_2 \cdot E_1 \cdot E_0 = 1$$

A sarà invece maggiore di B se:

a $A_3 > B_3$ oppure:

b $A_3 = B_3$ e $A_2 > B_2$ oppure:

c $A_3 = B_3$ e $A_2 = B_2$ e $A_1 > B_1$ oppure:

d $A_3 = B_3$ e $A_2 = B_2$ e $A_1 = B_1$ e $A_0 > B_0$

Cioè:

$$Y = A_3 \cdot \overline{B_3} + E_3 \cdot A_2 \cdot \overline{B_2} + E_3 \cdot E_2 \cdot A_1 \cdot \overline{B_1} + E_3 \cdot E_2 \cdot E_1 \cdot A_0 \cdot \overline{B_0}$$

La condizione $A < B$ si realizza scambiando nell'ultima espressione A con B.

1.10 Parità di un numero binario

Dato un numero binario, si definisce *parità* la parità della somma dei bit. Così ad esempio: 1101 ha parità dispari (numero dispari di 1); mentre 1001 ha parità pari.

L'uso della parità è utile per controllare se un numero binario è stato trasmesso correttamente da un elemento di circuito ad un altro elemento, o anche da un sito ad un sito distante. Infatti il rumore che è inevitabilmente presente sulle linee usate per la trasmissione può far sì che un bit cambi valore. Se ammettiamo che la probabilità che, per effetto del rumore, 2 bit in una medesima parola (e.g. di 8 bit) cambino valore, sia trascurabile, possiamo verificare l'eventuale presenza di errori aggiungendo alla parola un ulteriore bit che ci dica quale deve essere la parità del numero. Così, se i numeri hanno 8 bit ed il numero in questione ha parità dispari, potremo aggiungere un nono bit di valore 1 che renda la parità complessiva pari. Se il numero di 8 bit aveva parità pari, aggiungeremo un nono bit di valore 0, con che la parità complessiva rimane pari. In ricezione, adopereremo un circuito che controlli la parità e verifichi che sia sempre pari. Se essa risulterà dispari il sistema denuncerà l'errore al circuito da cui l'informazione proviene, il quale invierà nuovamente il dato. Ovviamente, è possibile adottare la convenzione di parità dispari, anziché pari come nell'esempio fatto.

Esaminiamo ora un semplice circuito che genera il bit di parità per una parola di 4 bit, ammettendo una parità pari.

Se avessimo solo due bit, la parità sarebbe quella mostrata nella tabella 1.30.

Vediamo che il "bit da aggiungere" altro non è che l'XOR di A e B.

Se il sistema ha quattro bit ($ABCD$), potremo definire la parità della coppia (AB), quella della coppia (CD) ed infine quella dei due bit di parità così ottenuti. Occorrerà cioè una cascata di XOR, come mostrato in figura 1.17.

A	B	parità del numero	bit da aggiungere
0	0	pari	0
0	1	dispari	1
1	0	dispari	1
1	1	pari	0

Tabella 1.30:

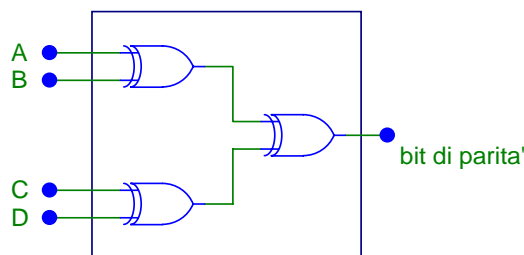


Figura 1.17:

1.11 Codificatori/Decodificatori

Un codificatore è un circuito che converte in un codice l'informazione corrispondente al verificarsi di uno tra n possibili eventi. Ad esempio, un codificatore è l'insieme dei circuiti presenti in un personal computer, che generano un numero binario (codice) quando si pigia un determinato tasto. Un decodificatore è viceversa un circuito che riceve in ingresso un codice (cioè un numero binario, codificato in modo opportuno) ed attiva una di n possibili linee (o eventi).

Esaminiamo come esempio di decodificatore un circuito per la conversione BCD-Decimale. BCD sta per "Binary Coded Decimal" ed è un modo per rappresentare (codificare) i numeri decimali. In tale sistema i numeri da 0 a 9 sono rappresentati da 4 bit (tabella 1.31).

Un decodificatore BCD-Decimale riceverà in ingresso i quattro bit ed attiverà una di dieci possibili linee di uscita, corrispondenti ai numeri decimali da 0 a 9. Notiamo per inciso che nel codice BCD un numero decimale di più cifre è rappresentato da tanti gruppi di quattro bit quante sono le cifre decimali che lo costituiscono. Ad esempio:

Se indichiamo con (a_3, a_2, a_1, a_0) i bit che costituiscono il codice BCD (con a_0 il meno significativo) e con $(\bar{a}_3, \bar{a}_2, \bar{a}_1, \bar{a}_0)$ i loro complementi, avremo:

$$\bar{a}_3 \cdot \bar{a}_2 \cdot \bar{a}_1 \cdot \bar{a}_0 \Rightarrow 0 \text{ decimale}$$

$$\bar{a}_3 \cdot \bar{a}_2 \cdot \bar{a}_1 \cdot a_0 \Rightarrow 1 \text{ decimale}$$

$$\bar{a}_3 \cdot \bar{a}_2 \cdot a_1 \cdot \bar{a}_0 \Rightarrow 2 \text{ decimale}$$

...

0	=	0000
1	=	0001
2	=	0010
3	=	0011
4	=	0100
5	=	0101
6	=	0110
7	=	0111
8	=	1000
9	=	1001

Tabella 1.31:

371 =	0011	0111	0001
	3	7	1

Tabella 1.32:

$$a_3 \cdot a_2 \cdot a_1 \cdot a_0 \Rightarrow 0 \text{ decimale}$$

Un circuito che realizza tale funzione logica é quello mostrato in figura 2.3:

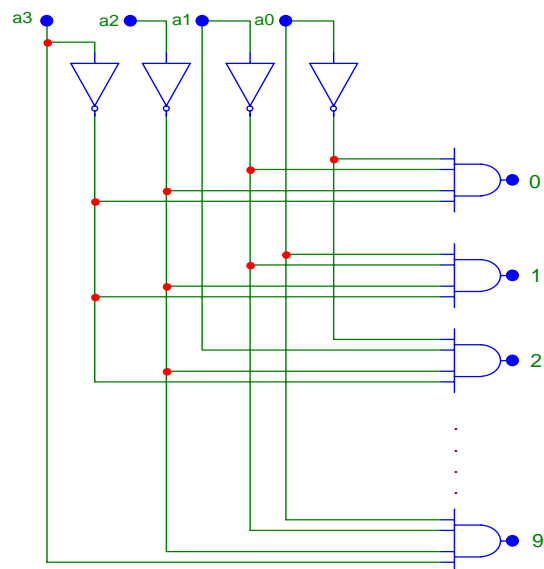


Figura 1.18:

Tale circuito é noto anche come *decodificatore da 4 a 10 linee*.

Esaminiamo ora la struttura di un codificatore. Ammettiamo di voler generare un carattere (codice) BCD in uscita pigiando uno di 10 tasti possibili. Siano i tasti $T_0 \cdots T_9$. La tabella delle verità che realizza la funzione di codifica é la 1.33.

T_9	T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Tabella 1.33:

Un tasto pigiato corrisponde alla variabile T uguale ad 1. Il tasto non pigiato corrisponde alla variabile T uguale a 0. Dalla tabella vediamo subito la corrispondenza tra le variabili Y e le T :

$$Y_0 = T_1 + T_3 + T_5 + T_7 + T_9$$

$$Y_1 = T_2 + T_3 + T_6 + T_7$$

$$Y_2 = T_4 + T_5 + T_6 + T_7$$

$$Y_3 = T_8 + T_9$$

Un circuito che realizza tale codifica può esser ottenuto da un integrato a matrice di diodi, come mostrato in figura 1.19.

Se uno degli interruttori $T_0 - T_9$ è chiuso, la corrispondente linea orizzontale viene a trovarsi a 5 V. Con ciò i diodi ad essa collegati avranno l'anodo a 5 V ed il catodo collegato alle resistenze R , a loro volta collegate a massa. In tali condizioni, la tensione sul catodo sarà $5-0.6=4.4$ V, cioè ancora al livello alto. L'uscita (O le uscite) collegate alla linea verticale cui è connesso il catodo del diodo in questione sarà quindi anch'essa al livello logico 1. Ciò implementa la funzione voluta.

Se in tale circuito due o più tasti vengono premuti contemporaneamente, il risultato può non essere quello desiderato. Un "codificatore con priorità" elimina tale problema. In un codificatore di questo tipo, se due o più tasti sono pigiati insieme, l'azione che ha luogo è quella che si avrebbe pigiando solo il tasto più alto nella sequenza $T_0 \cdots T_9$. Per ottenere tale funzione dobbiamo modificare la tabella delle verità come mostrato nella tabella 1.34.

Dove la x sta ad indicare che è irrilevante se quel tasto sia pigiato o no. Se ora si pigia il tasto T_2 ma per errore si pigia contemporaneamente il tasto T_3 , allora $T_3 \neq 0$ ed anche se $T_2 = 1$, diverranno "alti" i bit Y_1 ed Y_0 . Vediamo come le variabili $Y_3 \cdots Y_0$ possano essere espresse in funzione di $T_9 \cdots T_0$. Esaminiamo Y_0 :

$$Y_0 = T_1 \overline{T_2} \overline{T_3} \overline{T_4} \overline{T_5} \overline{T_6} \overline{T_7} \overline{T_8} \overline{T_9} + T_3 \overline{T_4} \overline{T_5} \overline{T_6} \overline{T_7} \overline{T_8} \overline{T_9} + \\ + T_5 \overline{T_6} \overline{T_7} \overline{T_8} \overline{T_9} + T_7 \overline{T_8} \overline{T_9} + T_9$$

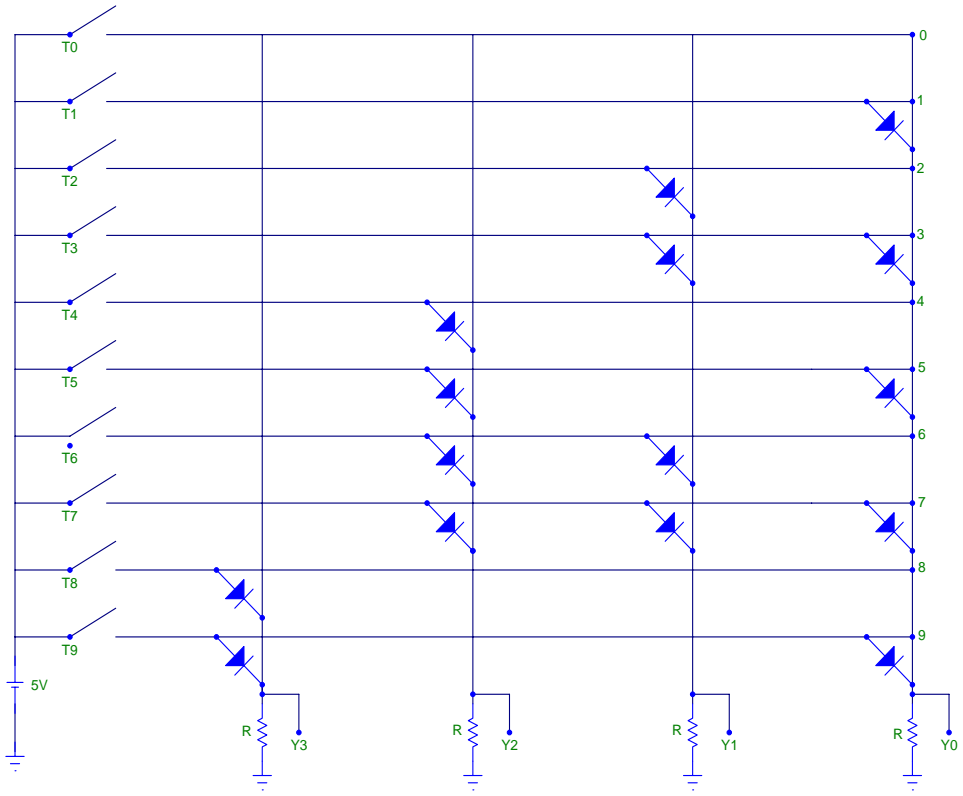


Figura 1.19:

T_9	T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	x	0	0	0	1
0	0	0	0	0	0	0	1	x	x	0	0	1	0
0	0	0	0	0	1	x	x	x	x	0	0	1	1
0	0	0	0	1	x	x	x	x	x	0	1	0	0
0	0	0	1	x	x	x	x	x	x	0	1	0	1
0	0	1	x	x	x	x	x	x	x	0	1	1	0
0	1	x	x	x	x	x	x	x	x	0	1	1	1
0	1	x	x	x	x	x	x	x	x	1	0	0	0
1	x	x	x	x	x	x	x	x	x	1	0	0	1

Tabella 1.34:

Questa si può semplificare in:

$$Y_0 = \overline{T}_9 \overline{T}_8 (T_7 + \overline{T}_7 B) + T_9$$

con:

$$B = \overline{T}_6 \overline{T}_5 \overline{T}_4 \overline{T}_3 \overline{T}_2 T_1 + \overline{T}_6 \overline{T}_5 \overline{T}_4 T_3 + \overline{T}_6 T_5 = \overline{T}_6 (T_5 + \overline{T}_5 C)$$

dove:

$$C = \overline{T}_4 \overline{T}_2 T_1 + \overline{T}_4 T_3 = \overline{T}_4 (\overline{T}_2 T_1 + T_3)$$

Ed infine:

$$Y_0 = T_9 + \overline{T}_8 (T_7 + \overline{T}_6 (T_5 + D))$$

con:

$$D = \overline{T}_4 (\overline{T}_1 T_1 + T_3)$$

Un decodificatore svolge un ruolo opposto; esso attiva una tra N linee d'uscita secondo l'informazione che riceve da due o più linee d'ingresso.

1.12 Multiplexer/Demultiplexer

Ammettiamo di voler "incanalare" su di un'unica linea dati che provengono da una di N linee diverse. Si può utilizzare a tale scopo un multiplexer. Il multiplexer ha N ingressi "dati", un ingresso "indirizzo", che specifica quale degli ingressi dati si vuol selezionare, ed un'unica uscita, sulla quale viene convogliato il dato. Un esempio di multiplexer è quello mostrato in figura 1.20

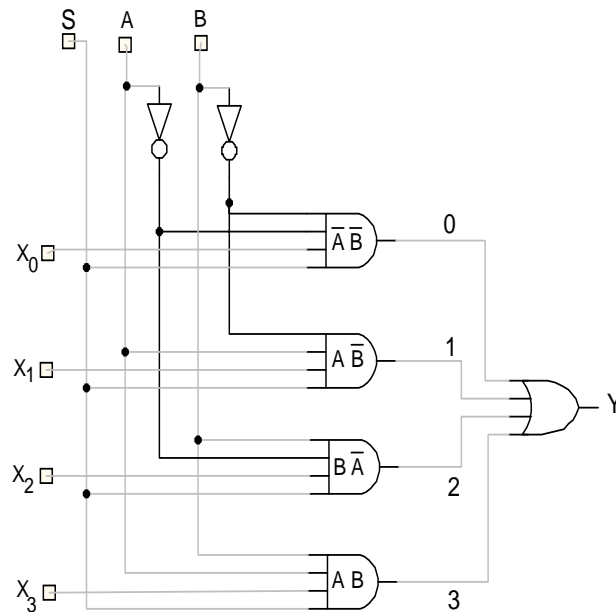


Figura 1.20:

Vediamo che se gli ingressi "indirizzo" A e B sono entrambi uguali a 0, $\overline{A} = 1$ e $\overline{B} = 1$ e la prima porta AND sarà abilitata ¹.

¹si dice che una porta è abilitata se essa è in grado di rispondere a variazioni dei segnali presenti sugli ingressi diversi da quello/quelli considerati di abilitazione. Ad esempio, se uno degli ingressi (considerato di abilitazione) di una porta AND a quattro ingressi è uguale a 0 l'uscita sarà 0 qualunque siano i valori presenti sugli altri ingressi. Si dice che in tal caso la porta è disabilitata

Quindi il segnale presente sulla linea d'ingresso X_0 sarà convogliato in uscita non appena l'ingresso di abilitazione S (Strobe) sia alto. Se $A=1$ e $B=0$, sarà abilitata la seconda porta e sarà quindi convogliato in uscita il segnale presente sulla linea X_1 , e così via.

Un multiplexer può esser utilizzato per effettuare la conversione parallelo-seriale. Ammettiamo infatti che ad un circuito arrivino dei bit, costituenti una "parola", su $n'=n+1$ linee parallele e che esso debba poi convogliare tali informazioni su di un'unica linea, come mostrato in figura 1.21.

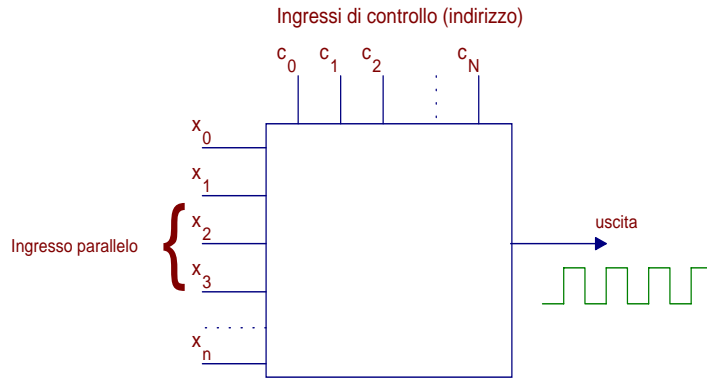


Figura 1.21:

Si potrà usare un multiplexer che riceve sulle linee x_0, x_1, \dots, x_n i bit da convogliare in uscita; gli ingressi di controllo (o di indirizzo) c_0, c_1, \dots, c_N , dovranno ricevere in successione temporale regolare i valori 000, 100, ..., 010, ..., 110, ... etc.. (con c_0 il bit meno significativo). In tal modo il bit presente sulla linea x_0 sarà convogliato per primo in uscita, seguirà poi x_1 , poi x_2 etc. Ovviamente il numero degli ingressi $n'=n+1$ dovrà esser legato al numero di ingressi di controllo $N'=N+1$ dalla relazione $n' = 2^{N'} - 1$. Così, per $n' = 4$ avremo $N' = 2$, per $n' = 8$, $N' = 3$ etc.

Un demultiplexer convoglia invece su una di $n'=n+1$ possibili linee di uscita i segnali che riceve su di una linea d'ingresso, facendo uso dell'informazione (indirizzo) che riceve su $N'=N+1$ linee di controllo. E' utile far presente che qualsiasi decodificatore può essere usato come demultiplexer utilizzando tutte le linee d'ingresso meno una per la funzione di selezione delle uscite.

Inoltre possiamo far vedere come un multiplexer possa essere adoperato per implementare una funzione logica. Consideriamo il multiplexer di figura 1.22

Gli ingressi sono x_0, x_1, x_2, x_3 . A e B sono gli ingressi di controllo (con A il meno significativo). S é lo Strobe. La tabella delle verità é quella di tabella 1.35.

Y può anche essere scritto come:

$$Y = x_0 \overline{A} \overline{B} + x_1 \overline{A} B + x_2 A \overline{B} + x_3 A B \quad (1.8)$$

Se ora volessimo implementare la seguente funzione logica:

$$Y = C \overline{B} \overline{A} + \overline{C} \overline{B} \overline{A} + C \overline{B} A + \overline{C} B A$$

notando che i primi due termini si possono semplificare in $\overline{B} \overline{A}$, cioè:

$$Y = \overline{B} \overline{A} + C \overline{B} A + \overline{C} B A$$

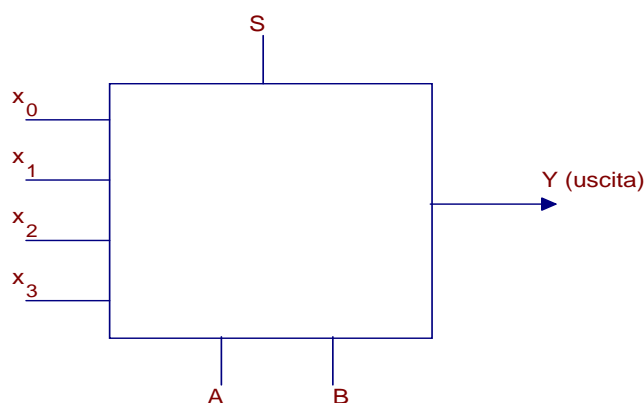


Figura 1.22:

B	A	S	Y
0	0	1	x_0
0	1	1	x_1
1	0	1	x_2
1	1	1	x_3

Tabella 1.35:

possiamo confrontare questa espressione con la 1.8, imponendo l'uguaglianza dei coefficienti che moltiplicano i termini $\overline{B}\overline{A}$, $\overline{B}A$, BA ottenendo:

$$x_0 = 1; \quad x_1 = C; \quad x_2 = 0; \quad x_3 = \overline{C}$$

Basterá quindi utilizzare il multiplexer indicato, assegnando agli ingressi x_0, x_1, x_2, x_3 i valori trovati, per implementare la funzione logica indicata.

I multiplexer/demultiplexer svolgono un ruolo di fondamentale importanza nei processori e piú in generale in tutti i sistemi digitali. Esistono in commercio numerosi integrati dedicati allo scopo. In figura 1.23 é illustrato il multiplexer 74LS151.

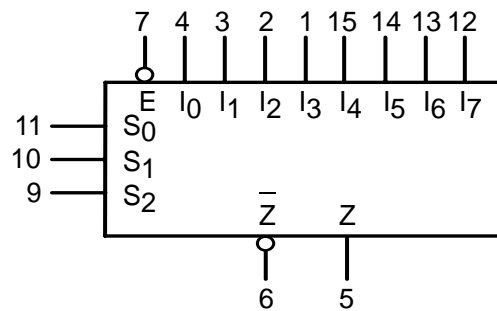
Gli ingressi sono indicati con $I_0 - I_7$. L'indirizzo, cioé il numero binario che rappresenta l'ingresso da selezionare, é impostato sui piedini $S_0 - S_2$. L'uscita Z ed il suo complemento sono disponibili sui piedini 5/6. Il piedino indicato con E (accompagnato da un'inversione), che in condizioni normali va tenuto basso, é l'ingresso di *enable*. Se si vuole inibire il trasferimento occorrerá portare tale ingresso al livello alto.

L'integrato 74LS138, il cui schema logico é mostrato in figura 1.24 é utilizzabile come demultiplexer da 1 ad 8.

1.13 Registri e memorie: i Flip-Flop

Perché un numero possa essere elaborato da un processore, esso deve essere preventivamente immagazzinato in un insieme di celle elementari o registri. Un tipico registro può contenere ad esempio un byte (8 bit), una parola di 16 bit o una parola

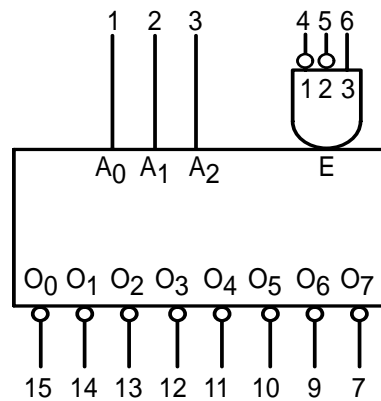
LOGIC SYMBOL



V_{CC} = PIN 16
GND = PIN 8

Figura 1.23:

LOGIC SYMBOL



V_{CC} = PIN 16
GND = PIN 8

Figura 1.24:

di 32 bit etc. Alla base di un generico registro é la cella elementare di memoria, spesso costituita da un *bistabile*, o flip-flop (FF). Un FF é un circuito con due ingressi (detti ingressi di Set e di Reset) e due uscite, convenzionalmente chiamate Q e \overline{Q} , una complementare dell'altra.

1.13.1 Flip-Flop di tipo RS

Esiste un'ampia varietà di FF. Quello che abbiamo sommariamente descritto é il flip-flop di tipo RS. Un FF RS può assumere due stati: nello stato di *Set*, le uscite sono: $Q = 1$, $\overline{Q} = 0$. Nello stato di *Reset* esse sono: $Q = 0$, $\overline{Q} = 1$. Vediamo così

che un FF può essere adoperato per memorizzare un bit. I valori delle uscite sono determinati dai valori dei segnali presenti sugli ingressi R, S, secondo la tabella delle verità mostrata nella 1.36.

R	S	Q_n	\overline{Q}_n
0	0	non consentito	
0	1	1	0
1	0	0	1
1	1	Q_{n-1}	\overline{Q}_{n-1}

Tabella 1.36:

Notiamo che lo stato in cui entrambi gli ingressi sono nello stato logico basso ($R=0$, $S=0$) é proibito. Lo stato in cui entrambi gli ingressi sono nello stato logico alto ($R=1$, $S=1$) é uguale a quello precedente l'applicazione di tali segnali sugli ingressi ($Q_n = Q_{n-1}$).

Il FF di tipo RS considerato é di tipo "asincrono" dove si intende che la transizione da uno stato all'altro é determinata esclusivamente dall'applicazione di determinati segnali sugli ingressi R ed S ed indipendente da eventuali impulsi di clock presenti nel sistema. Per veder ciò piú chiaramente, cominciamo con l'esaminare la struttura interna di un FF di tipo RS, che utilizza due porte NAND:

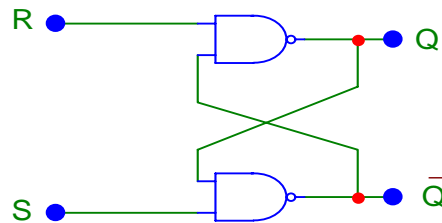


Figura 1.25:

Vediamo che se $S=0$ ed $R=1$, avremo $Q = 0$ e $\overline{Q} = 1$. Se $S=1$ ed $R=0$, si ha invece $Q = 1$ e $\overline{Q} = 0$. Se R ed S sono entrambi uguali ad 1, le uscite Q_n e \overline{Q}_n dipendono soltanto da Q_{n-1} e \overline{Q}_{n-1} , cioè dai valori che Q e \overline{Q} avevano prima dell'applicazione dei segnali su S ed R. In particolare, se era $Q_{n-1} = 0$ e $\overline{Q}_{n-1} = 1$, le uscite saranno $Q_n = 0$, $\overline{Q}_n = 1$. Se viceversa era $Q_{n-1} = 1$, $\overline{Q}_{n-1} = 0$, si avrà $Q_n = 1$, $\overline{Q}_n = 0$. Vediamo così che lo stato con $R=S=1$ non modifica le uscite Q e \overline{Q} .

Vediamo invece cosa accade nello stato "proibito" con $R=S=0$. Se un ingresso di una porta NAND é 0, l'uscita sarà 1 qualunque sia il segnale presente sull'altro ingresso (in tale situazione la porta é disabilitata). Vediamo quindi che, con $R=S=0$ avremo $Q = \overline{Q} = 1$. Tale situazione é anomala, poiché vogliamo limitarci a due stati: $(Q = 1 ; \overline{Q} = 0)$, $(Q = 0 ; \overline{Q} = 1)$.

C'è inoltre l'ulteriore problema costituito dal fatto che, in una eventuale successiva transizione allo stato con $S=R=1$, lo stato delle uscite Q e \overline{Q} verrebbe a dipendere

dal ritardo relativo tra i segnali applicati agli ingressi S ed R. Se la transizione di S avviene prima di quella di R, il sistema andrà nello stato di Set ($Q = 1$), altrimenti andrà in quello di Reset ($\overline{Q} = 1$). Poiché il ritardo relativo è un elemento aleatorio è chiaro come tale situazione non sia accettabile. Vedremo in seguito come in alcuni tipi di FF (ad esempio i JK) non esista alcuno stato proibito.

Si può ottenere un circuito con una tabella delle verità sostanzialmente uguale a quella precedente, facendo precedere ciascuno dei due gates NAND da un ulteriore NAND e scambiando i ruoli di Q e \overline{Q} come mostrato in figura 1.26:

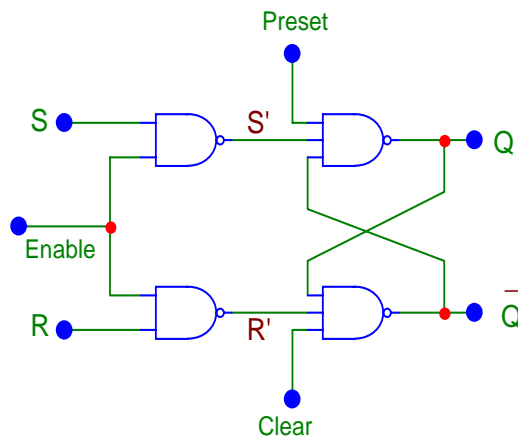


Figura 1.26:

Ammettiamo che gli ingressi di *Preset* e di *Clear* siano entrambi alti. Se l'*enable* è alto, le porte NAND, in ingresso saranno "abilitate" (cioè le loro uscite, S' ed R', dipenderanno dai segnali presenti sugli ingressi S ed R). Ciò è mostrato nella tabella 1.37, dove vediamo che ora lo stato "proibito" è quello che corrisponde ad $R=S=1$.

R	S	R'	S'	Q_n	\overline{Q}_n
0	0	1	1	Q_{n-1}	\overline{Q}_{n-1}
0	1	1	0	1	0
1	0	0	1	0	1
1	1	non consentito			

Tabella 1.37:

Se l'ingresso di enable è basso, le porte in ingresso risulteranno disabilitate (cioè non risponderanno più ai segnali presenti sugli ingressi R ed S). Avremo in tale situazione $R'=S'=1$ e $Q_n = Q_{n-1}$, $\overline{Q}_n = \overline{Q}_{n-1}$. Con gli ingressi di Preset e Clear entrambi alti, le due porte poste a valle risponderanno solo ai valori dei rimanenti ingressi.

L'ingresso che abbiamo chiamato "Enable" può essere utilizzato per un funzionamento "sincrono" di tale FF. Se cioè immaginiamo di aver applicato agli ingressi

R ed S due valori (ad esempio $R=1$, $S=0$), ed applichiamo all'ingresso di Enable un impulso di clock, finché questo è basso le uscite Q e \overline{Q} rimangono inalterate, mentre esse acquisteranno i valori $Q = 0$, $\overline{Q} = 1$ non appena il clock sia divenuto alto. L'impulso (positivo) del clock abilita cioè il trasferimento dei segnali presenti sugli ingressi R ed S, verso le uscite Q e \overline{Q} rispettivamente.

Gli ingressi di Preset e Clear che, come si è detto, vanno normalmente tenuti alti, possono essere adoperati per fissare dei valori iniziali prestabiliti delle uscite. Immaginiamo infatti di tenere basso l'impulso di clock, con che $S'=R'=1$. Se ora poniamo Preset=1 e Clear=0 avremo $\overline{Q} = 1$, $Q = 0$. Se invece poniamo Preset=0 e Clear=1, avremo $\overline{Q} = 0$, $Q = 1$. Ponendo successivamente Preset=Clear=1 il FF potrà nuovamente rispondere ai livelli presenti sugli ingressi R, S.

1.13.2 Flip-Flop di tipo D

Se tra gli ingressi S ed R di un FF di tipo RS poniamo un invertitore (NOT) come mostrato in figura, otteniamo un FF di tipo D (DATA).

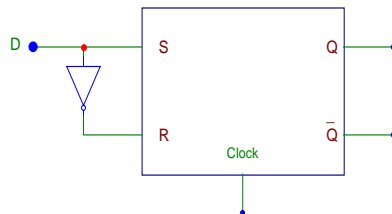


Figura 1.27:

La presenza dell'invertitore fa sì che se $S=1$, sia $R=0$ e viceversa. Tale FF ha un solo ingresso (D) al posto dei due ingressi separati S ed R. Il dato presente sull'ingresso D viene trasferito in Q quando l'impulso di clock diviene alto. Un esempio di FF di tipo D è quello mostrato in figura 1.28. La configurazione dei terminali (*pins*) è mostrata in figura 1.29.

Questo (74AHC74) è un chip che contiene 2 FF di tipo D. Ciascuno dei due FF contenuti nel chip ha, oltre all'ingresso D, gli ingressi di Preset e Clear, l'ingresso di clock e le uscite Q e \overline{Q} .²

Notiamo che in questo FF la transizione avviene nell'istante in cui il clock diviene alto. Una volta che tale livello del clock sia stato raggiunto, le uscite non risentono più di eventuali variazioni dell'ingresso D. Tale tipo di variazione dello stato di un FF è noto con il nome di "edge-triggering". Moltissimi dei FF disponibili in commercio

²Nella sigla che identifica il chip, le ultime due cifre (74) specificano la funzione effettuata (FF di tipo D nel nostro caso). Le prime due cifre numeriche (nuovamente 74, ma è un caso!) si riferiscono al range di temperatura in cui può operare: 74 implica un'applicazione "commerciale", per un range di temperatura che va da $0^{\circ}C$ a $70^{\circ}C$. Se trovassimo un 54 al posto di tali due cifre, ciò implicherebbe un dispositivo adatto ad applicazioni "militari", in un range di temperature compreso tra $-55^{\circ}C$ e $+125^{\circ}C$. Le due lettere intermedie (LS nel nostro caso) sta poi ad indicare la classe o tipo del chip. LS sta per "Low power Schottky". Altre sigle sono: S=Schottky; AS=Advanced Schottky; HC=High-Speed CMOS; HCU=HCMOS unbuffered; HCT=HCMOS con inputs TTL. I tipi 74-HC, 74HCU, 74HCT lavorano nel range di temperature che va da $-40^{\circ}C$ a $+85^{\circ}C$.

Dual D-type flip-flop with set and reset; positive-edge trigger

74AHC74; 74AHCT74

FEATURES

- ESD protection:
HBM EIA/JESD22-A114-A
exceeds 2000 V
MM EIA/JESD22-A115-A
exceeds 200 V
- Balanced propagation delays
- Inputs accepts voltages higher than V_{CC}
- For AHC only:
operates with CMOS input levels
- For AHCT only:
operates with TTL input levels
- Output capability: standard
- I_{CC} category: flip-flops
- Specified from
-40 to +85 and +125 °C.

DESCRIPTION

The 74AHC/AHCT74 are high-speed Si-gate CMOS devices and are pin compatible with low power Schottky TTL (LSTTL). They are specified in compliance with JEDEC standard No. 7A.

The 74AHC/AHCT74 dual positive-edge triggered, D-type flip-flops with individual data (D) inputs, clock (CP) inputs, set (\overline{S}_D) and reset (\overline{R}_D) inputs; also complementary Q and \overline{Q} outputs.

The set and reset are asynchronous active LOW inputs and operate independently of the clock input. Information on the data input is transferred to the Q output on the LOW-to-HIGH transition of the clock pulse. The D inputs must be stable one set-up time prior to the LOW-to-HIGH clock transition for predictable operation.

Schmitt-trigger action in the clock input makes the circuit highly tolerant to slower clock rise and fall times.

QUICK REFERENCE DATA

GND = 0 V; $T_{amb} = 25\text{ °C}$; $t_r = t_f \leq 3.0\text{ ns}$.

SYMBOL	PARAMETER	CONDITIONS	TYPICAL		UNIT
			AHC	AHCT	
t_{PHL}/t_{PLH}	propagation delay nCP to nQ, n \overline{Q} n \overline{S}_D , n \overline{R}_D to nQ, n \overline{Q}	$C_L = 15\text{ pF}$; $V_{CC} = 5\text{ V}$	3.7	3.3	ns
f_{max}	max. clock frequency		130	100	MHz
C_I	input capacitance	$V_I = V_{CC}$ or GND	4.0	4.0	pF
C_{PD}	power dissipation capacitance	$C_L = 50\text{ pF}$; $f = 1\text{ MHz}$; notes 1 and 2	12	16	pF

Notes

1. C_{PD} is used to determine the dynamic power dissipation (P_D in μW).

$$P_D = C_{PD} \times V_{CC}^2 \times f_i + \sum (C_L \times V_{CC}^2 \times f_o) \text{ where:}$$

f_i = input frequency in MHz; f_o = output frequency in MHz;

$\sum (C_L \times V_{CC}^2 \times f_o)$ = sum of outputs;

C_L = output load capacitance in pF;

V_{CC} = supply voltage in Volts.

2. The condition is $V_I = \text{GND to } V_{CC}$.

FUNCTION TABLES

Table 1 See note 1

INPUT				OUTPUT	
n \overline{S}_D	n \overline{R}_D	nCP	nD	nQ	n \overline{Q}
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H	H

Table 2 See note 1

INPUT				OUTPUT	
n \overline{S}_D	n \overline{R}_D	nCP	nD	nQ _{n+1}	n \overline{Q} _{n+1}
H	H	↑	L	L	H
H	H	↑	H	H	L

Note to Tables 1 and 2

1. H = HIGH voltage level;
L = LOW voltage level;
X = don't care;
↑ = LOW-to-HIGH CP transition;
Q_{n+1} = state after the next LOW-to-HIGH CP transition.

Figura 1.28:

sono di questo tipo. Gli ingressi asincroni (\overline{S}_D e \overline{R}_D) sono *active low*, cioè hanno un effetto solo se vengono tenuti bassi. Durante il normale funzionamento del FF con trasferimento del dato (D) effettuato dal clock, essi vanno tenuti *alti*.

In un normale FF di tipo D, l'output segue l'input in corrispondenza al fronte di salita dell'impulso di clock. Esiste un tipo particolare di D FF, noto come LATCH, in cui l'output diviene una copia dell'ingresso quando il clock é alto, e conserva tale valore quando il clock diviene basso. In altre parole, la differenza tra i due tipi é che un normale D FF é "edge triggered" mentre un LATCH é "level triggered". Data la similarità con il D, si conserva la dizione "registro di tipo D" per il normale D e si

PINNING

PIN	SYMBOL	DESCRIPTION
1 and 13	$1\bar{R}_D$ and $2\bar{R}_D$	asynchronous reset-direct input (active LOW)
2 and 12	1D and 2D	data inputs
3 and 11	1CP and 2CP	clock input (LOW-to-HIGH, edge-triggered)
4 and 10	$1\bar{S}_D$ and $2\bar{S}_D$	asynchronous set-direct input (active LOW)
5 and 9	1Q and 2Q	true flip-flop outputs
6 and 8	$1\bar{Q}$ and $2\bar{Q}$	complement flip-flop outputs
7	GND	ground (0 V)
14	V_{CC}	DC supply voltage

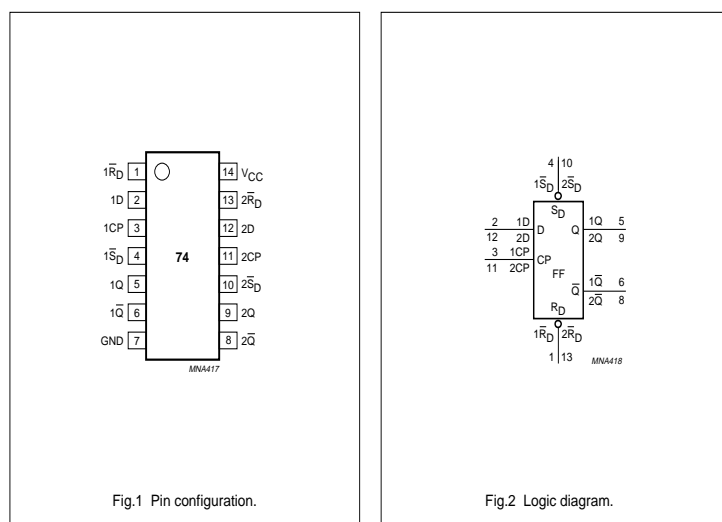


Figura 1.29:

usa quella di "transparent latch" per il nuovo registro (in cui l'output "si aggancia" all'ingresso per l'intera durata dell'impulso di clock.

Un esempio di FF di tipo D é il '574, mentre un latch é il '573. Questi, come mostrato in figura 1.30 sono dei chip con 8 FF in ciascuna unità (sono cioè di tipo "octal"). Il '574 ha un ingresso di clock indicato con *cp* e con il simbolo > che denota un trigger sul fronte di salita del clock. Il '573 ha l'analogo ingresso indicato con LE (Latch Enable). Entrambi hanno un ulteriore ingresso di abilitazione/disabilitazione, indicato con *OEBAR*.

1.13.3 Flip-Flop di tipo JK

Un tipo di FF che ha un ampio campo di applicazioni é il JK. Questo differisce da quello di tipo RS nel fatto che ora lo stato in cui entrambi gli ingressi (che ora si chiamano J e K) sono alti é consentito. In tale stato, l'arrivo di un impulso di clock fa passare le uscite del FF negli stati complementari di quelli in cui erano prima: $Q_n = \bar{Q}_{n-1}$, $\bar{Q}_n = Q_{n-1}$. Ciò può essere ottenuto facendo precedere ciascuno dei due ingressi di un FF RS, da una porta AND, a sua volta pilotata dall'uscita complementare (la superiore da \bar{Q} , quella inferiore da Q), come mostrato in figura 1.31.

Se $J=1$ e $K=0$, esaminiamo i due possibili stati di Q e \bar{Q} :

a) $Q = 1$, $\bar{Q} = 0$

In tal caso avremo $S=0$, $R=0$ e quindi lo stato del FF non cambierà all'arrivo dell'impulso di clock.

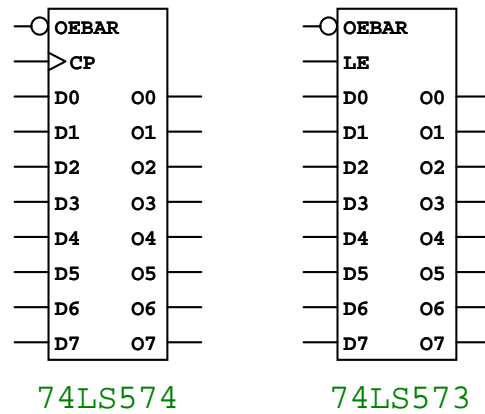


Figura 1.30:

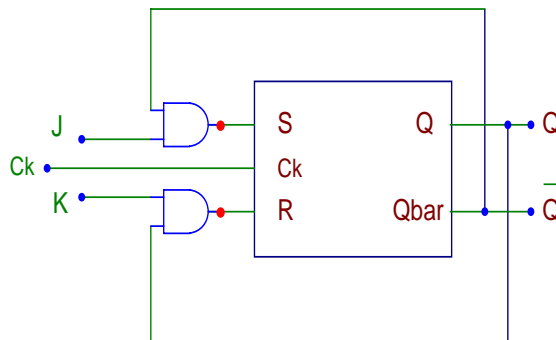


Figura 1.31:

b) $Q = 0, \overline{Q} = 1$

Ora avremo $R=0, S=1$. In tal caso, all'arrivo dell'impulso di clock le uscite acquisteranno i valori:

$Q = 1, \overline{Q} = 0$

Se era $J=0, K=1$ possiamo anticipare, data la simmetria del circuito, che le uscite saranno $Q = 0, \overline{Q} = 1$ dopo l'applicazione di un impulso di clock. Verifichiamolo, esaminando i due possibili stati del FF:

a) $Q = 1, \overline{Q} = 0$

In tal caso avremo: $S=0, R=1$; da cui seguirà $Q = 0, \overline{Q} = 1$.

b) $Q = 0, \overline{Q} = 1$

In tal caso avremo: $R=0, S=0$ e quindi $Q_n = Q_{n-1}$.

Esaminiamo infine lo stato $J=K=1$. In tal caso, se era $Q_{n-1} = 1, \overline{Q}_{n-1} = 0$ avremo che la porta AND inferiore é abilitata ($Q=1, K=1$) mentre é disabilitata quella superiore (poiché $\overline{Q} = 0$). Ne segue che, con l'arrivo dell'impulso di clock sarà $Q = 0, \overline{Q} = 1$, cioè il sistema passerà nello stato complementare di quello in cui era. Il successivo impulso di clock troverà ora abilitata la porta superiore e disabilitata quella inferiore. Di conseguenza, con tale impulso di clock Q diventerà

1 e \overline{Q} 0.

Il sistema cambia quindi stato ad ogni impulso di clock (fintantoché J e K vengono mantenuti fissi al livello alto). Ciò é mostrato nel grafico di figura 1.32:

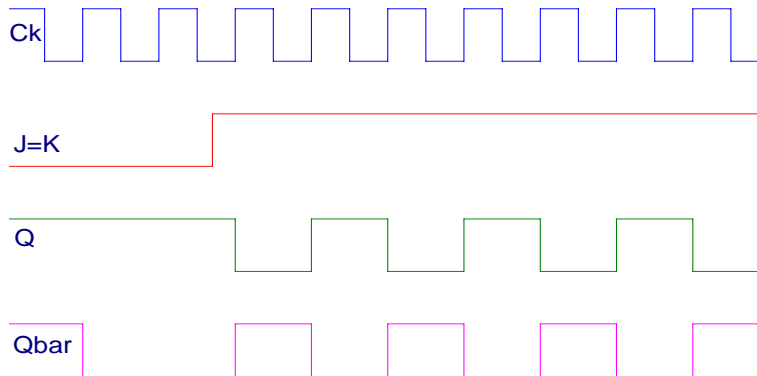


Figura 1.32:

Notiamo che la frequenza dell'onda rettangolare $Q(\overline{Q})$ é la metà di quella del clock. Un FF di tipo JK con $J=K=1$ effettua quella che vien chiamata la funzione "TOGGLE" ed é adoperato come divisore di frequenza. Ponendone due o piú in cascata si può realizzare una divisione della frequenza del clock per 4 o per potenze di 2 piú elevate. L'operazione di questo circuito potrà presentare dei problemi, poiché l'uscita (Q) tenderá ad oscillare tra 0 ed 1 nell'intervallo di tempo t_c in cui il clock é alto. Per esempio, se $J=K=1$ e $Q=0$, all'arrivo di un impulso di clock Q diverrá 1 dopo un piccolo ritardo Δt (dipendente dalla struttura interna del dispositivo). A questo punto avremo $J=K=1$, $Ck=1$ e $Q=1$. Ora Q tenderá a divenire 0, a meno che il Ck non sia nel frattempo sceso a 0. L'uscita Q altrimenti oscillerá a lungo tra 0 ed 1 ed alla fine dell'impulso di clock il suo stato sará casuale. Tale fenomeno va sotto il nome di "race-around-condition" ed ha luogo se $\Delta t \ll t_c$. Esso può essere eliminato facendo uso di:

- FF di tipo Master-Slave
- FF di tipo Edge-Triggered

Il principio su cui sono basati i primi é illustrato in figura 1.33 (dove ammetteremo che sia $J=K=1$).

Il primo dei due FF RS effettua la transizione quando il clock é alto. Tuttavia in tale intervallo di tempo lo stato di Q e \overline{Q} (uscita del secondo FF) non può cambiare. Infatti, la presenza dell'invertitore prima del Ck_2 fa sí che in tale intervallo di tempo il secondo FF sia disabilitato. La transizione di Q_2 e \overline{Q}_2 avviene solo quando il segnale di clock diviene basso. Il comportamento di questo FF differisce da quello di un normale JK, per il fatto che la transizione dell'uscita avviene sull'impulso negativo del clock.

I FF del tipo edge-triggered non rispondono al livello del clock, ma al fronte di discesa di questo. In questo caso gli ingressi non hanno effetto sulle uscite se non in corrispondenza della transizione negativa dell'impulso di clock.

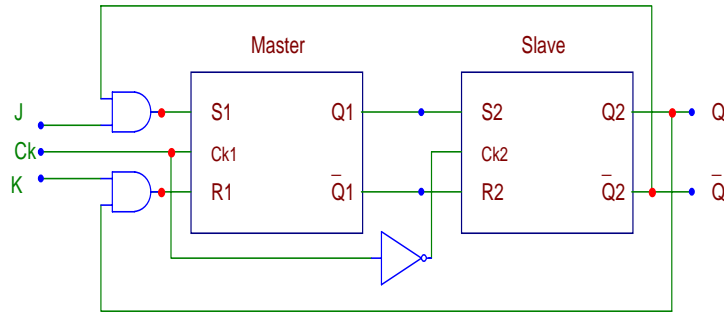


Figura 1.33:

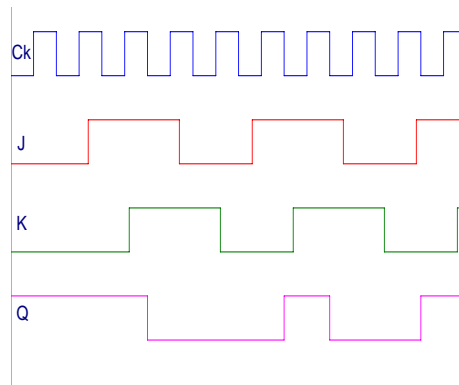


Figura 1.34:

La figura 1.34 mostra un esempio di forme d'onda per un FF JK del tipo Master-Slave.

Come esempi di FF Master-Slave JK possiamo menzionare il modello SN74LS76 della Motorola. Questo chip ha due FF JK, come mostrato in figura 1.35. Ciascuno dei due FF ha, oltre agli ingressi J e K, due ingressi di Set e Reset asincroni, denotati con \overline{S} ed \overline{C} nella tabella di figura.

La tabella 1.38 mostra un elenco di alcuni dei piú comuni tipi di FF disponibili in commercio. Per ciò che riguarda i simboli che vengono normalmente adoperati dai produttori nelle specifiche dei FF, dobbiamo ricordare che un FF con transizione sul fronte di salita del clock é comunemente specificato con un $>$ sull'ingresso del clock. Un FF con transizione sul fronte di discesa del clock é indicato con un'inversione "o" prima del medesimo simbolo. Vedasi ad esempio la figura 1.36. Quello a sinistra é un FF di tipo D con transizione sul fronte di salita del clock, quello a destra é un JK con transizione sul fronte di discesa.



DUAL JK FLIP-FLOP WITH SET AND CLEAR

The SN54/74LS76A offers individual J, K, Clock Pulse, Direct Set and Direct Clear inputs. These dual flip-flops are designed so that when the clock goes HIGH, the inputs are enabled and data will be accepted. The Logic Level of the J and K inputs will perform according to the Truth Table as long as minimum set-up times are observed. Input data is transferred to the outputs on the HIGH-to-LOW clock transitions.

MODE SELECT — TRUTH TABLE

OPERATING MODE	INPUTS				OUTPUTS	
	S _D	C _D	J	K	Q	Q̄
Set	L	H	X	X	H	L
Reset (Clear)	H	L	X	X	L	H
Undetermined	L	L	X	X	h	h
Toggle	H	H	h	h	q	q
Load "0" (Reset)	H	H	l	h	L	H
Load "1" (Set)	H	H	h	l	H	L
Hold	H	H	l	l	q	q

*Both outputs will be HIGH while both S_D and C_D are LOW, but the output states are unpredictable if S_D and C_D go HIGH simultaneously.

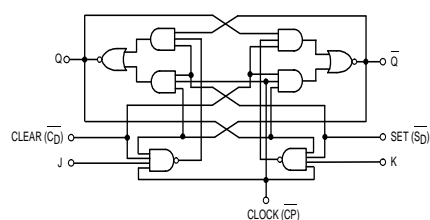
H, h = HIGH Voltage Level

L, l = LOW Voltage Level

X = Immaterial

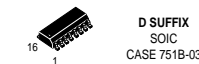
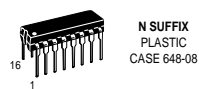
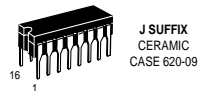
l, h (q) = Lower case letters indicate the state of the referenced input (or output) one setup time prior to the HIGH-to-LOW clock transition

LOGIC DIAGRAM



SN54/74LS76A

DUAL JK FLIP-FLOP
WITH SET AND CLEAR
LOW POWER SCHOTTKY



ORDERING INFORMATION

SN54LSXXJ Ceramic
SN74LSXXN Plastic
SN74LSXXD SOIC

LOGIC SYMBOL

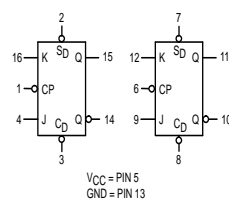


Figura 1.35:

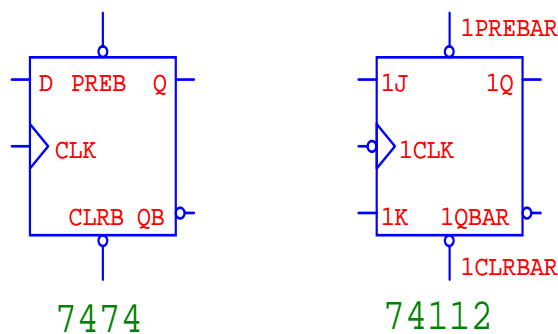


Figura 1.36:

1.14 Shift-Registers

Gli *shift-registers* (SR) (o registri a scorrimento), sono essenzialmente delle catene di FF, opportunamente collegati tra loro ed utilizzati per memorizzare una "stringa"

Dispositivo	Funzione
70	JK con ingressi di Preset e Clear
71	RS con ingressi di Preset e Clear
72	JK con ingressi di Preset e Clear
73	Dual JK
74	Dual tipo D
76	Dual JK
105	JK con clock e con ingressi di Preset e Clear
175	Quad tipo D
273	Octal tipo D
276	Quad JK
375	Octal tipo D

Tabella 1.38:

di bits. Esiste una grande varietà di SR. Consideriamo come primo esempio uno SR seriale realizzato con FF di tipo D. Ricordiamo che in tale tipo di FF il dato presente sull'ingresso D viene trasferito in uscita (Q) all'arrivo di un impulso di clock (in genere sul fronte di salita di questo). Uno SR costituito da FF di tipo D é mostrato in figura 1.37.

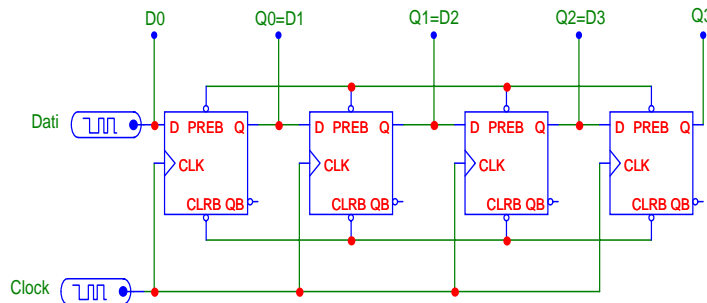


Figura 1.37:

Ammettiamo che un 1 (livello alto) sia applicato all'ingresso (D_0) del primo FF, mentre gli ingressi di Preset e Clear di tutti i FF vengono mantenuti alti. L'applicazione di un segnale di clock trasferirà tale dato in $Q_0 = D_1$. Il successivo segnale di clock trasferirà il dato in $Q_1 = D_2$, il terzo lo trasferirà in $Q_2 = D_3$ ed infine il quarto in Q_3 . Alla fine (ammettendo che D_0 sia messo a zero dopo il primo impulso di Clock) le uscite Q_0, Q_1, Q_2, Q_3 conterranno nell'ordine i bits (0,0,0,1). Se applicassimo in ingresso la sequenza di bit (1001) ed in fase con tale applicazione gli impulsi di clock, avremmo la situazione illustrata in figura 1.38.

Vediamo che, dopo l'applicazione del quarto impulso di clock, nei registri Q_0, \dots, Q_3 sono contenuti i bit 1001. Abbiamo così trasformato una distribuzione seriale di bits, applicati sequenzialmente in ingresso, in una parallela, sulle 4 linee d'uscita $Q_3 - Q_0$.

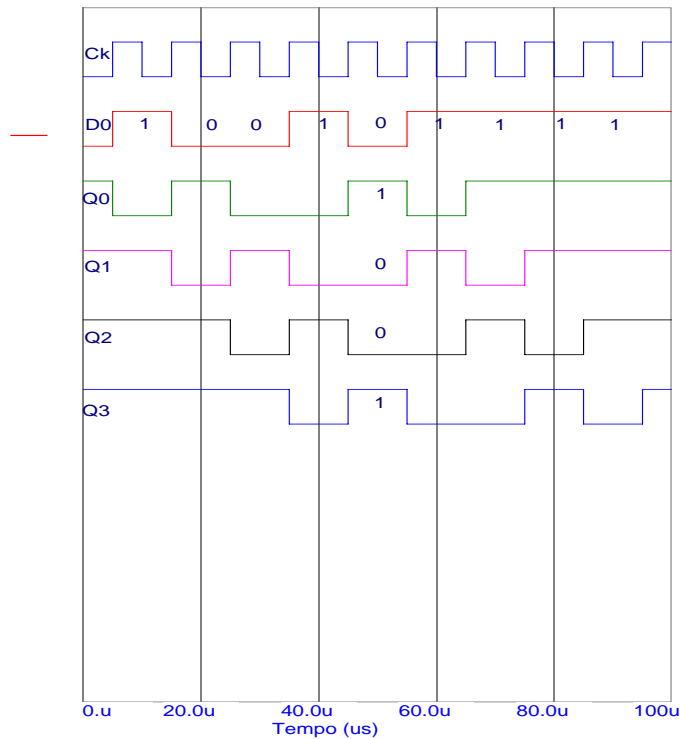


Figura 1.38:

Notiamo che, se vogliamo che il numero rimanga nel registro dobbiamo, dopo il quarto impulso di clock, arrestare quest'ultimo. I dati possono alternativamente esser prelevati in forma seriale applicando altri quattro impulsi di clock. Questo tipo di registro, che ha un ingresso seriale ed un'uscita parallela, è chiamato "Serial-In-Parallel-Out" (o SIPO).

1.14.1 Universal Shift-Register

Il circuito che ora esamineremo può essere utilizzato indifferentemente per immettere dei dati in parallelo ed estrarli in modo seriale, o viceversa. Esso può anche essere adoperato per leggere i dati in serie ed estrarli in serie, o per leggerli in parallelo ed estrarli in parallelo. A tale scopo si fa uso di una linea di controllo per l'abilitazione dell'input in parallelo. Un esempio di tale circuito (SN74LS195A) è mostrato in figura 1.39. Esso è noto come "Universal Shift Register".

Tale dispositivo è utilizzabile in un'ampia classe di applicazioni, che vanno da operazioni di "shift" al conteggio ed all'immagazzinamento dei dati. Esso, operando a velocità elevate (dell'ordine di 40 MHz), può effettuare trasferimento dei dati parallelo/seriale, seriale parallelo e shift seriale.

La tabella delle verità mostrata in figura 1.40 illustra le caratteristiche di funzionamento.

I dati sono immagazzinati nei quattro FF mostrati in figura. Se l'ingresso indicato con \overline{PE} (Preset Enable) è alto, i dati possono essere caricati in modo seriale attraverso gli ingressi J e \overline{K} che sono in genere uniti insieme. Se infatti $J = \overline{K} = 1$, ammettendo che sia $Q_0 = 0$ ($\overline{Q}_0 = 1$) la prima delle tre porte AND nel primo blocco a sinistra in figura avrà l'uscita alta e, a causa della doppia inversione presente all'u-

SN54/74LS195A

LOGIC DIAGRAM

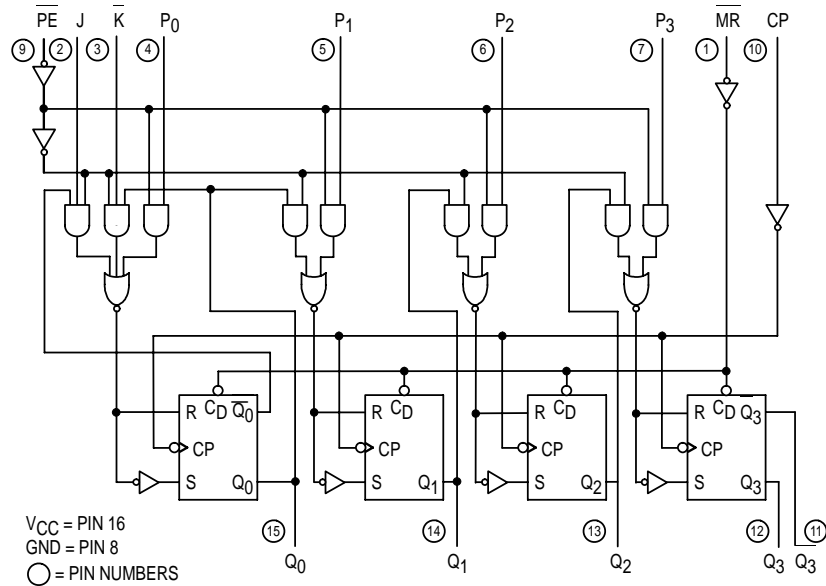


Figura 1.39:

MODE SELECT — TRUTH TABLE

OPERATING MODES	INPUTS					OUTPUTS				
	MR	PE	J	K	P _n	Q ₀	Q ₁	Q ₂	Q ₃	Q ₃
Asynchronous Reset	L	X	X	X	X	L	L	L	L	H
Shift, Set First Stage	H	h	h	h	X	H	q ₀	q ₁	q ₂	q ₂
Shift, Reset First	H	h	l	l	X	L	q ₀	q ₁	q ₂	q ₂
Shift, Toggle First Stage	H	h	h	l	X	q ₀	q ₀	q ₁	q ₂	q ₂
Shift, Retain First Stage	H	h	l	h	X	q ₀	q ₀	q ₁	q ₂	q ₂
Parallel Load	H	l	X	X	p _n	p ₀	p ₁	p ₂	p ₃	p ₃

L = LOW voltage levels

H = HIGH voltage levels

X = Don't Care

l = LOW voltage level one set-up time prior to the LOW to HIGH clock transition.

h = HIGH voltage level one set-up time prior to the LOW to HIGH clock transition.

p_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.

Figura 1.40:

scita della porta OR sottostante ed all'ingresso Set del primo FF, avremo $Set = 1$. Se Q_0 fosse stato uguale ad 1 ($\overline{Q_0} = 0$), sarebbe stata abilitata la seconda anziché la prima delle due porte AND associate al primo blocco e l'uscita della porta OR sarebbe stata comunque uguale ad 1. Quindi, con $J = \overline{K} = 1$ l'ingresso di Set del primo FF sarà uguale all'ingresso J. Di conseguenza l'uscita Q_0 diverrà 1 con il successivo impulso di Clock (più esattamente, sul fronte di salita di questo).

Notiamo per inciso che, con il \overline{PE} alto, la terza porta AND associata al primo blocco é disabilitata. In tali condizioni l'uscita della porta sar  uguale a 0, qualunque sia il valore presente in P_0 .

In modo analogo, se $J = \overline{K} = 0$, l'ingresso di Set del primo FF sar  uguale a 0 e quello di Reset 1. Con ci , l'uscita Q_0 diverr  uguale a 0 in corrispondenza al fronte di salita del prossimo impulso di Clock. Tale caricamento seriale del dato   quello indicato nella seconda e terza riga della tabella (Shift-Set First Stage - Shift-Reset) di figura 1.40.

Esaminiamo ora ci  che accade con l'arrivo dei successivi impulsi di Clock. Notiamo che, con l'ingresso \overline{PE} alto, la prima porta AND di ciascuno dei blocchi mostrati in figura   abilitata (e la terza nel primo blocco disabilitata). Se esaminiamo la prima porta AND associata al secondo blocco, vediamo che essa ha come ingresso Q_0 . Il collegamento della porta OR sottostante   tale che il Set del relativo FF   anch'esso uguale a Q_0 . Vediamo cos  che il successivo impulso di Clock far  si che Q_1 divenga uguale a Q_0 (che era 1 nel nostro esempio iniziale). In modo analogo vediamo che il collegamento dell'uscita Q_1 del secondo FF alla prima porta AND del terzo blocco far  si che l'uscita Q_2 divenga uguale a Q_0 con l'arrivo del terzo impulso di Clock. Infine, il quarto impulso di Clock far  si che Q_3 divenga uguale a Q_0 . Lo shift di quattro posizioni del dato impostato in J   cos  completato con l'applicazione di quattro impulsi di Clock.

Se il \overline{PE}   basso, i dati possono essere caricati in parallelo facendo uso degli ingressi P_0, P_1, P_2, P_3 . Infatti, con $\overline{PE} = 0$, delle porte AND presenti in ciascuno dei quattro blocchi, solo quelle all'estrema destra saranno abilitate. Ammettiamo ora ad esempio che sia: $P_0 = 1, P_1 = 0, P_2 = 1, P_3 = 0$. Ci  implicher : $S_0 = 1, S_1 = 0, S_2 = 1, S_3 = 0$.

Questi dati saranno trasferiti in Q_0, Q_1, Q_2, Q_3 sul fronte di salita del successivo impulso di Clock. Si   in tal modo effettuato il *caricamento* dei dati impostati sugli ingressi ($P_0 \cdots P_3$) in ($Q_0 \cdots Q_3$). Ci    indicato nell'ultima riga della tabella di figura 1.40.

Se ora si vuole effettuare lo shift dei dati verso destra, occorre mettere alto il \overline{PE} . In tal modo il successivo impulso di Clock trover  abilitata la prima porta AND di ciascun blocco, collegata all'uscita Q del blocco precedente. Il dato sar  cos  trasferito da Q_i a Q_{i+1} .

L'ingresso indicato con \overline{MR} (Master Reset) pu , come mostrato nella prima riga della medesima tabella, essere adoperato per porre uguali a zero le uscite $Q_0 \dots Q_3$ di tutti i FF della catena.

Esaminiamo infine il funzionamento del circuito quando (J, \overline{K}) assumono i valori (0,1) o (1,0), con \overline{PE} alto (quarta e quinta riga della tabella delle verit ).

Se $J = 1$ e $\overline{K} = 0$, la prima delle porte AND presenti nel primo blocco di figura sar  abilitata, mentre la seconda   disabilitata. In tali condizioni l'uscita del primo AND dipender  dal valore di \overline{Q}_0 e quindi di Q_0 . Se \overline{Q}_0   alto (Q_0 basso) l'uscita dell'AND sar  alta e tale risulter  quindi l'ingresso di Set del primo FF. Con ci  $Q_0 \rightarrow 1$ e $\overline{Q} \rightarrow 0$ con il successivo impulso di Clock. Se fosse stato $Q_0 = 1$ la transizione causata dall'arrivo dell'impulso di Clock sarebbe stata opposta. Vediamo cos  che con $J = 1$ e $\overline{K} = 0$ il primo FF effettua la funzione "TOGGLE" caratteristica di un JK. Ci    indicato come "Shift, Toggle First Stage" nella quarta riga della tabella.

Il secondo FF, come   facile verificare, acquister  il valore di Q_0 (cio  1) con il secondo impulso di Clock, il terzo FF con il terzo impulso di Clock etc.

Se invece $J = 0$ e $\overline{K} = 1$, vediamo che verrà ad essere abilitata la seconda delle porte AND presenti nel primo blocco. Poiché questa ha come terzo ingresso Q_0 , vediamo che se $Q_0 = 0$ l'uscita di tale porta, e quindi l'ingresso di Set del FF, sarà bassa. Con ciò sarà alto l'ingresso di Reset del FF e Q_0 rimarrà uguale a 0. Se Q_0 fosse stato uguale ad 1 sarebbe invece stato alto l'ingresso di Set del FF e Q_0 sarebbe rimasto uguale ad 1. Tale funzione è indicata con "Shift, Retain First Stage" nella quinta colonna della tabella.

Un registro a scorrimento simile per molti rispetti al precedente, ma con la possibilità di shift sia a destra che a sinistra, è il SN74LS194, la cui struttura è mostrata in figura 1.41 e la cui tabella delle verità è illustrata in figura 1.42.

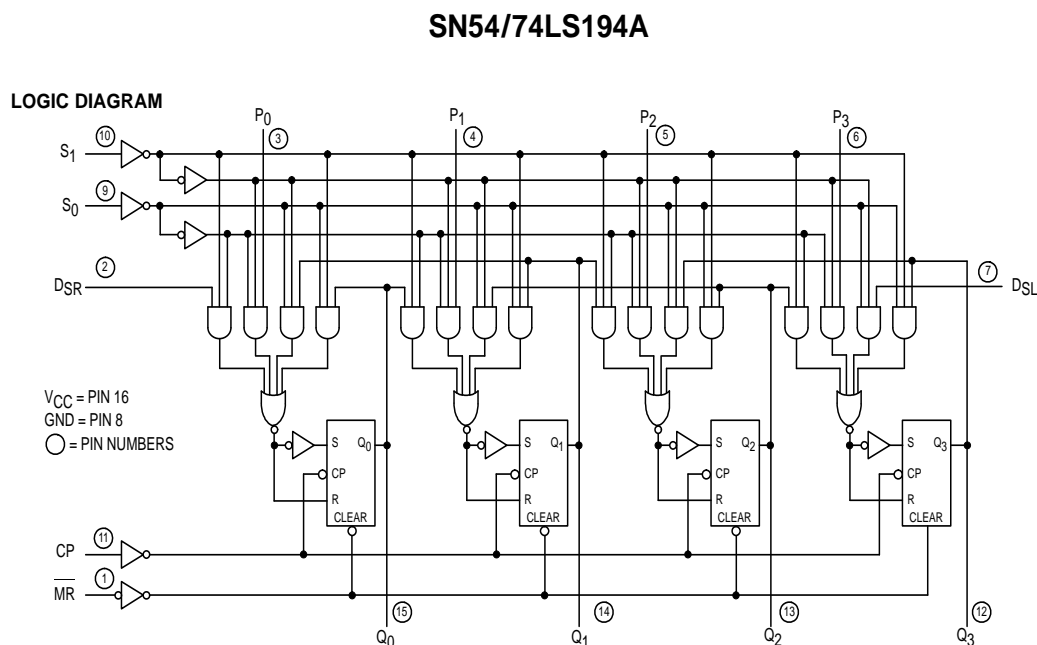


Figura 1.41:

1.14.2 First-In, First-Out SR

Un ulteriore tipo di registro che è ampiamente adoperato è il cosiddetto "First-In, First-Out" (FIFO). Lo schema è mostrato in figura 1.43.

Come vediamo, esso consiste di un normale SR con possibilità di ingressi paralleli. Il dato in input è applicato alla porta AND G_D (in modo seriale). Le porte $G_0 - G_3$ sono abilitate dagli ingressi $Q_0 \cdots Q_3$, uno solo dei quali è normalmente alto. Un opportuno circuito di comando fa sì che gli ingressi Q_0, Q_1, Q_2, Q_3 siano abilitati consecutivamente in corrispondenza ai quattro impulsi di clock che agiscono su G_D abilitando l'immissione sulla linea L del dato corrispondente. Vediamo ciò più in dettaglio:

- (a) il primo impulso di clock abilita G_D e manda $Q_0 \rightarrow 1$. Simultaneamente (o poco prima) il dato (primo bit) è immesso sulla linea L. L'abilitazione di G_0 fa sì che il dato sia caricato in Q_0 .

MODE SELECT — TRUTH TABLE

OPERATING MODE	INPUTS						OUTPUTS			
	MR	S ₁	S ₀	D _{SR}	D _{SL}	P _n	Q ₀	Q ₁	Q ₂	Q ₃
Reset	L	X	X	X	X	X	L	L	L	L
Hold	H	l	l	X	X	X	q ₀	q ₁	q ₂	q ₃
Shift Left	H	h	l	X	l	X	q ₁	q ₂	q ₃	L
	H	h	l	X	h	X	q ₁	q ₂	q ₃	H
Shift Right	H	l	h	l	X	X	L	q ₀	q ₁	q ₂
	H	l	h	h	X	X	H	q ₀	q ₁	q ₂
Parallel Load	H	h	h	X	X	P _n	P ₀	P ₁	P ₂	P ₃

L = LOW Voltage Level

H = HIGH Voltage Level

X = Don't Care

l = LOW voltage level one set-up time prior to the LOW to HIGH clock transition

h = HIGH voltage level one set-up time prior to the LOW to HIGH clock transition

p_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.

Figura 1.42:

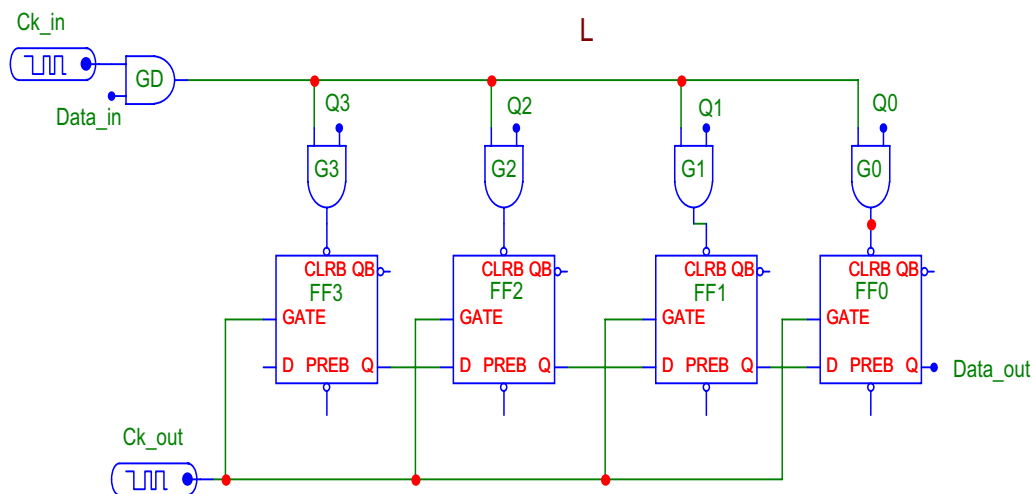


Figura 1.43:

- (b) il secondo bit é immesso in ingresso ed il secondo impulso di clock applicato. Questo abilita G_D e manda $Q_0 \rightarrow 0$ e $Q_1 \rightarrow 1$. In tal modo G_1 viene ad essere abilitato ed il dato (secondo bit) caricato in Q_1 .
- (c) il terzo bit é immesso in ingresso ed il terzo impulso di clock applicato. Questo abilita G_D e manda $Q_0, Q_1 \rightarrow 0$ e $Q_2 \rightarrow 1$. Ora G_2 sarà abilitato ed il terzo bit caricato in Q_2 .
- (d) il quarto bit é immesso in ingresso ed il quarto impulso di clock applicato. Questo abilita G_D e manda $Q_0, Q_1, Q_2 \rightarrow 0$ e $Q_3 \rightarrow 1$. Ora G_3 sarà abilitato ed il quarto bit caricato in Q_3 .

Notiamo che i FF adoperati sono di tipo D, con ingressi di preset e clear. L'ingresso di clear é quello indicato in figura mentre quello di preset é ottenuto complementando questo mediante un invertitore (non indicato in figura).

La lettura é effettuata applicando impulsi di clock alla linea indicata da Ck_{out} . Ciascun impulso di clock genera uno shift a destra, di una posizione, di tutti i bit memorizzati. Da notare che il contatore (non indicato) che abilita in successione le porte AND $G_3 - G_0$ viene decrementato durante l'applicazione degli impulsi Ck_{out} . In altre parole ammettendo che dopo il quarto impulso di Ck_{in} avessimo $Q_0 = Q_1 = Q_2 = Q_3 = 0$, l'applicazione di un impulso di clock (Ck_{out} pone $Q_0 = Q_1 = Q_2 = 0$ e $Q_3 = 1$. Se a questo punto immettiamo un nuovo dato, questo sará caricato in FF3. Il dispositivo che effettua tale operazione é un "up-down" counter, che "conta" gli impulsi di clock che gli arrivano e mette degli opportuni 1 su quattro linee d'uscita, in sequenza crescente (cioé 0000, 0001, 0010, 0011, ...) se il clock in arrivo é Ck_{in} ; decrescente (cioé, ... 0011, 0010, 0001, 0000) se il clock in ingresso é Ck_{out} . Ciò é mostrato schematicamente in figura 1.44.

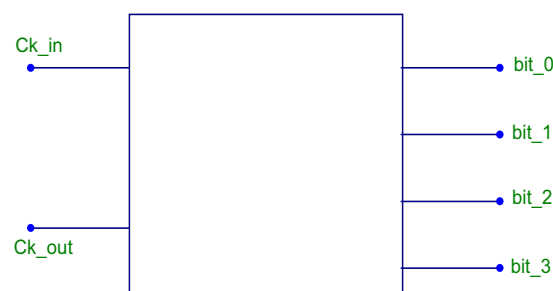


Figura 1.44:

1.15 Applicazioni degli Shift-Registers e dei Flip-Flop

1.15.1 Applicazioni degli shift-registers

Come accennato, questi possono essere adoperati per memorizzare stringhe di bits, quali ad esempio bytes o parole di 16/32 bits. Essi hanno tuttavia molte altre applicazioni, non meno importanti. Notiamo intanto che se facciamo scorrere verso destra (right-shift) il contenuto di un registro, perdiamo il bit meno significativo. Se questo era uguale a 0, il numero immagazzinato viene ad essere diviso per due come conseguenza di tale operazione. Se esso era invece 1, allora il numero originario meno 1 viene ad essere diviso per 2. Viceversa, uno spostamento a sinistra (left-shift) moltiplica per 2 il numero che era stato inizialmente caricato nel registro. Vediamo quindi che, con successivi shift-left/shift-right possiamo moltiplicare/dividere per potenze di 2 un dato numero. Ovviamente, nel corso di uno shift-right perdiamo i bit meno significativi, mentre perdiamo quelli piú significativi nel corso di uno shift-left. Questo non é un problema se il registro é molto lungo.

Un'altra applicazione degli SR é la conversione serie/parallelo e viceversa. Se carichiamo un numero, ad esempio di 8 bit, in uno SR e lo facciamo in parallelo sugli 8 FF di cui il registro é costituito, possiamo estrarre poi gli 8 bit in serie,

applicando 8 impulsi di clock. Viceversa, possiamo caricare gli 8 bit in modo seriale e leggerli in parallelo sulle 8 linee collegate ai piedini Q_i degli 8 FF.

Gli SR possono inoltre essere convenientemente utilizzati per generare sequenze predeterminate di impulsi, dove ciascun impulso può poi esser utilizzato per dar inizio ad una serie di operazioni. Immaginiamo infatti di collegare l'uscita dell'ultimo FF di uno SR come quello mostrato in figura 1.45, con l'ingresso di Set del primo FF.

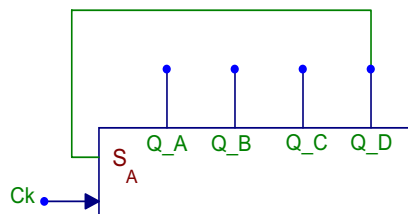


Figura 1.45:

Se nei FF A, B, C, D abbiamo inizialmente caricato i bit (0,0,0,1) rispettivamente, l'applicazione di successivi impulsi di clock farà "viaggiare" l'1 da Q_D in Q_A , poi da Q_A in Q_B , etc. Vediamo così che dopo 4 impulsi di clock l'1 è tornato in Q_D . Le linee Q_A , Q_B , Q_C , Q_D , assumeranno consecutivamente i valori mostrati nella tabella 1.39, dopo il numero di impulsi di clock indicato.

impulsi di Ck	Q_A	Q_B	Q_C	Q_D
0	0	0	0	1
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Tabella 1.39:

Con tale dispositivo possiamo anche abilitare, a comando (tramite gli impulsi di clock) una delle quattro linee A, B, C, D (ad esempio, far scoccare la scintilla in una delle candele del motore di un'automobile).

Ma è anche possibile, con tale sistema, generare una sequenza predeterminata di bits. Ad esempio, se nei quattro FF avessimo inizialmente memorizzato la sequenza 1001, dopo l'applicazione di successivi impulsi di clock avremmo le sequenze indicate nella tabella 1.40.

Notiamo ancora che, nel primo degli esempi visti, Q_D vale 1 ogni 4 impulsi di clock. Un tale sistema può quindi essere anche adoperato come "divisore di frequenza" (per 4 nell'esempio considerato, ma si può ottenere una divisione per potenze più elevate di 2, con registri a molti bit). Un tale tipo di utilizzo, dove l'uscita dell'ultimo FF è collegata all'ingresso del primo, va sotto il nome di "generatore di sequenze" o anche "contatore ad anello" (ring counter).

Se i FF di cui il registro è costituito sono di tipo RS o JK, è possibile collegare l'uscita dell'ultimo FF della catena con l'ingresso R (o K) del primo. Si ottiene

impulsi di Ck	Q_A	Q_B	Q_C	Q_D
0	1	0	0	1
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Tabella 1.40:

cosí quello che é noto come "Johnson Counter" o "Twisted-Ring Counter". Si può verificare che con tale sistema si può ottenere una divisione di frequenza per $2N$, dove N é la lunghezza del registro (anziché per N come in un normale Ring Counter).

1.15.2 Applicazioni dei FF

Contatori asincroni

I FF sono ampiamente usati per realizzare dei contatori. Consideriamo come primo esempio quello di figura 1.46 (noto come ripple counter) che utilizza quattro FF di tipo JK.

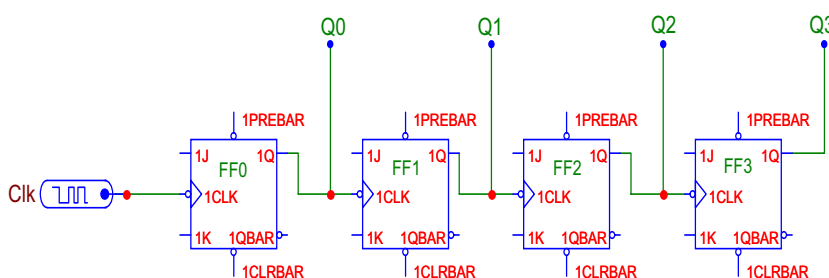


Figura 1.46:

Ammettiamo che tutti gli ingressi J e K, come pure quelli di Preset e Clear, siano fissi ad 1. In tal caso i quattro FF effettueranno la funzione TOGGLE, cioè cambieranno stato ad ogni impulso presente sull'ingresso di clock. Più precisamente, la transizione delle uscite avverrà sul fronte di discesa dell'impulso di clock. Notiamo ora che per il secondo FF fa da clock l'uscita del primo; per il terzo, l'uscita del secondo etc. Di conseguenza le forme d'onda sulle uscite $Q_0 - Q_3$ saranno quelle mostrate in figura 1.47.

Vediamo che la frequenza di Q_0 é la metà di quella del clock, quella di Q_1 un quarto etc. Inoltre notiamo che la sequenza dei bit (Q_3, Q_2, Q_1, Q_0) inizia con (0,0,0,0) poi (0,0,0,1), diviene poi (0,0,1,0), (0,0,1,1) etc, cioè il registro $Q_3 \cdots Q_0$ "conta" gli impulsi di clock in arrivo al primo FF. Ovviamente, dopo il 15^{mo} impulso di clock il contatore si "resetta", cioè le uscite saranno (0,0,0,0), per poi ricominciare a cre-

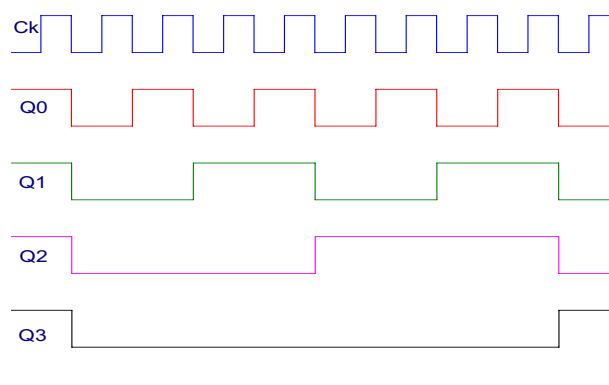


Figura 1.47:

scere. Se colleghiamo ciascuna delle uscite ad un LED potremo vedere il conteggio (binario) del numero di impulsi di clock ricevuto dal FF0.

Il contatore, con una piccola modifica, può contare "all'indietro". E' sufficiente a tale scopo che il clock di ciascun FF sia collegato al \overline{Q} (anziché al Q) del FF che lo precede. Ciò é facile da verificare.

E' possibile realizzare anche una catena di FF che, con un opportuno comando di controllo, conti in entrambi i versi. E' sufficiente collegare l'uscita di un FF al clock del successivo nel modo illustrato in figura 1.48.

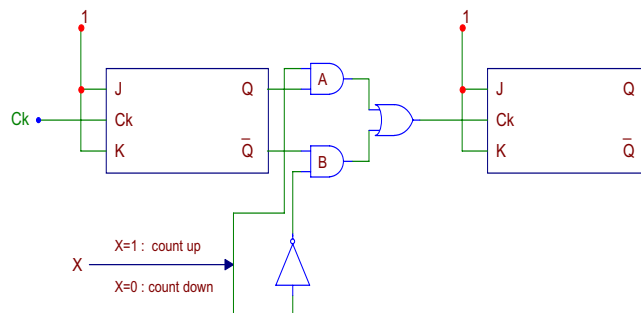


Figura 1.48:

Se l'ingresso X é alto, la porta AND B é disabilitata, mentre é abilitata la A. Il contatore esegue il ciclo in ordine crescente. Se invece l'ingresso X é basso, sarà abilitata (notare l'invertitore) la porta B e disabilitata la A. Il contatore eseguirá il ciclo in ordine decrescente.

E' possibile modificare i collegamenti nel circuito di figura 1.46, in modo da realizzare un contatore che, anziché resettarsi ogni 16 impulsi, si resettì ogni 10. Si ottiene in tal modo quello che é noto come un "contatore decimale" o Binary Coded Decimal (BCD).

La modifica é illustrata in figura 1.49 dove, come nel caso precedente, tutti i terminali J e K sono posti al livello logico alto. La presenza della porta AND, con ingressi collegati a Q1 e Q3 e con uscita collegata al Clear, fa sí che il contatore si resettì non appena Q1=Q3=1. Come si può vedere dalla figura 1.47, ciò accade dopo il decimo impulso di clock.

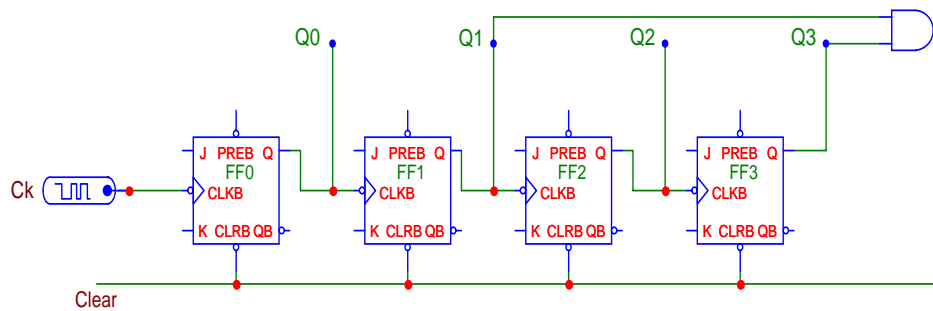


Figura 1.49:

Contatori sincroni

I contatori fin qui esaminati sono noti come "contatori asincroni". Ciò poiché la transizione del secondo FF della catena avviene soltanto dopo la transizione del primo (essendo l'ingresso di clock del secondo alimentato dall'uscita (Q o \overline{Q}) del primo, ed analogamente per i successivi). Ciò costituisce uno svantaggio, poiché il tempo di propagazione (propagation delay) di un FF è relativamente lungo ($\tau \approx 20 - 30 ns$ nel caso del '74). Se tutti i FF sono nello stato logico 1, tale tempo è massimo, poiché il successivo intervallo di clock manda il primo FF da 1 a 0, questo a sua volta manda il secondo da 1 a 0 (con un ritardo τ), il secondo manda il terzo da 1 a 0 (con uguale ritardo) e così via. Il ritardo complessivo sarà in tal caso $N\tau$, se N è il numero di FF della catena. Per ridurre il tempo di propagazione conviene adottare uno schema in cui tutti i FF effettuano la transizione simultaneamente. Tale tipo di contatore va sotto il nome di "contatore sincrono". Un esempio è mostrato in figura 1.50.

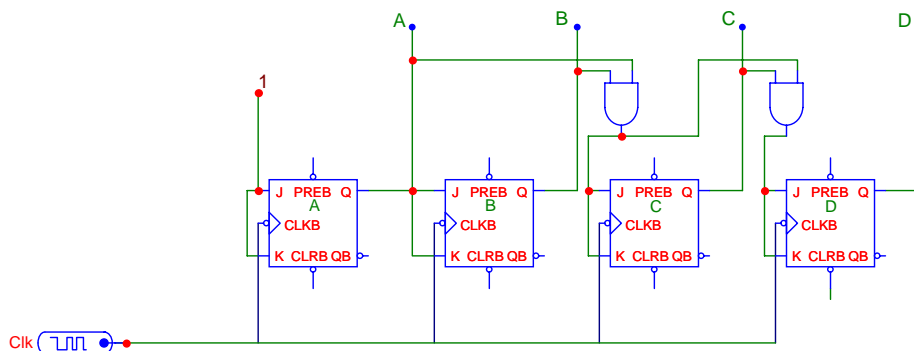


Figura 1.50:

Qui il primo FF (quello relativo alla cifra meno significativa) ha $J=K=1$, in modo da cambiare stato in corrispondenza ad ogni impulso di clock. Il secondo FF (B) ha $J=K=A$, in modo da commutare (in corrispondenza all'arrivo di un segnale di clock) solo quando $A=1$. Il terzo FF (C) commuta quando sia A che B sono

uguali ad 1. Infine D commuta se é verificata la condizione $A=B=C=1$. Come possiamo facilmente verificare (vedasi la tabella 1.41) la presenza delle porte AND soddisfa la logica insita nella successione dei numeri binari crescenti con $(D, C, B, A) = (0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), \dots$.

Clock	A	B	C	D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1
16	0	0	0	0

Tabella 1.41:

Infatti, se associamo il bit meno significativo al primo FF(A) e quello piú significativo all'ultimo (D), la tabella ci dice che:

- a** il primo FF deve commutare ad ogni impulso di clock
- b** il secondo FF deve commutare in corrispondenza ad un impulso di clock se l'uscita del primo (A) vale 1
- c** il terzo FF deve commutare in corrispondenza ad un impulso di clock se sia l'uscita del primo (A) che quella del secondo (B) valgono 1
- d** l'ultimo FF deve commutare in corrispondenza ad un impulso di clock se tutte e tre le uscite (A, B, C) valgono 1

I collegamenti sono quindi quelli necessari per un conteggio crescente. Vediamo che, dopo il quindicesimo impulso di clock il contatore si resetta automaticamente e ricomincia il conteggio.

Esaminiamo il tempo di propagazione in ciascuna delle transizioni indicate nella tabella, ignorando in un primo tempo, per semplicitá, i ritardi dovuti alle porte AND. Nel passaggio dalla prima alla seconda riga il tempo di propagazione é solo quello (τ) relativo alla commutazione del primo FF (gli altri non cambiano stato).

Analogamente, nella successiva transizione (passaggio dalla seconda alla terza riga della tabella) il tempo di commutazione é solo quello relativo al secondo FF. Nella terza transizione cambiano stato i primi tre FF (A,B,C). Tuttavia il ritardo é ancora τ , poiché i tre cambiamenti di stato avvengono contemporaneamente. Proseguendo l'analisi é facile vedere che il ritardo complessivo nel conteggio fino a 15 é pari a 15τ . Questo risultato é da confrontare con quello che si incontra nel caso di un contatore asincrono, in cui il ritardo é 15τ nella sola transizione $(1\ 1\ 1\ 1 \rightarrow 0\ 0\ 0\ 0)$.

In realtà, nel caso del contatore sincrono occorre tener conto del ritardo aggiuntivo dovuto alle porte AND, che é dell'ordine di (10-20 ns) per ciascun FF che alimenta una porta AND.

Un contatore sincrono binario a 4 bit, di uso abbastanza generale é il 74LS191. Esso può eseguire il conteggio in avanti o indietro a seconda del valore logico presente sull'ingresso di controllo "up/down". Il 74LS190, la cui struttura interna é mostrata in figura 1.51 e le cui caratteristiche sono mostrate in figura 1.52, é praticamente equivalente al 74LS191, con la sola differenza che esso é predisposto per il conteggio decimale (BCD).

74LS90 Decade Counter

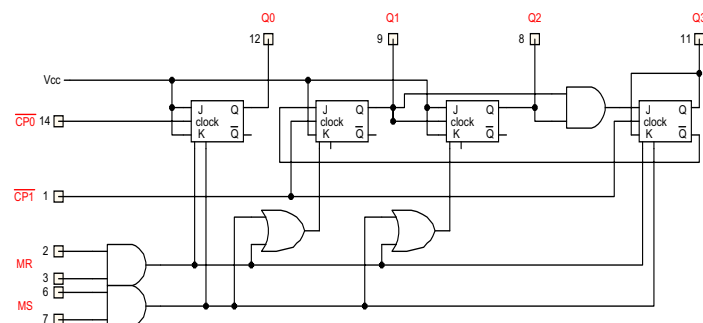


Figura 1.51:

E' possibile, collegando due o più contatori in serie, realizzare conteggi fino a potenze più elevate di 2 (o di 10). Ad esempio, collegando in serie tre contatori BCD 74LS190 é possibile, come mostrato in figura 1.53, realizzare un contatore modulo 1000.

Vediamo che l'ingresso DN/UP (Down/Up) é collegato a massa, il che implica un conteggio crescente (massa=logico 0). L'ingresso di clock di tutti e tre i contatori riceve il medesimo segnale d'ingresso (cioé gli impulsi da contare). Il 74LS190 é costruito in modo tale che sul piedino con la sigla RC (Ripple Clock Output) compaia un impulso "basso" (cioé uno 0) di durata uguale alla porzione "bassa" del clock d'ingresso, quando si verifichi un "overflow", cioé quando i quattro FF abbiano superato il massimo conteggio consentito (9 nel caso del 74LS190). Tale livello basso, collegato all'Enable (E) del contatore successivo, fa sí che questo cominci a contare. Così, quando il primo contatore (da sinistra) sia passato dal 9 allo 0, il

ripartirà da 0, arrivando fino a 9 e poi a 0. A questo punto il primo contatore genera un nuovo RC ed invia quindi un'Enable al secondo, che ora passerà a 2, e così di seguito. Quando anche il secondo contatore sia arrivato a 9 e poi a 0, esso genererà un RC e quindi invierà un'Enable al terzo, che inizierà a contare la centinaia.

1.16 Alcune applicazioni pratiche di registri e contatori in esperimenti di fisica

Abbiamo già visto come una catena di FF possa essere utilizzata come divisore di frequenza. Un contatore può essere utilizzato, ovviamente, per contare. Ad esempio, se in un processo industriale si vuol contare il numero di oggetti che sono passati su di un nastro trasportatore in un certo tempo sarà sufficiente far uso di un dispositivo (un diodo emettitore di luce (LED) ed una cellula fotoelettrica) per generare un impulso ogni volta che la luce emessa dal LED è interrotta dal passaggio di uno degli oggetti e non arriva quindi alla cellula fotoelettrica. Un contatore opportuno incrementerà di un'unità il conteggio ogni volta che gli arrivi un impulso. Un contatore separato conterà il numero di impulsi che gli pervengono, generati da un opportuno oscillatore (clock) previamente calibrato, che fornirà l'informazione temporale. Possiamo utilizzare un dispositivo simile per misurare l'intervallo di tempo tra due eventi. La figura 1.54 illustra un sistema di questo tipo.

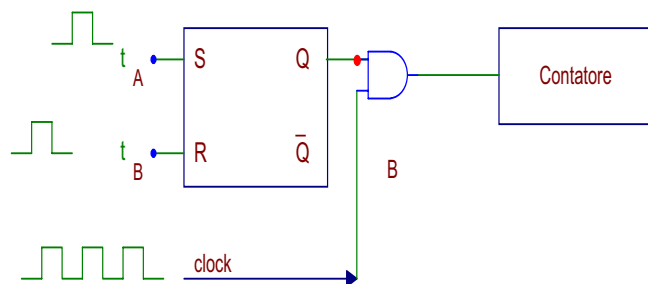


Figura 1.54:

Gli "eventi" in questo caso sono costituiti dal passaggio di un oggetto attraverso due traguardi A e B (come nel caso del nastro trasportatore). Un primo impulso viene generato al tempo t_A dal passaggio dell'oggetto davanti al traguardo A ed un secondo al tempo t_B in cui l'oggetto passa davanti al traguardo B.

L'impulso t_A sia inviato all'ingresso di Set di un FF di tipo RS (in cui era inizialmente $Q = 0$, $\bar{Q} = 1$) mentre l'impulso generato all'istante t_B sia inviato all'ingresso di Reset. Il gate AND che segue è collegato a valle ad un contatore (opportunamente calibrato), mentre dei due ingressi, uno è collegato al Q del FF e l'altro riceve una sequenza di impulsi di clock, di frequenza nota.

Il funzionamento del circuito è il seguente: quando l'impulso generato al tempo t_A arriva al FF, questo avrà $S=1$, $R=0$ e quindi $Q=1$. Subito dopo sarà $S=R=0$ e quindi lo stato del FF rimane invariato fino al tempo t_B , quando R diventa 1 e quindi Q va a 0. Nell'intervallo di tempo $t_B - t_A$ in cui $Q=1$, la porta AND è abilitata e

quindi essa lascia passare gli impulsi di clock in arrivo. Il contatore che segue conta tali impulsi di clock, il cui numero é proporzionale all'intervallo di tempo $t_B - t_A$.

Il sistema andrà ovviamente calibrato preventivamente. Un dubbio che può venire é il seguente. Sia l'impulso generato al tempo t_A che quello generato al tempo t_B sono prodotti dal medesimo dispositivo (cellula fotoelettrica seguita da monostabile). Come possiamo far sí che dei due segnali generati su di una stessa linea, uno vada all'ingresso S e l'altro all'ingresso R? Ciò é semplice se sostituiamo il FF RS con un JK in cui $J=K=1$, come mostrato in figura 1.55.

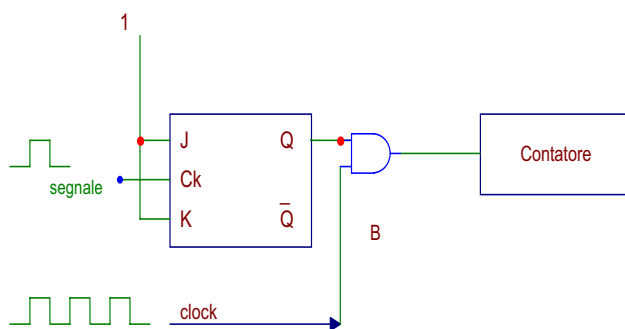


Figura 1.55:

Se inizialmente era $Q=0$, l'arrivo del segnale generato al tempo t_A manda Q in 1, mentre il successivo arrivo del segnale generato al tempo t_B manda Q in 0 e \bar{Q} in 1. Un dispositivo molto simile a questo può essere adoperato se si vuole misurare una frequenza, cioè il numero di impulsi/sec. Tale dispositivo é mostrato in figura 1.56.

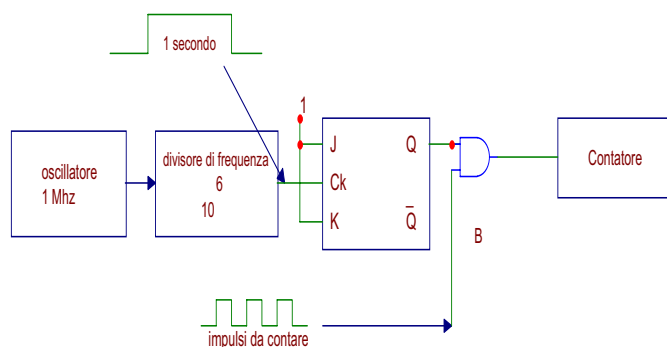


Figura 1.56:

Gli impulsi in arrivo (di forma generica, ad esempio sinusoidale) vengono trasformati in impulsi rettangolari ed inviati ad uno degli ingressi di una porta AND. L'altro ingresso della porta é collegato all'uscita Q di un FF JK con $J=K=1$, che riceve sull'ingresso di clock un segnale ottenuto demoltiplicando per un fattore 10^6 un segnale di 1 Mhz. In tale circostanza Q é uguale ad 1 per un tempo di 1s. La porta AND sarà quindi abilitata per 1 s ed il contatore conterà il numero di impulsi che in tale tempo gli arrivano, cioè la frequenza voluta. Naturalmente occorrerà aggiungere qualcosa al circuito per far sí che alla fine dell'intervallo di tempo (di 1 s), il sistema si arresti e consenta la lettura del contatore.

Anche se non specificamente relativo ad applicazioni in esperimenti di fisica, incontrano ampio utilizzo i contatori decimali. Infatti, in un contatore BCD (Binary Coded Decimal) un numero é espresso come una sequenza di cifre decimali. Ad esempio, il numero 3702 é memorizzato come mostrato nella tabella 1.42.

0011	0111	0000	0010
3	7	0	1

Tabella 1.42:

I dati provenienti dalla tastiera di un computer sono normalmente espressi in BCD, anche se si tratta di caratteri. Infatti, a ciascuno di questi é possibile associare, mediante un codice (ad esempio il codice ASCII - American Standard Code for Information Exchange), in maniera del tutto analoga ai numeri, una stringa di bit. Il codice ASCII é a 7 bit. I principali caratteri ed i codici corrispondenti sono mostrati nella tabella di figura 1.57. L'integrato 74LS93, che consente il conteggio fino a 16, può essere azzerato, come si é visto, in corrispondenza del numero 9. E' però piú conveniente usare un contatore predisposto per contare in base 10, come il 74LS90.

Nella tabella di figura 1.57, la prima colonna contiene il numero in decimale, la seconda il corrispondente valore esadecimale, la terza il corrispondente simbolo.

Un'alternativa spesso adoperata é quella di un contatore con uscita esadecimale (base 16). Come visto, in tale rappresentazione, un numero e.g. di 2 bytes (16 bit) é rappresentato da una stringa di 4 caratteri esadecimali. Ad esempio, il numero binario: 1010 1101 0110 0011 é rappresentato in esadecimale da: *AD63*.

1.17 Decodificatori e Display

Per visualizzare un numero binario abbiamo varie alternative. La prima é quella, ovvia, di visualizzare ogni singolo bit mediante un LED. Se il LED é acceso il bit corrispondente vale 1, se esso é spento vale 0. Tale soluzione non é pratica per la visualizzazione di numeri con molti bit. Essa inoltre non offre una lettura immediata del numero, piú comoda se fatta in decimale o in esadecimale.

Utilizzando una decodifica BCD, dove occorre decodificare soltanto i numeri da 0 a 9 (in binario da (0,0,0,0) a (1,0,0,1)) si può far uso del chip 74LS42, che é un integrato a 16 pin. Il display può essere effettuato utilizzando dei "display a 7 segmenti", in cui a ciascun numero BCD (tra 0 e 9) corrisponde l'accensione degli opportuni segmenti luminosi (LED), come mostrato in figura 1.58.

L'identificazione dei segmenti é quella mostrata in corrispondenza dell'8, in basso nella figura. Il decodificatore BCD a sette segmenti 74LS48, mostrato in figura 1.59, é un decoder/driver adatto a comandare i segmenti di un LED. Esso contiene sia la circuiteria che fornisce la decodifica dei quattro bit d'ingresso nelle 7 linee d'uscita, sia quella di alimentazione dei singoli segmenti.

L'indicatore può essere spento portando a 0 l'ingresso di blanking (\overline{BI} : pin 4).

La tabella delle verità di tale circuito é mostrata, insieme al display riassuntivo e ad altre caratteristiche, in figura 1.60

La decodifica per gruppi di 4 dal binario all'esadecimale può essere effettuata utilizzando l'integrato 74LS154, le cui caratteristiche sono mostrate in figura 1.61.

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
0 0 NUL	16 10 DLE	32 20 (space)	48 30 0
1 1 SOH	17 11 DC1	33 21 !	49 31 1
2 2 STX	18 12 DC2	34 22 "	50 32 2
3 3 ETX	19 13 DC3	35 23 #	51 33 3
4 4 EOT	20 14 DC4	36 24 \$	52 34 4
5 5 ENQ	21 15 NAK	37 25 %	53 35 5
6 6 ACK	22 16 SYN	38 26 &	54 36 6
7 7 BEL	23 17 ETB	39 27 '	55 37 7
8 8 BS	24 18 CAN	40 28 (56 38 8
9 9 TAB	25 19 EM	41 29)	57 39 9
10 A LF	26 1A SUB	42 2A *	58 3A :
11 B VT	27 1B ESC	43 2B +	59 3B ;
12 C FF	28 1C FS	44 2C ,	60 3C <
13 D CR	29 1D GS	45 2D -	61 3D =
14 E SO	30 1E RS	46 2E .	62 3E >
15 F SI	31 1F US	47 2F /	63 3F ?

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
64 40 @	80 50 P	96 60 `	112 70 p
65 41 A	81 51 Q	97 61 a	113 71 q
66 42 B	82 52 R	98 62 b	114 72 r
67 43 C	83 53 S	99 63 c	115 73 s
68 44 D	84 54 T	100 64 d	116 74 t
69 45 E	85 55 U	101 65 e	117 75 u
70 46 F	86 56 V	102 66 f	118 76 v
71 47 G	87 57 W	103 67 g	119 77 w
72 48 H	88 58 X	104 68 h	120 78 x
73 49 I	89 59 Y	105 69 i	121 79 y
74 4A J	90 5A Z	106 6A j	122 7A z
75 4B K	91 5B [107 6B k	123 7B {
76 4C L	92 5C \	108 6C l	124 7C
77 4D M	93 5D]	109 6D m	125 7D }
78 4E N	94 5E ^	110 6E n	126 7E ~
79 4F O	95 5F _	111 6F o	127 7F

Figura 1.57:

Questo ha 16 uscite, corrispondenti ciascuna ad uno dei 16 simboli da 0 a 15 (F esadecimale). Ciascuna delle 16 uscite può essere collegata ad un opportuno LED, che si accende quando l'uscita corrispondente assuma il livello basso.

Un circuito che effettua sullo stesso chip sia la decodifica che il display esadecimale è il TIL311. Questo riceve sui piedini (3,2,13,12) il carattere esadecimale (con il 3 il bit meno significativo, ed effettua un display esadecimale del relativo carattere (cioè da 0 ad F). Questo display è quello adoperato in alcune delle esperienze di laboratorio. Le caratteristiche di tale integrato sono mostrate nelle figure 1.62 e 1.63.

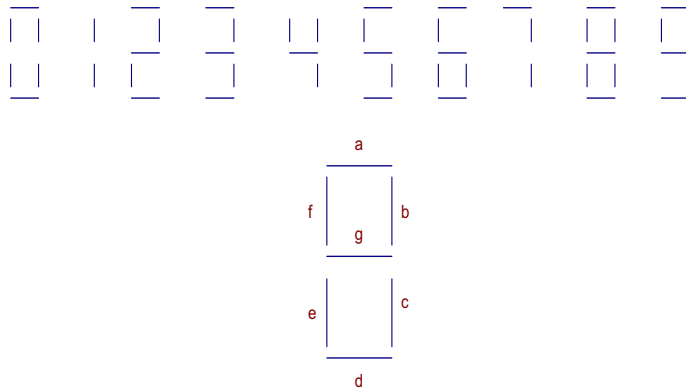


Figura 1.58:

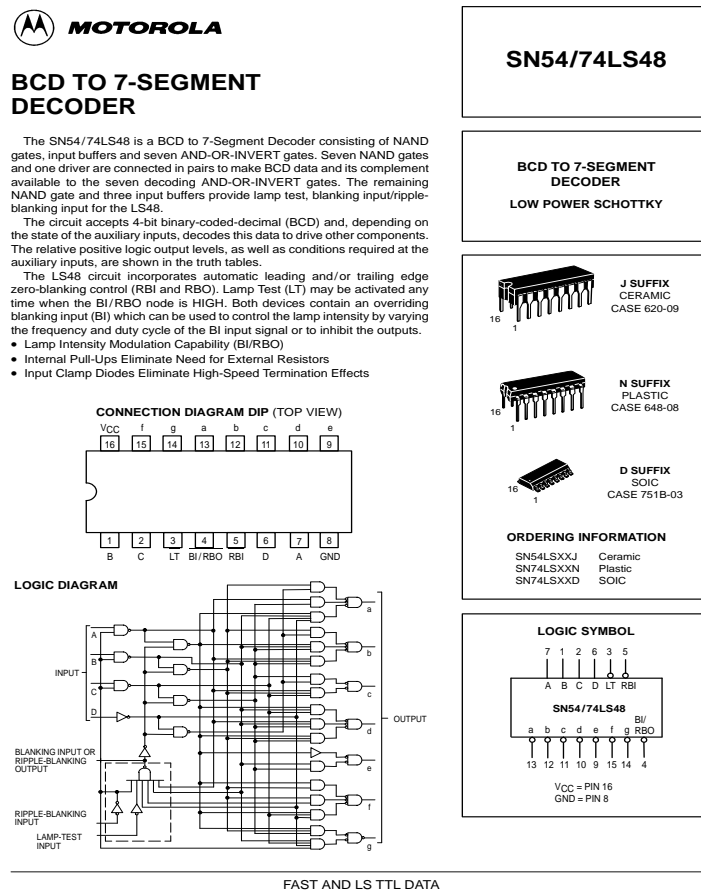
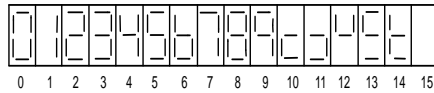


Figura 1.59:

La figura 1.64 mostra uno schema a blocchi del TIL311.

Notiamo la presenza di un "Latch" in ingresso. Questo é essenzialmente un insieme di 4 FF di tipo D. Esso ha lo scopo di "mantenere" i dati presentati in ingresso, per il tempo necessario alla decodifica ed al successivo display a lettura. Più precisamente, finché il segnale presente sul piedino 5 é basso, le uscite del Latch seguono l'ingresso (piedini 3,2,13,12). Quando il segnale é alto, le uscite non



NUMERICAL DESIGNATIONS — RESULTANT DISPLAYS

TRUTH TABLE
SN54/74LS48

DECIMAL OR FUNCTION	INPUTS						OUTPUTS								NOTE
	LT	RBI	D	C	B	A	BI/RBO	a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	H	H	H	H	H	L	1	
1	H	X	L	L	L	H	H	L	H	H	L	L	L	1	
2	H	X	L	L	H	L	H	H	H	L	H	H	L		
3	H	X	L	L	H	H	H	H	H	H	L	L	L		
4	H	X	L	H	L	L	H	L	H	H	L	L	H		
5	H	X	L	H	L	H	H	H	L	H	H	L	H		
6	H	X	L	H	H	L	H	L	L	H	H	H	H		
7	H	X	L	H	H	H	H	H	H	H	L	L	L		
8	H	X	H	L	L	L	H	H	H	H	H	H	H		
9	H	X	H	L	L	H	H	H	H	L	L	L	H		
10	H	X	H	L	H	L	H	L	L	L	H	H	L		
11	H	X	H	L	H	H	H	L	L	H	H	L	L		
12	H	X	H	H	L	L	H	L	H	L	L	L	H		
13	H	X	H	H	L	H	H	H	L	L	H	L	H		
14	H	X	H	H	H	L	H	L	L	L	H	H	H		
15	H	X	H	H	H	H	H	L	L	L	L	L	L		
BI	X	X	X	X	X	X	L	L	L	L	L	L	L	2	
RBI	H	L	L	L	L	L	L	L	L	L	L	L	L	3	
LT	L	X	X	X	X	X	H	H	H	H	H	H	H	4	

NOTES:

- (1) BI/RBO is wired-AND logic serving as blanking input (BI) and/or ripple-blanking output (RBO). The blanking out (BI) must be open or held at a HIGH level when output functions 0 through 15 are desired, and ripple-blanking input (RBI) must be open or at a HIGH level if blanking of a decimal 0 is not desired. X=input may be HIGH or LOW.
- (2) When a LOW level is applied to the blanking input (forced condition) all segment outputs go to a LOW level, regardless of the state of any other input condition.
- (3) When ripple-blanking input (RBI) and inputs A, B, C, and D are at LOW level, with the lamp test input at HIGH level, all segment outputs go to a HIGH level and the ripple-blanking output (RBO) goes to a LOW level (response condition).
- (4) When the blanking input/ripple-blanking output (BI/RBO) is open or held at a HIGH level, and a LOW level is applied to lamp-test input, all segment outputs go to a LOW level.

Figura 1.60:

cambiano più. Si noti la presenza di due terminali d'ingresso (4 e 10) per l'accensione del punto decimale a destra ed a sinistra rispettivamente. Questi LED sono accesi attraverso gli invertitori indicati in figura.

1.18 Moltiplicazione e divisione in binario

L'operazione di moltiplicazione in binario tra due bit equivale all'AND di essi (tabella 1.43).

0	×	0	=	0
1	×	0	=	0
0	×	1	=	0
1	×	1	=	1

Tabella 1.43:

La moltiplicazione tra numeri di più bit può essere effettuata con una semplice

DM74LS154 4-Line to 16-Line Decoder/Demultiplexer

General Description

Each of these 4-line-to-16-line decoders utilizes TTL circuitry to decode four binary-coded inputs into one of sixteen mutually exclusive outputs when both the strobe inputs, G1 and G2, are LOW. The demultiplexing function is performed by using the 4 input lines to address the output line, passing data from one of the strobe inputs with the other strobe input LOW. When either strobe input is HIGH, all outputs are HIGH. These demultiplexers are ideally suited for implementing high-performance memory decoders. All inputs are buffered and input clamping diodes are provided to minimize transmission-line effects and thereby simplify system design.

Features

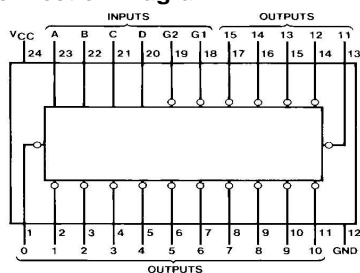
- Decodes 4 binary-coded inputs into one of 16 mutually exclusive outputs
- Performs the demultiplexing function by distributing data from one input line to any one of 16 outputs
- Input clamping diodes simplify system design
- High fan-out, low-impedance, totem-pole outputs
- Typical propagation delay
 - 3 levels of logic 23 ns
 - Strobe 19 ns
- Typical power dissipation 45 mW

Ordering Code:

Order Number	Package Number	Package Description
DM74LS154WM	M24B	24-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300 Wide
DM74LS154N	N24A	24-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-010, 0.600 Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagram



Logic Diagram

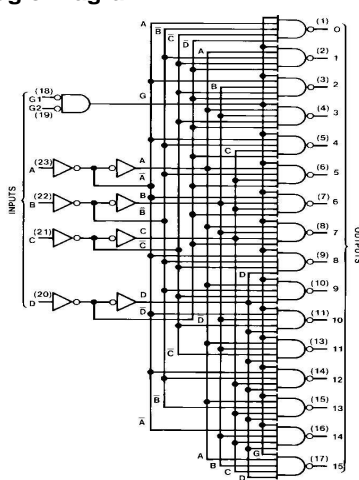


Figura 1.61:

estrapolazione dell'analogo processo nel sistema decimale. Mostriamo alcuni esempi (tabelle 1.44 e 1.45).

1	0	1	1	=	11	×
0	1	0	1	=	5	
1	0	1	1			
0	0	0	0			
1	0	1	1			
0	0	0	0			
1	1	0	1	1	1	= 55

Tabella 1.44:

Vediamo che ciascun bit del secondo numero (a partire dalla destra) deve essere

LATCH STROBE INPUT (PIN 5) - When low, the data in the latches follow the data on the latch data inputs. When high, the data in the latches will not change.

BLANKING INPUT (PIN 8) - When high, the display is blanked regardless of the levels of the other inputs. When low, a character is displayed as determined by the data in the latches.

DECIMAL POINT CATHODES are not connected to the logic chip. External current limiting resistors must be used.

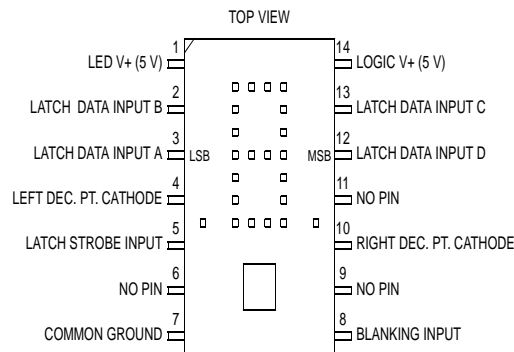


Figura 1.62:

TIL311 HEXADECIMAL DISPLAY WITH LOGIC

SODS001D - D1176, MARCH 1972 - REVISED FEBRUARY 1992

description

This hexadecimal display contains a four-bit latch, decoder, driver, and 4×7 light-emitting-diode (LED) character with two externally-driven decimal points in a 14-pin package. A description of the functions of the inputs of this device follows.

FUNCTION	PIN NO.	DESCRIPTION
LATCH STROBE INPUT	5	When low, the data in the latches follow the data on the latch data inputs. When high, the data in the latches will not change. If the display is blanked and then restored while the enable input is high, the previous character will again be displayed.
BLANKING INPUT	8	When high, the display is blanked regardless of the levels of the other inputs. When low, a character is displayed as determined by the data in the latches. The blanking input may be pulsed for intensity modulation.
LATCH DATA INPUTS (A, B, C, D)	3, 2, 13, 12	Data on these inputs are entered into the latches when the enable input is low. The binary weights of these inputs are A = 1, B = 2, C = 4, D = 8.
DECIMAL POINT CATHODES	4, 10	These LEDs are not connected to the logic chip. If a decimal point is used, an external resistor or other current-limiting mechanism must be connected in series with it.
LED SUPPLY	1	This connection permits the user to save on regulated V_{CC} current by using a separate LED supply, or it may be externally connected to the logic supply (V_{CC}).
LOGIC SUPPLY (V_{CC})	14	Separate V_{CC} connection for the logic chip
COMMON GROUND	7	This is the negative terminal for all logic and LED currents except for the decimal points.

The LED driver outputs are designed to maintain a relatively constant on-level current of approximately 5 mA through each LED that forms the hexadecimal character. This current is virtually independent of the LED supply voltage within the recommended operating conditions. Drive current varies slightly with changes in logic supply voltage resulting in a change in luminous intensity as shown in Figure 2. This change will not be noticeable to the eye. The decimal point anodes are connected to the LED supply; the cathodes are connected to external pins. Since there is no current limiting built into the decimal point circuits, this must be provided externally if the decimal points are used.

The resultant displays for the values of the binary data in the latches are as shown below.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Figura 1.63:

moltiplicato per i bit del primo. A tali "moltiplicazioni" potranno provvedere delle porte AND. Il risultato ottenuto dalla moltiplicazione del primo bit andrà sommato al risultato della moltiplicazione del secondo, shiftato a sinistra di una posizione; questo andrà sommato al risultato della moltiplicazione del terzo bit, shiftato a sini-

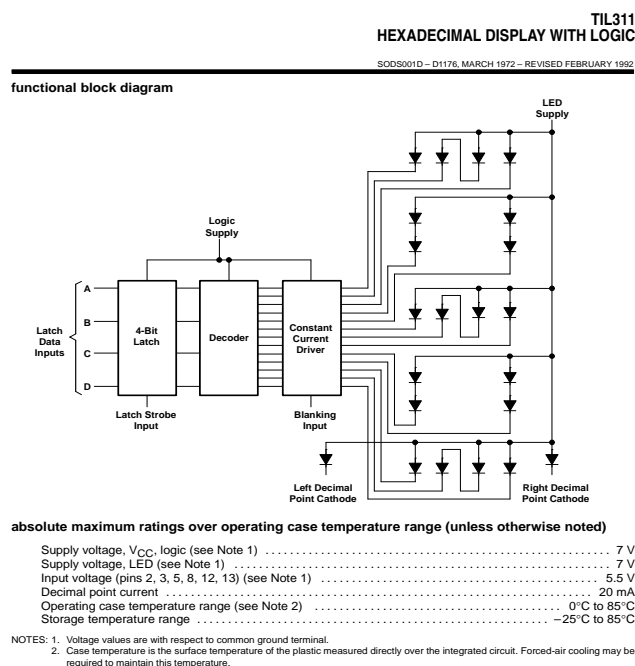


Figura 1.64:

	1	1	0	1	=	13	×
	1	0	0	1	=	9	
	1	1	0	1			
	0	0	0	0			
	0	0	0	0			
1	1	0	1				
1	1	1	0	1	0	1	= 117

Tabella 1.45:

stra di due posizioni ... etc. Vediamo così che un "moltiplicatore" può esser ottenuto effettuando sequenzialmente degli AND, degli shift e delle somme. Il circuito mostrato in figura 1.65 illustra una possibile implementazione di un moltiplicatore di due numeri a 3 bit.

Per il risultato della moltiplicazione è ovviamente necessario un registro a 6 bit. Il circuito comprende uno SR a 5 bit per il moltiplicando, uno SR a 3 bit per il moltiplicatore ed un accumulatore a 6 bit, oltre ad un sommatore a 6 bit. Inizialmente si azzerava il registro accumulatore, si caricano i tre bit del moltiplicatore nel relativo registro (con Q_0 corrispondente al bit meno significativo) ed i tre bit del moltiplicando nei registri $Q_0 - Q_2$ corrispondenti. Le uscite delle cinque porte NAND contengono a questo punto i prodotti del bit meno significativo del moltiplicatore (Q_0) per i tre bit del moltiplicando. Più in particolare, la porta indicata con e fornisce il prodotto dei due bit meno significativi dei due numeri, la porta d fornisce il prodotto del bit Q_0 del moltiplicatore per il bit Q_1 del moltiplicando, la porta c fornisce il prodotto del bit Q_0 del moltiplicatore per il bit Q_2 del moltiplicando.

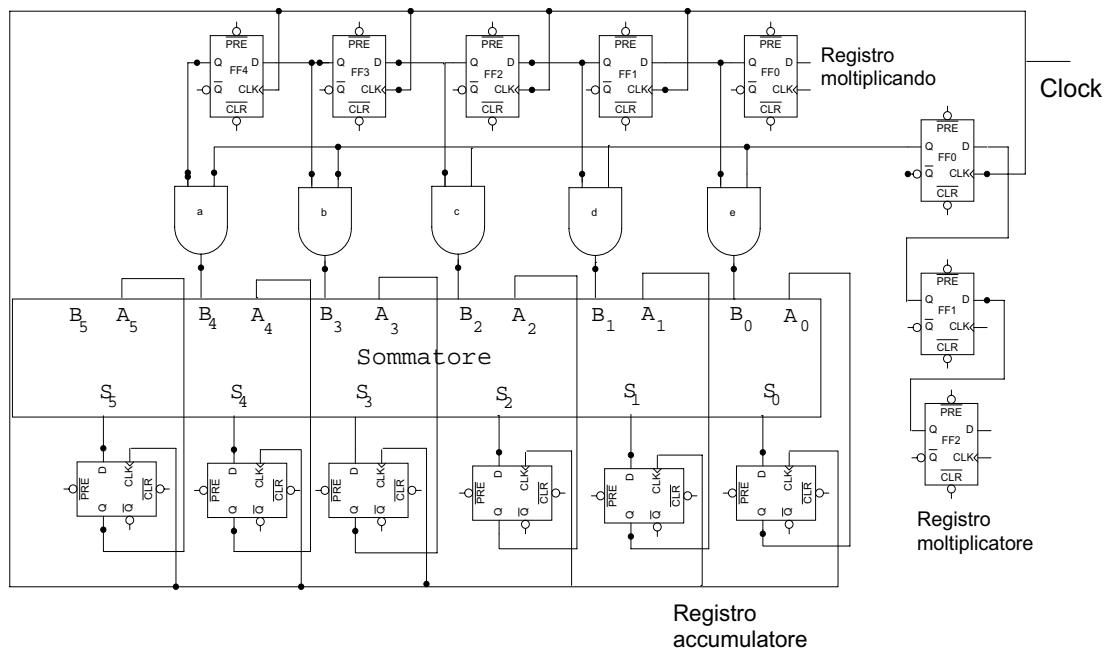


Figura 1.65:

cando. Le porte a e b sono per il momento al livello logico 0. I prodotti parziali così ottenuti sono ora disponibili agli ingressi del sommatore, in B_0, B_1, B_2 . Essi sono anche disponibili all'uscita del sommatore (S_0, S_1, S_2) essendo gli altri ingressi (A_0, A_1, A_2) per il momento nulli. Al primo impulso di clock, il prodotto parziale (S_0, S_1, S_2) viene trasferito nei tre FF all'estrema destra del registro accumulatore e contemporaneamente agli ingressi (A_0, A_1, A_2) del sommatore. Il medesimo impulso di clock agisce sul registro moltiplicatore e sul registro moltiplicando nel modo seguente:

1. I bit del registro moltiplicatore vengono spostati in alto di una posizione, con che Q_1 viene trasferito in FF0
2. i tre bit del moltiplicando vengono spostati a sinistra di una posizione.

Notiamo che ora il registro all'estrema destra del moltiplicando è a livello logico 0. E' facile vedere che l'uscita della porta AND indicata con e è a livello logico 0, mentre le uscite dalle porte (d, c, b) contengono i prodotti del secondo bit del moltiplicatore per i tre bit del moltiplicando. Notiamo anche che con tale metodo, i tre bit ottenuti sono presenti agli ingressi (B_1, B_2, B_3) del sommatore e quindi risultano già shiftati a sinistra di una posizione, che è proprio quel che si fa nella moltiplicazione binaria. Ora all'uscita del sommatore comparirà, in (S_0, S_1, S_2, S_3, S_4) la somma del risultato della prima moltiplicazione e di quello della seconda (shiftata). Il successivo impulso di clock trasferisce tali uscite nei corrispondenti FF dell'accumulatore e quindi agli ingressi (A_0, \dots, A_4) del sommatore.

Il medesimo impulso di clock causa uno shift del bit più significativo del registro moltiplicatore (che era a questo punto in FF1) nel registro FF0 e simultaneamente sposta di una posizione a sinistra i bit del registro moltiplicando. Notiamo che ora i registri FF0 ed FF1 del moltiplicando saranno a livello logico 0. Le uscite

delle porte AND (c, b, a) forniscono ora i prodotti del bit piú significativo del moltiplicatore per i tre bit del moltiplicando. Tali prodotti sono ora disponibili agli ingressi (B_2, B_3, B_4) del sommatore. Questo fornirá ora alle uscite ($S_0 - S_5$) la somma finale richiesta. Sarà sufficiente un ulteriore impulso di clock per trasferirla nell'accumulatore e simultaneamente azzerare il registro moltiplicatore.

Il processo, effettuato come si é visto in modo seriale, é intrinsecamente lento. Esistono però in commercio dei moltiplicatori che effettuano le operazioni in parallelo, con un notevole guadagno in velocità.

1.19 La divisione in binario

L'idea base di un circuito digitale che effettua l'operazione di divisione tra due numeri interi (positivi) può essere illustrata con un semplice esempio, relativo al caso di numeri decimali. Ammettiamo infatti di voler dividere 33 per 7. Possiamo sottrarre il 7 dal 33 e verificare che la differenza (26) sia maggiore di 7. In tal caso sottraiamo da tale differenza ancora una volta il 7, ottenendo:

$$26 - 7 = 19$$

Procedendo ancora troviamo:

$$19 - 7 = 12$$

$$12 - 7 = 5$$

Poiché l'ultima differenza trovata é minore di 7, dobbiamo arrestare il processo e potremo dire che la parte intera del numero che esprime il rapporto é pari al numero delle sottrazioni effettuato, cioè 4. Dobbiamo ora calcolare la parte frazionaria del rapporto. A tale scopo sottraiamo ora 0.7 dall'ultima differenza trovata (5), tante volte quante ne occorrono per ottenere un resto inferiore a 0.7 (tabella 1.46).

5	-	0.7	=	4.3	(1)
4.3	-	0.7	=	3.6	(2)
3.6	-	0.7	=	2.9	(3)
2.9	-	0.7	=	2.2	(4)
2.2	-	0.7	=	1.5	(5)
1.5	-	0.7	=	0.8	(6)
0.8	-	0.7	=	0.1	(7)

Tabella 1.46:

Poiché il numero di sottrazioni effettuate é 7, concludiamo che 7 é la prima cifra decimale. Il rapporto desiderato, con la precisione di una cifra decimale é quindi 4.7. Per ottenere la successiva cifra possiamo procedere in modo analogo: sottraiamo 0.07 da 0.1 tante volte quante ne occorrono per trovare un resto inferiore a 0.07.

Troviamo cosí che la successiva cifra decimale é un 1, ed il rapporto é, a questo livello di precisione, 4.71. Il processo può essere ripetuto fino al livello di precisione voluto.

Ci proponiamo ora di vedere come il conteggio necessario per ottenere *la parte intera* del rapporto voluto possa essere implementato in logica binaria. Ammettiamo, per esemplificare, che divisore N e dividendo D abbiano 4 bit, e che il dividendo sia maggiore del divisore. Ad esempio sia $D=14_D = 1110_B$ ed $N=4_D = 0100_B$. La prima sottrazione, effettuata con il metodo del complemento a 2, fornisce il risultato mostrato nella tabella 1.47.

$$\begin{array}{rcccccl}
 & 1 & 1 & 1 & 0 & + \\
 & 1 & 1 & 0 & 0 & = \\
 \hline
 carry \rightarrow & 1 & 1 & 0 & 1 & 0
 \end{array}$$

Tabella 1.47:

Abbiamo cioè un bit di riporto (o di overflow), il quale indica che la differenza (uguale a $1010_B = 10_D$) è positiva. Sottraendo ancora abbiamo il risultato mostrato nella tabella 1.48.

$$\begin{array}{rcccccl}
 & 1 & 0 & 1 & 0 & + \\
 & 1 & 1 & 0 & 0 & = \\
 \hline
 carry \rightarrow & 1 & 0 & 1 & 1 & 0
 \end{array}$$

Tabella 1.48:

Abbiamo ancora una differenza positiva (come evidenziato dalla presenza del bit di riporto) uguale a 6_D . Un'ulteriore sottrazione fornisce quanto si può vedere nella tabella 1.49.

$$\begin{array}{rcccccl}
 & 0 & 1 & 1 & 0 & + \\
 & 1 & 1 & 0 & 0 & = \\
 \hline
 carry \rightarrow & 1 & 0 & 0 & 1 & 0
 \end{array}$$

Tabella 1.49:

cioè un numero positivo (2_D). Successivamente si ottiene quanto mostrato nella tabella 1.50.

$$\begin{array}{rcccccl}
 & 0 & 0 & 1 & 0 & + \\
 & 1 & 1 & 0 & 0 & = \\
 \hline
 carry \rightarrow & 0 & 1 & 1 & 1 & 0
 \end{array}$$

Tabella 1.50:

Vediamo che ora il bit di riporto è 0, cioè la differenza è negativa. La parte intera del rapporto desiderato è pari al numero di volte che la differenza ha fornito un bit di riporto uguale ad 1, cioè 3_D . Un circuito che realizza la funzione desiderata è quello mostrato in figura 1.66.

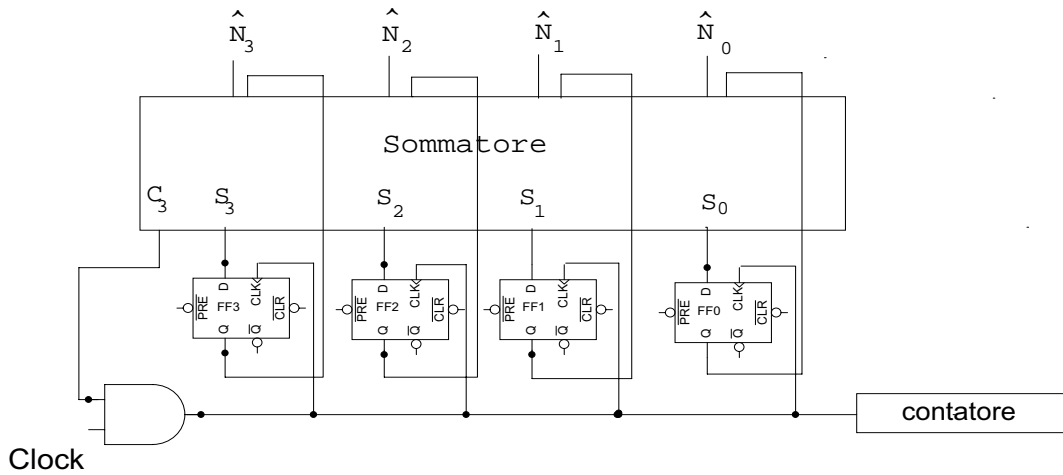


Figura 1.66:

Il dividendo é caricato in un registro, costituito da quattro FF di tipo D. Il sommatore indicato effettua la somma del contenuto di questo registro e del complemento a 2 del divisore, indicato in figura con $\hat{N}_3, \hat{N}_2, \hat{N}_1, \hat{N}_0$. Il dividendo é caricato inizialmente nel registro. Dopo che la prima somma sarà stata effettuata, il riporto $C_3 = 1$, mentre i registri Q_3, Q_2, Q_1, Q_0 conterranno il risultato della differenza tra dividendo e divisore. Poiché $C_3 = 1$, il prossimo impulso di clock farà avanzare il contatore di una unità. Ora il sommatore effettuerà la differenza tra il nuovo contenuto del registro ed il divisore. Se C_3 é ancora uguale ad 1, la porta AND continuerà ad essere abilitata ed il contatore avanzerà di una unità al prossimo impulso di clock. Il processo si ripete fino a quando C_3 non sia divenuto uguale a 0, al qual punto il contatore si arresta.