Lezioni di Elettronica

V. Flaminio

Dicembre 2001

Capitolo 1

Sistemi Logici

1.1 Sistemi di numerazione

Nella vita quotidiana siamo abituati ad usare il sistema decimale. In tale sistema la "base" é 10, ció 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 ció 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, ció 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

Scriveremo:

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

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 puó é esprimibile come:

9×10^0	$+$
5×10^1	$+$
1×10^2	$+$
4×10^3	$+$
1×10^4	$+$
3×10^5	$=$

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:

0×2^0	$+$
1×2^1	$+$
0×2^2	$+$
1×2^3	$+$
1×2^4	$+$
1×2^5	$+$
0×2^6	$+$
0×2^7	$+$
1×2^8	$=$

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

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:

7×8	$=$	56
$56 + 2$	$=$	58

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

Nel sistema esadecimale si dispone di 16 simboli (i numeri decimali da 0 a 9, e le lettere A, B, C, D, E, F). Abbiamo:

A	$\rightarrow$	10_{10}
B	$\rightarrow$	11_{10}
C	$\rightarrow$	12_{10}
D	$\rightarrow$	13_{10}
E	$\rightarrow$	14_{10}
F	$\rightarrow$	15_{10}

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:

1	$\times$	16	=	16
16	+	A	=	26

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

É facile convertire in ottale un numero binario. Basta tener presente che un insieme di tre bit binari puó rappresentare i numeri base del sistema ottale, cioé 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. Cosí 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 puó 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 sará 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, fornirá il numero binario richiesto.

Vediamo ció con un esempio

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

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

Il numero binario é quindi:

1 0 1 1 1 0 1 1 0

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.

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

Poiché la disuguaglianza é soddisfatta, scriviamo un 1, sottraiamo 1 dal risultato e moltiplichiamo per 2 (se non lo fosse scriveremmo 0 e moltiplicheremmo ancora per 2):

$$1.625 \ - \ 1 \ = \ 0.625$$

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

$$0.625 \times 2 \ = \ 1.25 \ \geq 1$$

$$1.25 \ - \ 1 \ = \ 0.25$$

Moltiplichiamo ancora 0.25 per 2 e verifichiamo:

$$0.25 \times 2 \ = \ 0.5 \quad < 1$$

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

Moltiplichiamo nuovamente per 2:

$$0.5 \times 2 \ = \ 1 \ \geq \ 1$$

$$1 \ - \ 1 \ = \ 0 \quad fine\, del\, processo$$

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:

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

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. Ció 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:

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

e per la costante di Boltzmann:

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

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 di seguito:

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

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

La caratteristica é ottenuta in realtá aggiungendo 64 all'esponente vero e proprio. Ció 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} , cioé in una potenza di 16

c) al valore di C_H ottenuto si somma 64 (decimale). Questa sará 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.

1.2 Il codice Grey

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 puó 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". Ció é fatto ad esempio dal codice Grey. In tale codice, i numeri binari 0 ed 1 corrispondono allo 0 ed 1 decimale rispettivamente:

0000	$\rightarrow$	0	
0001	$\rightarrow$	1	
$----$	$---$	$-----$	prima riflessione
0011	$\rightarrow$	2	
0010	$\rightarrow$	3	
$----$	$---$	$-----$	seconda riflessione
0110	$\rightarrow$	4	
0111	$\rightarrow$	5	
0101	$\rightarrow$	6	
0100	$\rightarrow$	7	

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.

Il processo puó 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. Cosí, nel passare dal 2 al 3 cambia solo il bit meno significativo, nel passare dal 3 al 4 cambia solo il terzo bit, etc. Ció non accade nel sistema binario "convenzionale" dove, nel passaggio da un valore a quello adiacente possono cambiare un numero arbitrario di bits. Ció 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.

1.3 Porte logiche

L'informazione elementare alla base del funzionamento di un qualsiasi sistema digitale é il bit. I due valori che questo puó 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 puó 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 piú basso dei valori di tensione ed

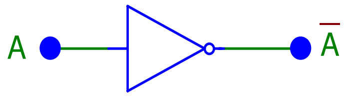


Figura 1.1:

all'1 il piú 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 puó acquistare o il valore 0 o il valore 1. A partire da una o piú 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, cosí anche una funzione logica di variabili logiche puó assumere solo i valori 1=T o 0=F.

La piú 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:

Una funzione logica puó esser caratterizzata scrivendo esplicitamente il valore assunto dalla funzione per ogni possibile valore del/degli ingressi. In genere si fa ció sotto forma di una tabella, detta "tabella delle veritá". Nel caso della funzione NOT questa é:

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

La funzione OR a due ingressi é definita dalla seguente tabella delle veritá:

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

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 +:

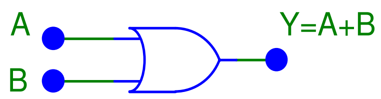


Figura 1.2:

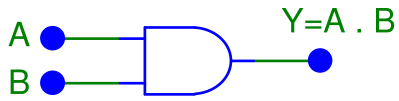


Figura 1.3:

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 seguente tabelle delle veritá:

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

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 quella della moltiplicazione: $A \cdot B$:

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

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

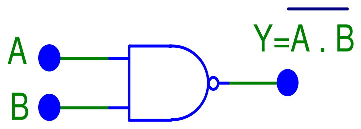


Figura 1.4:

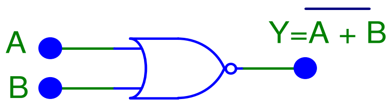


Figura 1.5:

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:

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

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

La funzione "OR esclusivo", indicata con XOR, ha la medesima tabella delle veritá 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 é quella per la quale é soddisfatto l'AND di A e B. Il simbolo del circuito che implementa la funzione XOR é mostrato in figura 1.6:

Il simbolo per l'XOR é il $\bigoplus$.

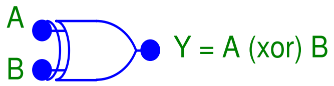


Figura 1.6:

1.4 Algebra di Boole

1.4.1 Identitá Booleane

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

Elenchiamo le piú 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 veritá, la (14):

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

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

$\overline{A}$	$\overline{B}$	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

Cambiare logica vuol dire complementare tutto, cioé scambiare 0 con 1 nella tabellina. Abbiamo indicato ció 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 é anche conseguenza delle leggi di De Morgan, che esamineremo tra poco.

Esaminiamo ora la tabella delle veritá della funzione XOR ($Y = A \bigoplus B$).

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

La tabella ci permette di scrivere diverse espressioni della funzione XOR, facendo uso di funzioni OR, NOT ed AND. Una prima espressione si puó ottenere esaminando le due righe in corrispondenza delle quali é Y = 1. La prima di queste corrisponde a A falso e B vero, cioé 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:

$$1)\ A \bigoplus B = \overline{A} \cdot B + A \cdot \overline{B} \tag{1.1}$$

Una seconda espressione dell'XOR puó essere ottenuta notando che l'XOR é vero quando é vero l'OR a condizione che sia falso l'AND. Cioé:

$$2)\ A \bigoplus B = (A + B) \cdot (\overline{A \cdot B}) \tag{1.2}$$

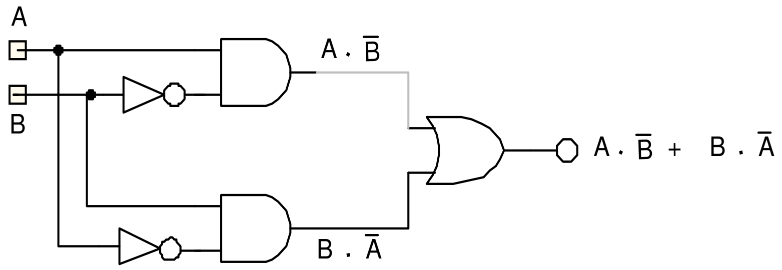


Figura 1.7:

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

$$3)\ A \bigoplus B \ = \ (A + B) \cdot (\overline{A} + \overline{B}) \tag{1.3}$$

Infine notiamo che il complemento dell'XOR é vero (cioé l'XOR é falso) nella prima e quarta riga della tabella delle veritá.

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

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

Cioé:

$$4)\ A \bigoplus B \ = \ \overline{\overline{A} \cdot \overline{B} + A \cdot B} \tag{1.4}$$

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

1.5 Leggi di De Morgan

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

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

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

La prima di queste ci dice che il complemento dell'AND di piú variabili logiche é uguale all'OR dei complementi delle singole variabili. La seconda dice che il complemento dell'OR di piú variabili logiche é uguale all'AND dei complementi

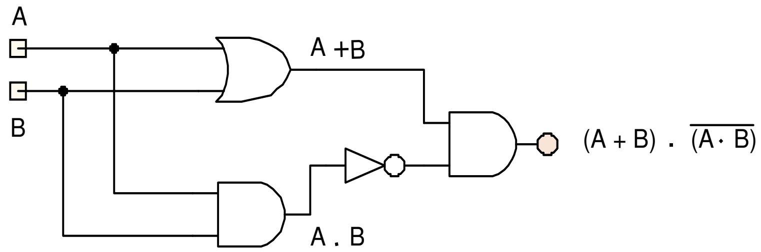


Figura 1.8:

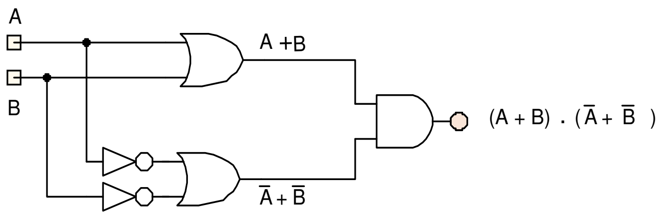


Figura 1.9:

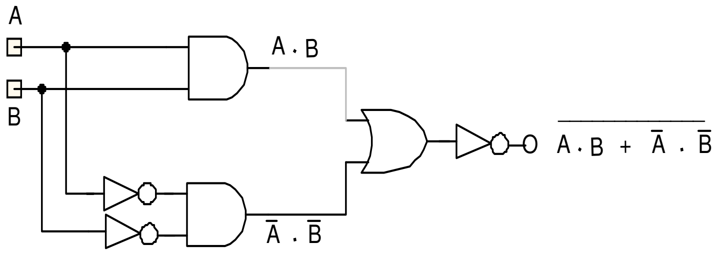


Figura 1.10:

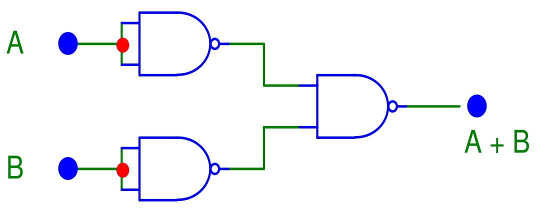


Figura 1.11:

delle singole variabili. Le leggi di De Morgan sono utilissime nel semplificare o trasformare espressioni logiche in piú 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:

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

Complementandola due volte, otteniamo:

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

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 complete. 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

1.6 Forme standard di funzioni logiche

Nel seguito, per semplificare la notazione, ometteremo il simbolo $\cdot$ che indica il prodotto logico di cue 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 puó essere riscritta nella forma:

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

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 ció con un esempio.

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

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 puó esser semplificata:

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

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

Un esempio di prodotto di somme é:

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

Ciascuno dei termini del prodotto é un "maxterm". La funzione scritta puó 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 puó facilmente essere ottenuta nel caso in cui la funzione L sia definita da una tabella della veritá. Ad esempio, facendo uso della medesima tabella usata sopra, calcoliamo $\overline{L}$:

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

da cui:

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

$$(X + Y + Z)(X + Y + \overline{Z})(\overline{X}Y\overline{Z})$$

che é un prodotto di somme in forma estesa.

1.7 Operazioni aritmetiche

Vediamo ora come, facendo uso di circuiti elementari che realizzino delle 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 seguente tabella delle veritá:

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

Vediamo facilmente che Σ coincide con $a \bigoplus b$, 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 a e del corrispondente di b potrá 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

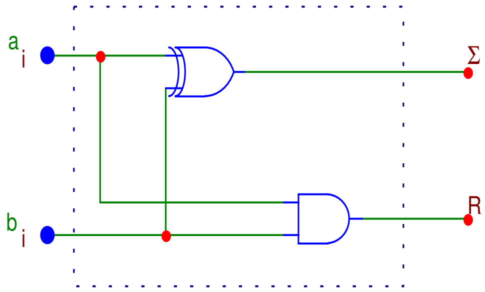


Figura 1.12:

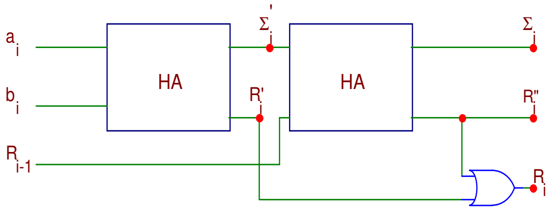


Figura 1.13:

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 occorrerá 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 puó in linea di principio essere ottenuto da due H.A., nel modo indicato in figura 1.13.

E' facile verificare che il riporto finale R_i é l'XOR 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 veritá per i termini a_i, b_i, R_{i-1}, Σ_i, R_i mostrata qui di seguito.

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

Dalla tabella delle veritá si ottiene:

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

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

Notiamo che R_i puó essere semplificato in:

$$R_i = b_i R_{i-1} + a_i b_i + a_i R_{i-1}$$

Σ_i ed R_i possono facilmente essere realizzati utilizzando porte logiche elementari.

1.8 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:

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

ció che di solito si esprime dicendo che $\overline{A}$ 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:

$$A\Sigma\overline{A} = 1111 \tag{1.5}$$

implica:

$$A\Sigma\left(\overline{A}\Sigma 1\right) = 10000 \tag{1.6}$$

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

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

e quindi:

$$B\Delta A = \left(B\Sigma\overline{A}\Sigma 1\right)\Delta 10000 \tag{1.7}$$

Dalla 1.7 segue che per sottrarre A da B occorre:

a complementare A, ottenendo $\overline{A}$

b sommare B ad $\overline{A}$

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 giá appreso, cioé 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'é sempre (cioé il numero ottenuto effettuando la somma: $B\Sigma\overline{A}\Sigma 1$ vale sempre 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 $(B\Sigma\overline{A})$.

Riscriviamo infatti la 1.7 nella forma:

$$B\Delta A \;=\; B\Sigma\overline{A}\Delta 1111 \;=\; \Delta\left[1111\Delta\left(B\Sigma\overline{A}\right)\right] \;=\; \Delta\overline{\left[B\Sigma\overline{A}\right]}$$

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

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

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

$$B\Delta A \;=\; B\Sigma\overline{A}\Sigma 1$$

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

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

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

$$B\Sigma\overline{A}\Sigma 1 \;=\; 10011 \;\rightarrow 0011 \;=\; 3$$

Notiamo per inciso che se B é maggiore di A, la somma di B e di $\overline{A}$ avrá sempre il bit piú significativo uguale ad 1. Ció suggerisce di usare tale bit come addendo (cioé 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 varrá 1 e ció é quel che occorre per realizzare la sequenza di operazioni sopra descritte. Se invece B é minore di A, la somma $B\Sigma\overline{A}$ avrá il bit piú significativo uguale a 0 ed aggiungerlo quindi alla soma $B\Sigma\overline{A}$ non cambierá nulla.

Vediamolo nel caso concreto in cui:

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

Avremo:

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

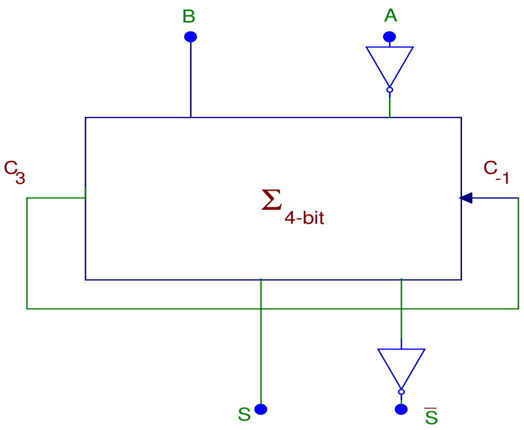


Figura 1.14:

Se complementiamo tale numero, troviamo:

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

da interpretare come numero negativo (cioé 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 (cioé 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 della somma (b5)

d se b5=1, il risultato ottenuto al punto (c), dopo aver lasciato cadere il bit piú significativo, rappresenta la differenza (positiva) $B\Delta A$

e se invece b5=0, il complemento del risultato ottenuto al punto (c) rappresenta il valore assoluto della differenza (negativa) $B\Delta A$.

Tutto ció é riassunto schematicamente in figura 1.14.

Il circuito contenuto nella scatolina 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 c3=1, la differenza $B\Delta A$ (positiva) é data da S.

1.9 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 puó 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 $C_2(N) + N = 100000000$:

$$\begin{array}{rc} 10110100 & + \\ 01001100 & = \\ \hline 100000000 & \end{array}$$

Cosí, se il primo bit sulla destra é un 1, il complemento a 2 puó 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 occorrerá 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 questa, 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 ció con un esempio, cominciando dal caso in cui sia A che B sono positivi. La somma é data da:

$$\begin{array}{rcccc} & & \text{binario} & \text{decimale} & \\ \text{A} & = & 00110 & 6 & + \\ \text{B} & = & 00101 & 5 & = \\ A\Sigma B & = & 01011 & 11 & \end{array}$$

Calcoliamo ora la differenza $A\Delta B$:

$$\begin{array}{rcccc} & & \text{binario} & \text{decimale} & \\ \text{A} & = & 00110 & 6 & + \\ -B = C_2(B) & = & 11011 & 5 & = \\ A\Sigma C_2(B) & = & 100001 & 1 & \end{array}$$

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

$$
\begin{array}{rcrrl}
 & & \text{binario} & \text{decimale} & \\
\mathrm{B} & = & 00101 & 5 & + \\
-A = C_2(A) & = & 11010 & 6 & = \\
B\Sigma C_2(A) & = & 11111 & -1 &
\end{array}
$$

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.

1.10 Comparatori digitali

Ammettiamo di voler comparare due semplici bit A e B. Le possibili configurazioni sono:

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	$A = B$	$A \cdot B$

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:

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

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

In definitiva abbiamo:

$$
\begin{array}{lr}
A > B & Y \; = \; A \cdot \overline{B} \; = \; 1 \\
A < B & Y \; = \; B \cdot \overline{A} \; = \; 1 \\
A = B & E \; \equiv \; Y \; = \; \overline{A \cdot \overline{B} + B \cdot \overline{A}} \; = \; 1
\end{array}
$$

Un circuito che fornisce in uscita la variabile Y é quello di figura 1.15.

Se vogliamo comparare dei numeri costituiti da piú bit, il circuito dovrá comparare i singoli bit ed un po' di logica aggiuntiva sará necessaria per ottenere la risposta. Ammettiamo ad esempio di voler comparare due numeri di 4 bit. L'uguaglianza dei due numeri richiederá ovviamente l'uguaglianza due a due dei bit corrispondenti.

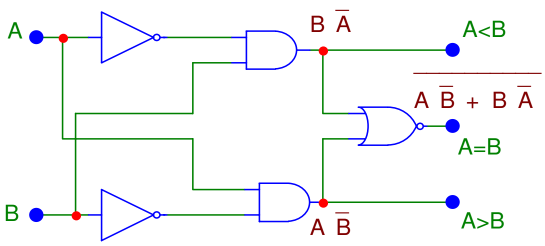


Figura 1.15:

Cosí, 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 richiederá:

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

cioé:

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

A sará 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 $A_0 > B_0$

Cioé:

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

La condizione $A < B$ si realizza scambiando A con B.

1.11 Paritá di un numero binario

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

L'uso della paritá é utile per controllare se un numero binario é 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 é inevitabilmente presente sulle linee usate per la trasmissione puó far sí che un bit cambi valore. Se ammettiamo che la probabilitá che, per effetto del rumore in una medesima parola (e.g. di 8 bit), 2 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 paritá del numero. Cosí, se i numeri hanno 8 bit ed il numero in questione ha paritá dispari, potremo aggiungere un nono bit di valore 1 che renda la paritá complessiva pari. Se il numero di 8 bit aveva paritá pari, aggiungeremo un nono bit di valore 0, con che la paritá complessiva rimane pari. In ricezione, adopereremo un circuito che controlli

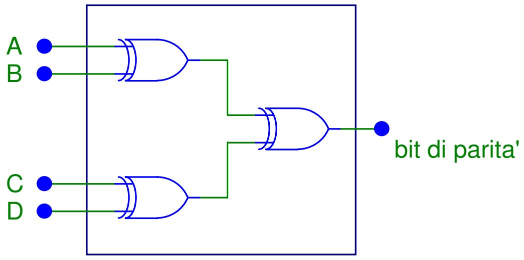


Figura 1.16:

la paritá e verifichi che sia sempre pari. Se essa risulterá dispari il sistema denuncerá l'errore al circuito da cui l'informazione proviene, il quale invierá nuovamente il dato. Ovviamente, é possibile adottare la convenzione di paritá dispari, anziché pari come nell'esempio fatto.

Esaminiamo ora un semplice circuito che genera il bit di paritá per una parola di 4 bit.

Se avessimo solo due bit, la paritá sarebbe:

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

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

Se il sistema ha quattro bit ($ABCD$), potremo definire la paritá della coppia (AB), quella della coppia (CD) ed infine quella dei due bit di paritá cosí ottenuti. Occorrerá cioé una cascata di XOR, come mostrato in figura 1.16.

1.12 Codificatori/Decodificatori

Un codificatore é un circuito che converte in un codice un'informazione corrispondente al verificarsi di uno tra n possibili eventi. Ad esempio, un codificatore é l'insieme dei circuiti presenti in un personal computer, che generano un numero binario (codice) quando si pigia un determinato tasto. Un decodificatore é viceversa un circuito che riceve in ingresso un codice (cioé 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 é un modo per rappresentare (codificare) i numeri decimali. In tale sistema i numeri da 0 a 9 sono rappresentati da 4 bit:

$$
\begin{array}{rcl}
0 & = & 0000 \\
1 & = & 0001 \\
2 & = & 0010 \\
3 & = & 0011 \\
4 & = & 0100 \\
5 & = & 0101 \\
6 & = & 0110 \\
7 & = & 0111 \\
8 & = & 1000 \\
9 & = & 1001
\end{array}
$$

Un decodificatore BCD-Decimale riceverá in ingresso i quattro bit ed attiverá 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 piú cifre é rappresentato da tanti gruppi di quattro bit quante sono le cifre decimali che lo costituiscono. Ad esempio:

371 =	0011	0111	0001
	3	7	1

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 $(\overline{a}_3, \overline{a}_2, \overline{a}_1, \overline{a}_0)$ i loro complementi, avremo:

$$\overline{a}_3 \cdot \overline{a}_2 \cdot \overline{a}_1 \cdot \overline{a}_0 \;\Rightarrow\; 0 \; decimale$$

$$\overline{a}_3 \cdot \overline{a}_2 \cdot \overline{a}_1 \cdot a_0 \;\Rightarrow\; 1 \; decimale$$

$$\overline{a}_3 \cdot \overline{a}_2 \cdot a_1 \cdot \overline{a}_0 \;\Rightarrow\; 2 \; decimale$$

$$\ldots$$

$$a_3 \cdot a_2 \cdot a_1 \cdot a_0 \;\Rightarrow\; 0 \; decimale$$

Un circuito che realizza tale funzione logica é quello mostrato in figura 1.17:

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 veritá che realizza la funzione di codifica é:

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:

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

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

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

$$Y_3 \;=\; T_8 + T_9$$

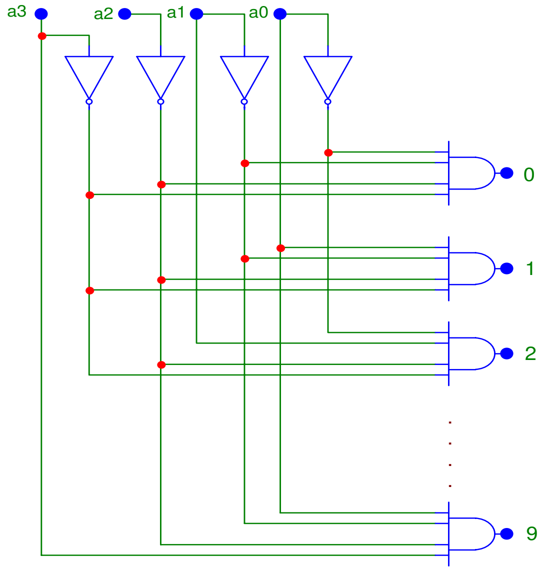


Figura 1.17:

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

Un circuito che realizza tale codifica puó esser ottenuto da un integrato a matrice di diodi.

Se in tale circuito due o piú tasti vengono premuti contemporaneamente, il risultato puó non essere quello desiderato. Un "codificatore con prioritá" elimina tale problema. In un codificatore dd questo tipo, se due o piú tasti sono pigiati insieme, l'azione che ha luogo é quella che si avrebbe pigiando solo il tasto piú alto nella sequenza $T_0 \cdots T_9$. Per ottenere tale funzione dobbiamo modificare la tabella delle veritá come segue:

Dove la x sta ad indicare che é 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:

$$Y_0 \;=\; T_1\overline{T}_2\overline{T}_3\overline{T}_4\overline{T}_5\overline{T}_6\overline{T}_7\overline{T}_8\overline{T}_9 \;+\; T_3\overline{T}_4\overline{T}_5\overline{T}_6\overline{T}_7\overline{T}_8\overline{T}_9 \;+$$

$$+T_5\overline{T}_6\overline{T}_7\overline{T}_8\overline{T}_9 \;+\; T_7\overline{T}_8\overline{T}_9 \;+\; T_9$$

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	0	1	x	x	x	0	0	1	1
0	0	0	0	0	1	x	x	x	x	0	1	0	0
0	0	0	0	1	x	x	x	x	x	0	1	0	1
0	0	0	1	x	x	x	x	x	x	0	1	1	0
0	0	1	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

Questa si puó semplificare in:

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

con:

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

dove:

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

Ed infine:

$$Y_0 \;=\; T_9 \;+\; \overline{T}_8\left(T_7 + \overline{T}_6\left(T_5 + C\right)\right)$$

con:

$$C \;=\; \overline{T}_4\left(\overline{T}_1 T_1 + T_3\right)$$

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

1.13 Multiplexer/Demultiplexer

Ammettiamo di voler "incanalare" su di un'unica linea dati che provengono da una di N linee diverse. Si puó 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 é quello mostrato in figura 1.18

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 sará abilitata [1].

[1]si dice che una porta é abilitatata se essa é 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 é uguale a 0 l'uscita sará 0 qualunque siano i valori presenti sugli altri ingressi. Si dice che in tal caso la porta é disabilitata

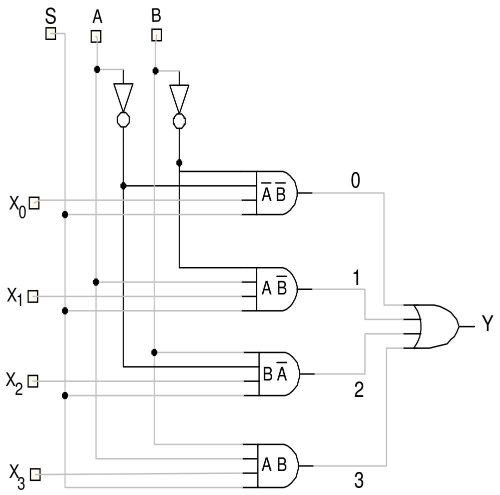


Figura 1.18:

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

Un multiplexer puó esser utilizzato per effettuare la conversione parallelo- seriale. Ammettiamo infatti che ad un circuito arrivino dei bit, costituenti una "parola", su n linee parallele e che esso debba poi convogliare tali informazioni su di un'unica linea, come mostrato in figura 1.19.

Si potrá usare un multiplexer che riceve sulle linee x_0, x_1,...,x_n i bit da convogliare in uscita; gli ingressi di controllo (o di indirizzo) c_0, $c_1, \cdots, c_n$, 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 sará convogliato per primo in uscita, seguirá poi x_1, poi x_2 etc. Ovviamente il numero degli ingressi n

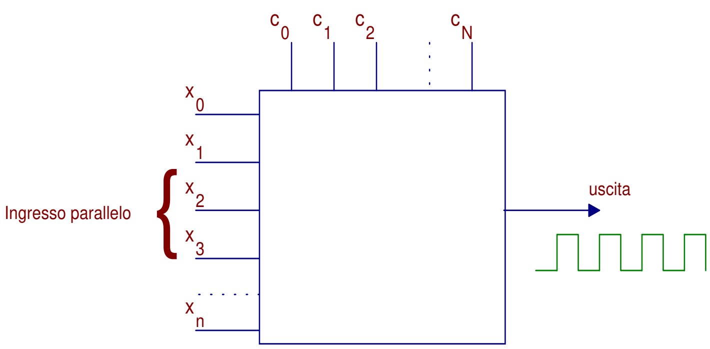


Figura 1.19:

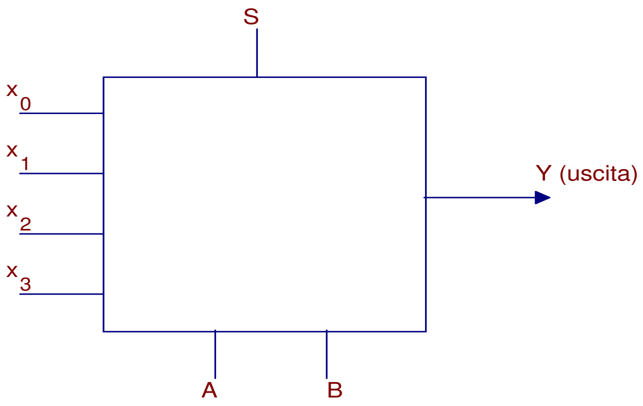


Figura 1.20:

dovrá esser legato al numero di ingressi di controllo N dalla relazione $n = 2^N$. Cosí, per $n = 4$ avremo $N = 4$, per $n = 8$, $N = 3$ etc.

Un demultiplexer convoglia invece su una di n possibili linee di uscita i segnali che riceve su di una linea d'ingresso, facendo uso dell'informazione (indirizzo) che riceve su N linee di controllo. E' utile far presente che qualsiasi decodificatore puó 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 1.20

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 veritá é la seguente:

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

Y puó anche essere scritto come:

$$Y = x_0\overline{A}\,\overline{B} + x_1 A\overline{B} + x_2\overline{A}\,B + x_3 AB \tag{1.8}$$

Se ora volessimo implementare la seguente funzione logica:

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

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

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

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:

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

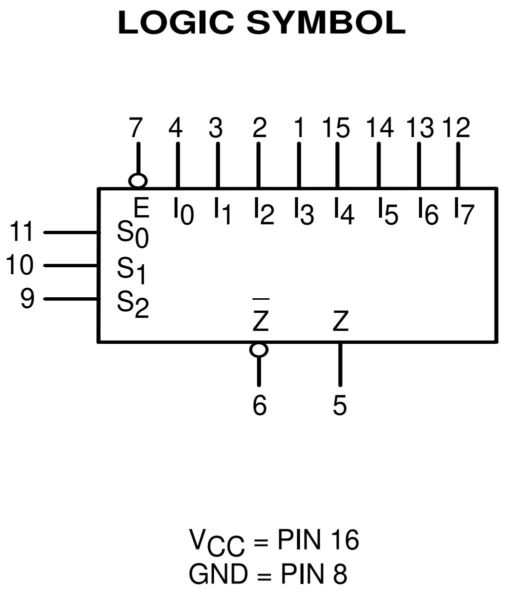


Figura 1.21:

Basterá quindi utilizzare il moltiplexer 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 microprocessori e piú in generale in tutti i sistemi digitali. Esistono in commercio numerosi integrati dedicati allo scopo. In figura 1.21 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 1.22 é utilizzabile come demultiplexer da 1 ad 8.

1.14 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 puó contenere ad esempio un byte (8 bit), una parola di 16 bit o una parola 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. Esiste in realtá un'ampia varietá di FF. Quello che abbiamo sommariamente descritto é il flip-flop di tipo RS. Un FF puó 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 cosí che un FF puó essere adoperato per memorizzare

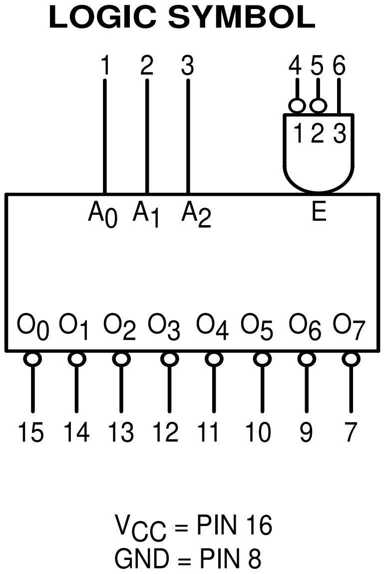


Figura 1.22:

un bit. I valori delle uscite sono determinati dai valori dei segnali presenti sugli ingressi R, S, secondo la seguente tabella delle veritá:

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

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 ció piú chiaramente, cominciamo con l'esaminare la struttura interna di un FF di tipo RS, che utilizza due porte NAND:

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 $\overline{Q}_{n-1}$, cioé 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 avrá $Q_n = 1$, $\overline{Q}_n = 0$. Vediamo cosí 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 sará 1 qualunque sia il segnale presente sull'altro ingresso (in tale situazione la porta é disabilitata). Vediamo quindi che, con R=S=0

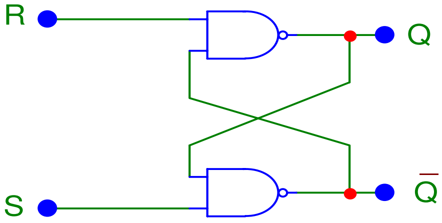


Figura 1.23:

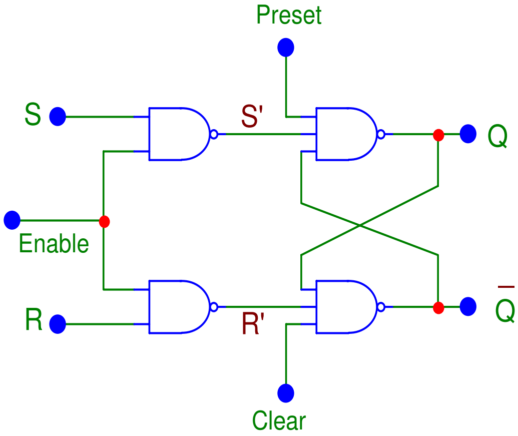


Figura 1.24:

avremmo $Q = \overline{Q} = 1$. Tale situazione é anomala, poiché vogliamo limitarci a due stati: $(Q = 1\,; \overline{Q} = 0$, $(Q = 0\,; \overline{Q} = 1$. C'é 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 andrá nello stato di Set ($Q = 1$), altrimenti andrá in quello di Reset ($\overline{Q} = 1$). Poiché il ritardo relativo é un elemento aleatorio é chiaro come tale situazione non sia accettabile. Vedremo in seguito come in alcuni tipi di FF (ad esempio i JK) non esista alcun stato proibito.

Si puó ottenere un circuito con la medesima tabella delle veritá di quello precedente, facendo precedere ciascuno dei due gates NAND da un invertitore e scambiando i ruoli di Q e $\overline{Q}$ come mostrato in figura:

Ammettiamo che gli ingressi di *Preset* e di *Clear* siano entrambi alti. Se l'enable é alto, le porte NAND, in ingresso saranno "abilitate" (cioé le loro uscite, S' ed R', dipenderanno dai segnali presenti sugli ingressi S ed R). Ció é mostrato nella tabella che segue.

Vediamo che ora lo stato "proibito" é quello che corrisponde ad R=S=1. Se l'ingresso di enable é basso, le porte in ingresso risulteranno disabilitate (cioé non risponderanno piú ai segnali presenti sugli ingressi R ed S. Avremo in tale situazione

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			

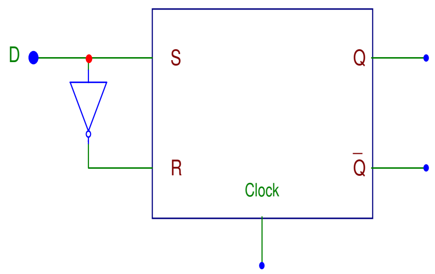


Figura 1.25:

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" puó essere utilizzato per un funzionamento "sincrono" di tale FF. Se cioé 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 é 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 cioé 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 é 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 $\overline{Q} = 0$, $Q = 1$.

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).

La presenza dell'invertitore fa sí che se S=1, sará 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 é quello mostrato in figura 1.26.

Questo (SN74LS74) é 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]

[2]Nella 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 é un caso!) si riferiscono al range di temperatura in cui puó operare: 74 implica un'applicazione "commerciale", per un range di temperatura che va da 0^oC a 70^oC. Se trovassimo un 54 al posto di tali due cifre, ció implicherebbe un dispositivo adatto ad applicazioni "militari", in un range di temperature

MOTOROLA

SN54/74LS74A

DUAL D-TYPE POSITIVE EDGE-TRIGGERED FLIP-FLOP

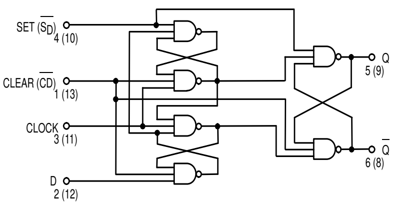
The SN54/74LS74A dual edge-triggered flip-flop utilizes Schottky TTL circuitry to produce high speed D-type flip-flops. Each flip-flop has individual clear and set inputs, and also complementary Q and $\overline{Q}$ outputs.

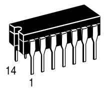
Information at input D is transferred to the Q output on the positive-going edge of the clock pulse. Clock triggering occurs at a voltage level of the clock pulse and is not directly related to the transition time of the positive-going pulse. When the clock input is at either the HIGH or the LOW level, the D input signal has no effect.

DUAL D-TYPE POSITIVE EDGE-TRIGGERED FLIP-FLOP

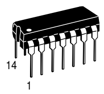
LOW POWER SCHOTTKY

LOGIC DIAGRAM (Each Flip-Flop)

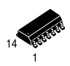




J SUFFIX
CERAMIC
CASE 632-08



N SUFFIX
PLASTIC
CASE 646-06



D SUFFIX
SOIC
CASE 751A-02

ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

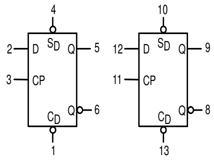
MODE SELECT — TRUTH TABLE

OPERATING MODE	INPUTS			OUTPUTS	
	$\overline{S_D}$	$\overline{S_D}$	D	Q	$\overline{Q}$
Set	L	H	X	H	L
Reset (Clear)	H	L	X	L	H
*Undetermined	L	L	X	H	H
Load "1" (Set)	H	H	h	H	L
Load "0" (Reset)	H	H	l	L	H

* 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. If the levels at the set and clear are near V_{IL} maximum then we cannot guarantee to meet the minimum level for V_{OH}.

H, h = HIGH Voltage Level
L, I = LOW Voltage Level
X = Don't Care
i, h (q) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the HIGH to LOW clock transition.

LOGIC SYMBOL



V_{CC} = PIN 14
GND = PIN 7

Figura 1.26:

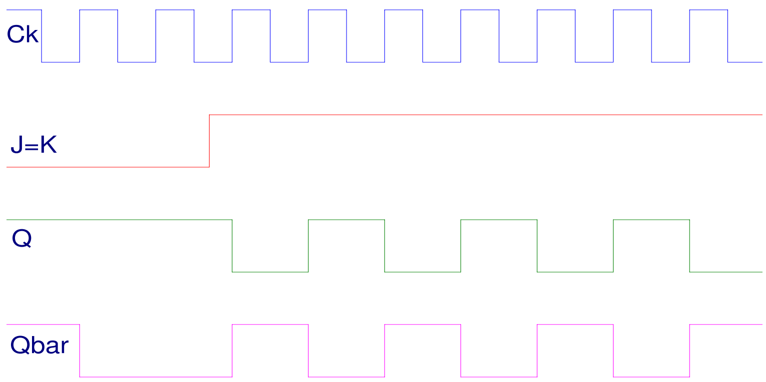


Figura 1.27:

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: $Q_n = \overline{Q}_{n-1}$, $\overline{Q}_n = Q_{n-1}$. Ció puó essere ottenuto facendo precedere ciascuno dei due ingressi di un FF RS, da una porta AND, come mostrato in figura.

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

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

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

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 seguirá $Q = 0$, $\overline{Q} = 1$.

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 $Q_n = 1$, $\overline{Q}_{n-1} = 0$ 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 sará $Q = 0$, $\overline{Q} = 1$, cioé il sistema passerá nello stato complementare di quello in cui era. Il successivo impulso di clock troverá ora abilitata la porta superiore e disabilitata quella inferiore. Di conseguenza, con tale impulso di clock Q diventerá 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). Ció é mostrato nel grafico di figura 1.27:

Notiamo che la frequenza dell'onda rettangolare $Q(\overline{Q})$ é la metá di quella del clock. Un FF di tipo JK con J=K=1 effettua quella che vien chiamata la funzione

compreso tra -55^oC e $+125^oC$. La sigla LS sta poi ad indicare la classe o tipo del chip. LS sta per "Low power Schottky".

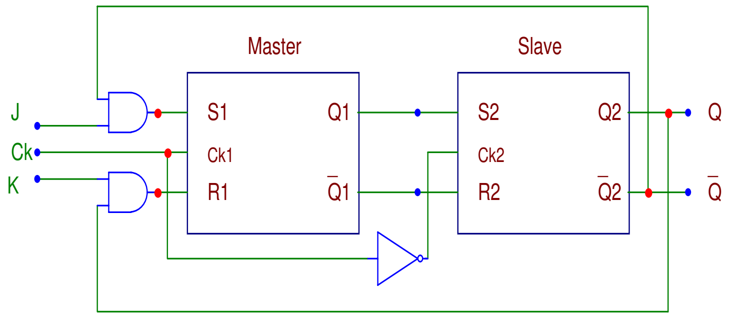


Figura 1.28:

"TOGGLE" ed é adoperato come divisore di frequenza. Ponendone due o piú in cascata si puó realizzare una divisione della frequenza del clock per 4 o per potenze di 2 piú elevate. L'operazione di questo circuito potrá 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 puó 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 1.28 (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 puó 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 sono letti se non in corrispondenza della transizione negativa dell'impulso di clock. In maniera analoga, le uscite devono attendere le successive transizioni negative del clock per poter variare.

La figura 1.29 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 1.30. 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.

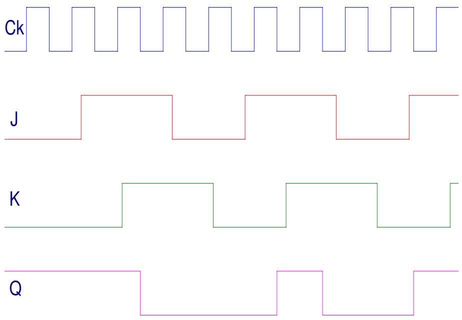
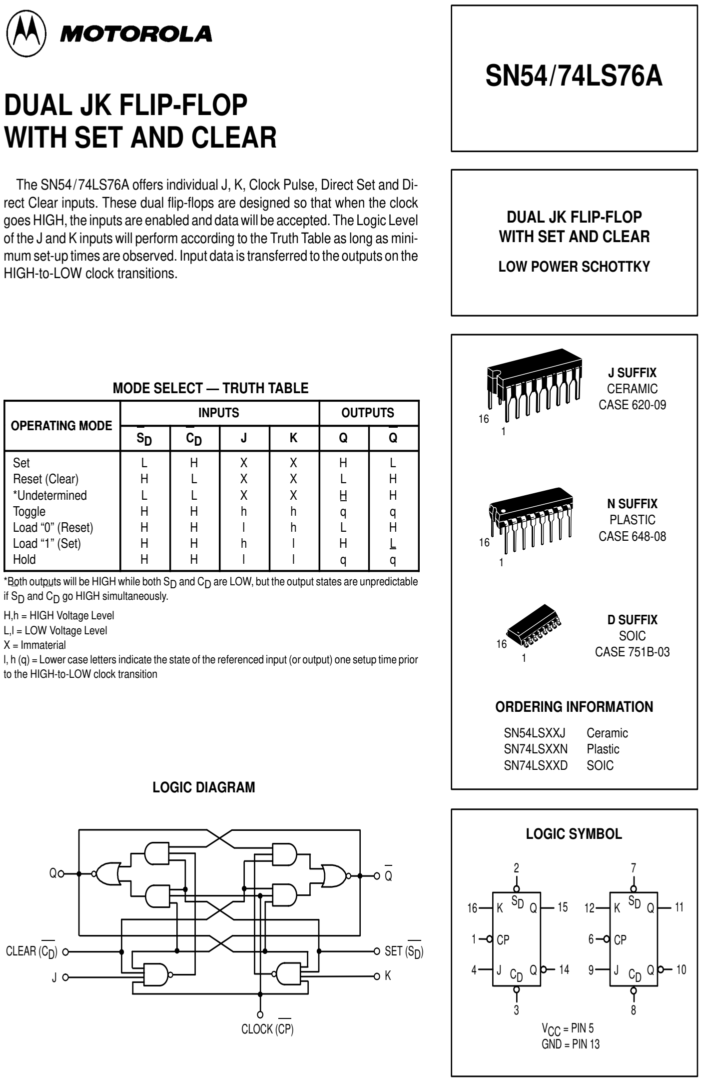


Figura 1.29:



MOTOROLA

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	
	$\overline{S}_D$	$\overline{C}_D$	J	K	Q	$\overline{Q}$
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	$\overline{q}$	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	$\overline{q}$

*Both outputs will be HIGH while both $\overline{S}_D$ and $\overline{C}_D$ are LOW, but the output states are unpredictable if $\overline{S}_D$ and $\overline{C}_D$ 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

Q, $\overline{Q}$, CLEAR ($\overline{C}_D$), SET ($\overline{S}_D$), J, K, CLOCK ($\overline{CP}$)

SN54/74LS76A

DUAL JK FLIP-FLOP WITH SET AND CLEAR

LOW POWER SCHOTTKY

J SUFFIX CERAMIC CASE 620-09

N SUFFIX PLASTIC CASE 648-08

D SUFFIX SOIC CASE 751B-03

ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

LOGIC SYMBOL

V_{CC} = PIN 5
GND = PIN 13

Figura 1.30:

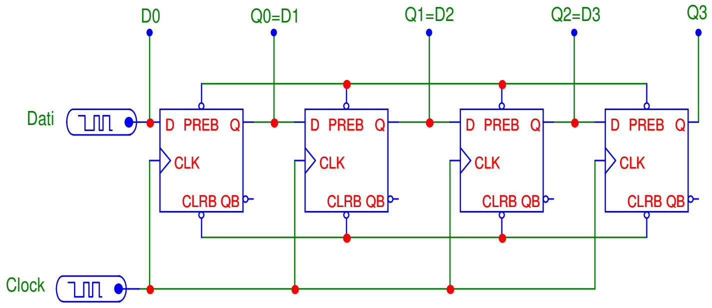


Figura 1.31:

1.15 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" di bits. Esiste una grande varietá 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. Uno SR costituito da FF di tipo D é mostrato in figura 1.31.

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 trasferirá tale dato in $Q_0 = D_1$. Il successivo segnale di clock trasferirá il dato in $Q_1 = D_2$, il terzo lo trasferirá 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 1.32.

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

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, é chiamato "Serial-In-Parallel-Out" (o SIPO).

Il circuito che ora esamineremo puó essere utilizzato indifferentemente per immettere dei dati in parallelo ed estrarli in modo seriale, o viceversa. Esso puó 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) é mostrato in figura 1.33. Esso é noto come "Universal Shift Register".

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

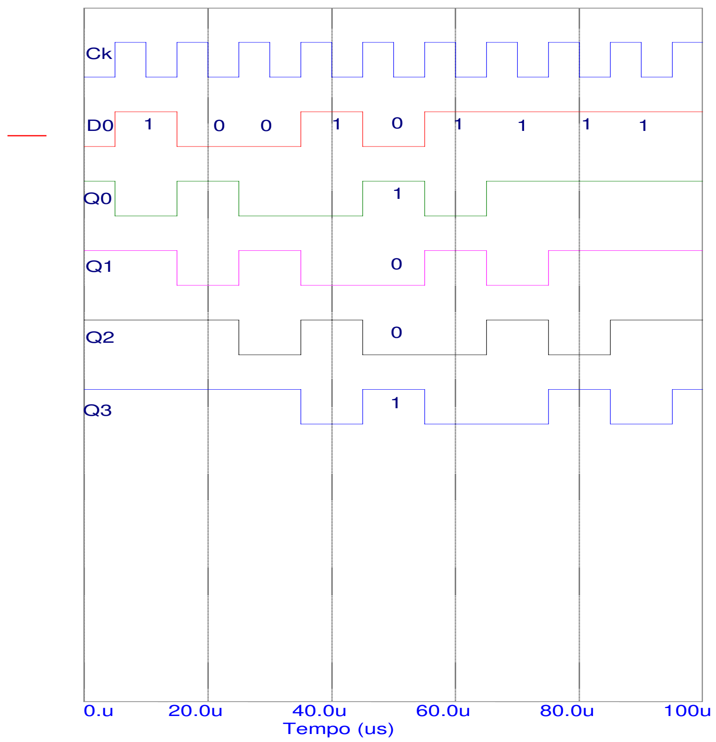


Figura 1.32:

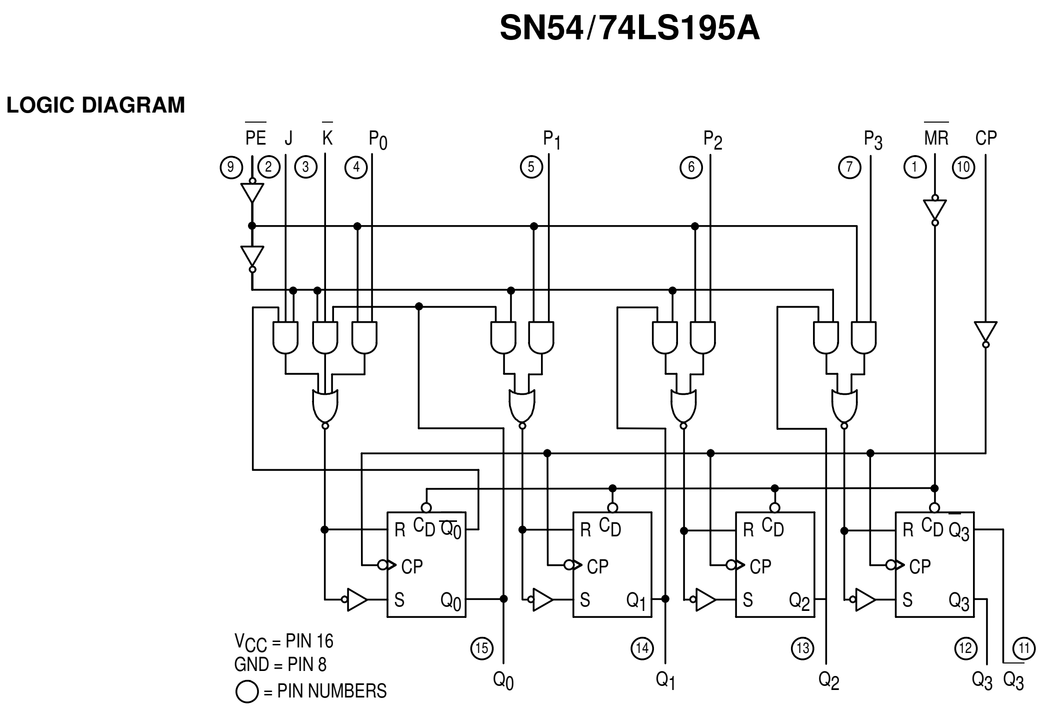


Figura 1.33:

MODE SELECT — TRUTH TABLE

OPERATING MODES	INPUTS					OUTPUTS				
	MR	PE	J	K	P_n	Q_0	Q_1	Q_2	Q_3	$\overline{Q_3}$
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	$\overline{q_2}$
Shift, Reset First	H	h	l	l	X	L	q_0	q_1	q_2	$\overline{q_2}$
Shift, Toggle First Stage	H	h	h	l	X	q_0	q_0	q_1	q_2	$\overline{q_2}$
Shift, Retain First Stage	H	h	l	h	X	q_0	q_0	q_1	q_2	$\overline{q_2}$
Parallel Load	H	l	X	X	p_n	p_0	p_1	p_2	p_3	$\overline{p_3}$

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 1.34:

La tabella delle veritá mostrata in figura 1.34 illustra le caratteristiche di funzionamento.

I dati sono immagazzinati nei quattro FF mostrati in figura. Se l'ingresso indicato con $\overline{PE}$ (Preset Enable) é 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 avrá l'uscita alta e, a causa della doppia inversione presente all'uscita 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 sará uguale all'ingresso J. Di conseguenza l'uscita Q_0 diverrá 1 con il successivo impulso di Clock (piú 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 sará 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 sará uguale a 0 e quello di Reset 1. Con ció, l'uscita Q_0 diverrá uguale a 0 in corrispondenza al fronte di salita del prossimo impulso di Clock. Tale caricamento seriale del dato é quello indicato nella seconda e terza riga della tabella (Shift-Set First Stage - Shift-Reset) di figura 1.34.

Esaminiamo ora ció 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 é abilitata (e la seconda disabilitata). Se esaminiamo la porta AND associata al secondo blocco, vediamo che essa ha come ingresso Q_0. Il collegamento della porta OR sottostante é tale che il Set del relativo FF é anch'esso uguale a Q_0. Vediamo cosí che il successivo impulso di Clock fará 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 porta AND del terzo blocco fará 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 fará si che Q_3 divenga uguale a Q_0. Lo shift di quattro posizioni del dato impostato in J é cosí completato con l'applicazione di quattro impulsi di Clock.

Se il $\overline{PE}$ é 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: $P_0 = 1, P_1 = 0, P_2 = 1, P_3 = 0$. Ció implicherá: $S_0 = 1, S_1 = 0, S_2 = 1, S_3 = 0$.

Questi dati saranno trasferiti in Q_0, Q_1, Q_2, Q_3 sul fronte di salita del successivo impulso di Clock. Si é in tal modo effettuato il *caricamento* dei dati impostati sugli ingressi $(P_0 \cdots P_3)$ in $(Q_0 \cdots Q_3)$. Ció é indicato nell'ultima riga della tabella di figura 1.34.

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 troverá abilitata la prima porta AND di ciascun blocco, collegata all'uscita Q del blocco precedente. Il dato sará cosí trasferito da Q_i a Q_{i+1}.

L'ingresso indicato con $\overline{MR}$ (Master Reset) puó, come mostrato nella prima riga della medesima tabella, essere adoperato per porre uguali a zero le uscite $Q_0 \cdot Q_3$ 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 (quinta riga della tabella delle veritá).

Se $J = 1$ e $\overline{K} = 0$, la prima delle porte AND presenti nel primo blocco di figura sará abilitata, mentre la seconda é disabilitata. In tali condizioni l'uscita del primo AND dipenderá dal valore di $\overline{Q}_0$ e quindi di Q_0. Se $\overline{Q}_0$ é alto (Q_0 basso) l'uscita dell'AND sará alta e tale risulterá quindi l'ingresso di Set del primo FF. Con ció $Q_0 \to 1$ e $\overline{Q} \to 0$ 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 cosí che con $J = 1$ e $\overline{K} = 0$ il primo FF effettua la funzione "TOGGLE" caratteristica di un JK.

Il secondo FF, come é facile verificare, acquisterá il valore di Q_0 (cioé 1) con il secondo impulso di Clock, il terzo FF con il terzo impulso di Clock etc.

Se invece é $J = 0$ e $\overline{K} = 1$, vediamo che verrá 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, sará bassa. Con ció sará alto l'ingresso di Reset del FF e Q_0 rimarrá 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 rimsasto uguale ad 1. Tale funzione é indicata con "Shift, Retain First Stage" nella quinta colonna della tabella.

Un registro a scorrimento simile per molti rispetti al precedente, ma con la possibilitá di shift sia a destra che a sinistra, é il SN74LS194, la cui struttura é mostrata in figura 1.35 e la cui tabella delle veritá é illustrata in figura 1.36.

Un ulteriore tipo di registro che é ampiamente adoperato é il cosiddetto "First-In, First-Out" (FIFO). Lo schema é mostrato in figura 1.37.

Come vediamo, esso consiste di un normale SR con possibilitá di ingressi paralleli. Il dato in input é 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 é normalmente alto. Un opportuno circuito di comando fa sí 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

SN54/74LS194A

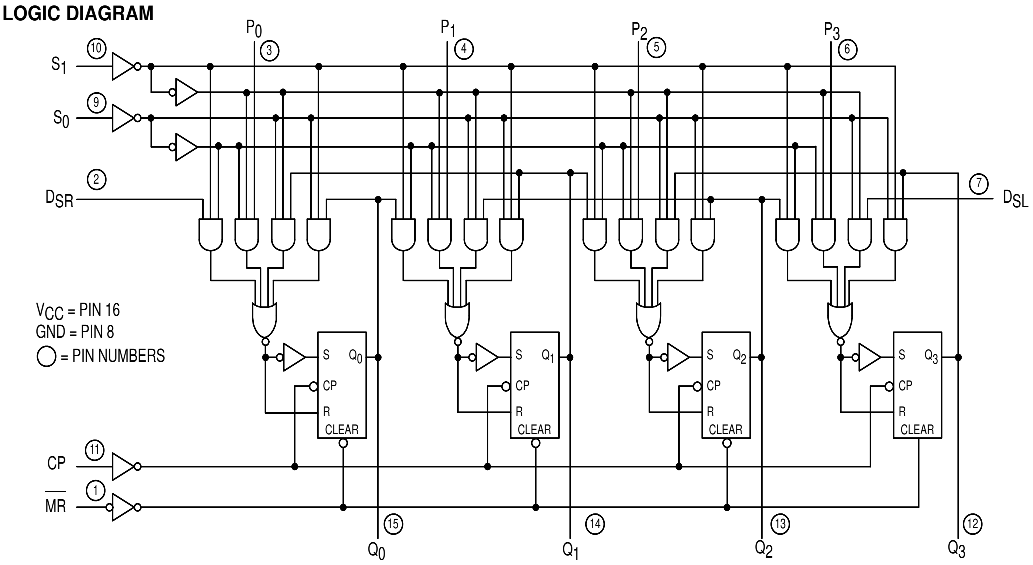


Figura 1.35:

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	I	I	X	X	X	q_0	q_1	q_2	q_3
Shift Left	H H	h h	I I	X X	I h	X X	q_1 q_1	q_2 q_2	q_3 q_3	L H
Shift Right	H H	I I	h h	I h	X X	X X	L H	q_0 q_0	q_1 q_1	q_2 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
I = 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 1.36:

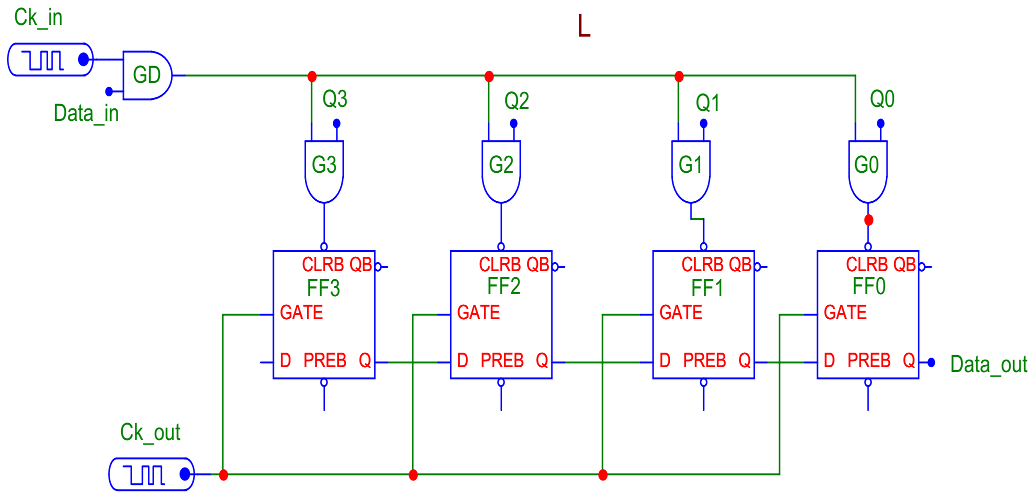


Figura 1.37:

abilitando l'immissione sulla linea L del dato corrispondente. Vediamo ció piú in dettaglio:

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

(b) il secondo bit é immesso in ingresso ed il secondo impulso di clock applicato. Questo abilita G_D e manda $Q_0 \to 0$ e $Q_1 \to 1$. 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 $Q_0, Q_1 \to 0$ e $Q_2 \to 1$. Ora G_2 sará 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 $Q_0, Q_1, Q_2 \to 0$ e $Q_3 \to 1$. Ora G_3 sará 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 $0000, 0001, 0010, 0011, \cdots$) se il clock in arrivo é Ck_{in}; decrescente (cioé, $\cdots 0011, 0010, 0001, 0000$) se il clock in ingresso é Ck_{out}. Ció é mostrato schematicamente in figura 1.38.

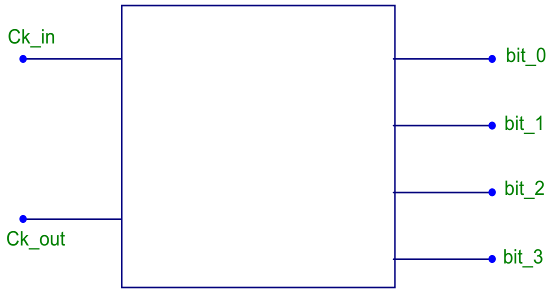


Figura 1.38:

1.16 Applicazioni degli Shift-Registers e dei Flip-Flop

1.16.1 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 puó poi esser utilizzato per dar inizio ad una sequenza di operazioni. Immaginiamo infatti di collegare l'uscita dell'ultimo FF di uno SR come quello mostrato in figura 1.39, con l'ingresso di Set del primo FF.

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

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).

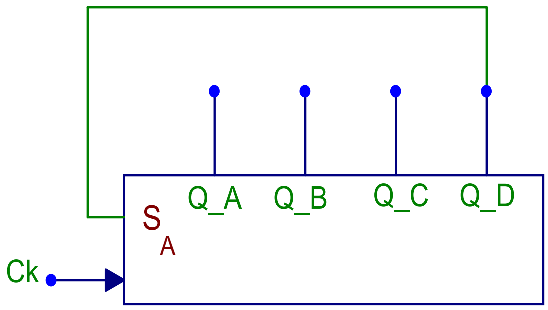


Figura 1.39:

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

Ma é 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 che segue.

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

Notiamo ancora che, nel primo degli esempi visti, Q_D vale 1 ogni 4 impulsi di clock. Un tale sistema puó quindi essere anche adoperato come "divisore di frequenza" (per 4 nell'esempio considerato, ma si puó ottenere una divisione per potenze piú elevata di 2, con registri a molti bit). Un tale tipo di utilizzo, dove l'uscita dell'ultimo FF é 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 é costituito sono di tipo RS o JK, é possibile collegare l'uscita dell'ultimo FF della catena con l'ingresso R (o K) del primo. Si ottiene cosí quello che é noto come "Johnson Counter" o "twisted-ring Counter". Si puó verificare che con tale sistema si puó ottenere una divisione di frequenza per 2N, dove N é la lunghezza del registro (anziché per N come in un normale Ring Counter).

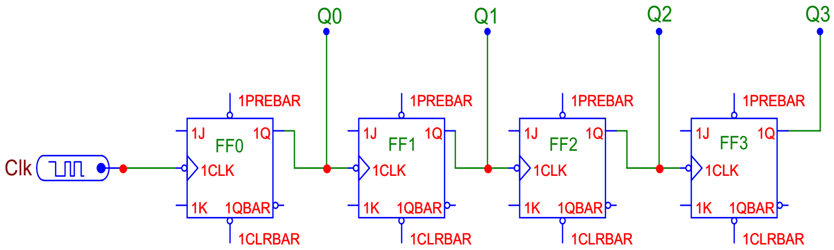


Figura 1.40:

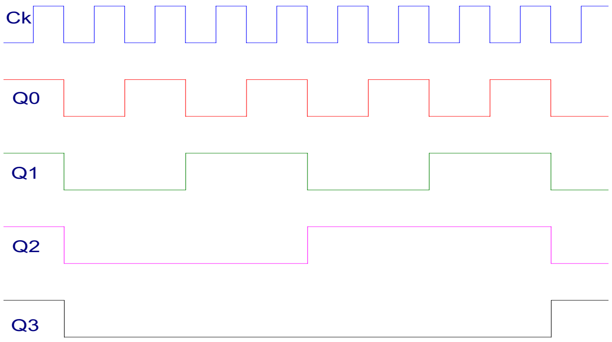


Figura 1.41:

1.16.2 Applicazioni dei FF

I FF sono ampiamente usati per realizzare dei contatori. Consideriamo come primo esempio quello di figura 1.40 (noto come ripple counter).

Ammettiamo che tutti gli ingressi J e K siano fissi a +1. In tal caso i quattro FF effettueranno la funzione TOGGLE, cioé cambieranno stato ad ogni impulso presente sull'ingresso di clock. Piú precisamente, la transizione delle uscite avverrá 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 $Q_3 - Q_0$ saranno quelle mostrate in figura 1.41.

Vediamo che la frequenza di Q_0 é la metá di quella del clock, quella di Q_1 un quarto etc. Inoltre notiamo che la sequenza dei bit $(Q_3Q_2Q_1Q_0)$ inizia con (0,0,0,0) poi (0,0,0,1), diviene poi (0,0,1,0), (0,0,1,1) etc, cioé "conta" gli impulsi di clock in arrivo al primo FF. Ovviamente, dopo il 15^{mo} impulso di clock il contatore si "resetta", cioé le uscite saranno (0,0,0,0), per poi ricominciare a crescere. Se colleghiamo ciascuna delle uscite ad un LED potremmo vedere il conteggio (binario) del numero di impulsi di clock ricevuto dal FF3.

Il contatore, con una piccola modifica, puó 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. Ció é facile da verificare.

E' possibile realizzare anche una catena di FF che, con un opportuno comando di controllo, possa contare in entrambi i versi. E' sufficiente collegare l'uscita di un

FF al clock del successivo come mostrato in figura.

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, sará abilitata (notare l'invertitore) la porta B e disabilitata la A. Il contatore eseguirá il ciclo in ordine decrescente. E' possibile modificare il collegamento mostrato nella figura a pag. 64, in modo da realizzare un contatore che, anziché resettarsi ogni 16 impulsi, si resetti ogni 10. Si ottiene in tal modo quello che é noto come un "contatore decimale" o Binary Coded Decimal (BCD). La modifica é illustrata nella figura che segue

dove, come nel caso precedente, tutti i terminali J e K sono posti al livello logico alto. La presenza della porta AND, con ingressi su Q1 e Q3 e con uscita collegata al Clear, fa sí che il contatore si resetti non appena Q1 = Q3 = 1. Come si puó vedere dalla figura a pag. 64, ció accade dopo il 10 impulso di clock. Si provi come esercizio a disegnare un contatore che si resetta dopo 3 impulsi di clock (contatore "modulo 3"). I contatori fin qui esaminati sono noti come "contatori asincroni". Ció 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) del primo, ed analogamente per i successivi. Ció costituisce uno svantaggio, poiché il tempo di propagazione é lungo. Se tutti i FF sono nello stato logico 1, tale tempo é massimo, poiché il successivo intervallo di clock manda il primo FF da 1 0, questo a sua volta manda il secondo da 1 0 (con un ritardo), il secondo manda il terzo da 1 0 (con uguale ritardo) e cosí via. Il ritardo complessivo sará in tal caso N , se N é il numero di FF della catena. Per ridurre il tempo di propagazione conviene adottare uno schema in cui tutti i FF vengono "triggerati" simultaneamente dal clock. Tale tipo di contatore va sotto il nome di "contatore sincrono". Esaminiamo un esempio.

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, ultimo FF nella catena) commuta se é verificata la condizione A = B = C = 1. Come possiamo facilmente verificare (vedasi la tabella alla pag. 68) la presenza della parte AND soddisfa la logica insita nella successione dei numeri binari crescente (0 0 0 0), (0 0 0 1), (0 0 1 0), (0 0 1 1), ... 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 di A valeva 1 c) il terzo FF deve commutare in corrispondenza ad un impulso di clock se sia l'uscita A che l'uscita B volevano 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 clock n 15 il contatore si resetta automaticamente e ricomincia il conteggio. I contatori sincroni offrono vantaggi di velocitá rispetto a quelli asincroni esaminati in precedenza, anche se un ritardo aggiuntivo é causato dalla parte NAND. Un contatore sincrono binario a 4 bit, di uso abbastanza generale é il 74LS191. Esso puó eseguire il conteggio in avanti o indietro a seconda del valore logico presente sull'ingresso di controllo "up/down". Il 74LS190 é praticamente equivalente al 74LS191, con la sola differenza che esso é predisposto per il conteggio decimale (BCD). E' possibile, collegando due o piú contatori n serie, realizzare conteggi fino a potenze piú elevate di 2 (o di 10). Ad esempio, collegando in serie tre contatori

BCD 74LS190 é possibile, come mostrato in figura, 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) compare 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. Cosí, quando il primo FF (da sinistra) sia passato dal 9 allo 0, il secondo inizia a contare. Poiché peró l'Enable ha la durata uguale a quella del livello ”basso” del clock d'ingresso, esso conterá solo fino ad 1, mentre il primo contatore ripartirá da 0, arrivando fino a 9 e poi a 0. A questo punto il primo FF genera un nuovo RC e quindi un'Enable ad secondo FF, che ora passerá a 2 e cosí di seguito. Quando anche il secondo FF sia arrivato a 9 e poi a 0, esso genererá un RC e quindi invierá un'Enable al terzo FF, che inizierá a contare la centinaia.

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

Abbiamo giá visto come una catena di FF possa essere utilizzata come divisore di frequenza. Un contatore puó 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 sará sufficiente far uso di un dispositivo (un diodo emettitore di luce-LED- ad una cellula fotoelettrica) per generare un impulso ogni volta che la luce emessa dal LED é interrotta dal passaggio di uno degli oggetti e non arriva quindi alla cellula fotoelettrica. Un contatore opportuno incrementerá di un'unitá il conteggio ogni volta che gli arrivi un impulso. Un contatore separato conterá il numero di impulsi che gli pervengono, generati da un opportuno oscillatore (clock) previamente calibrato, che fornirá l'informazione temporale. Possiamo utilizzare un dispositivo simile per misurare l'intervallo di tempo tra due eventi. La figura illustra un sistema di questo tipo. 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 temo tA dal passaggio dell'oggetto davanti al traguardi A ed un secondo al tempo tB in cui l'oggetto passa davanti al traguardo B.

L'impulso tA sia inviato all'ingresso di Set di un FF di tipo RS (in cui era inizialmente Q=0, =1) mentre l'impulso generato all'istante tB sia inviato all'ingresso di Reset. Il gate AND che segue é collegato a valle ad un contatore (opportunamente calibrato), mentre dei due ingressi, uno é collegato al Q del DD e l'altro riceve una sequenza di impulsi di clock, di frequenza nota. Il funzionamento del circuito é il seguente: quando l'impulso generato al tempo tA arriva al FF, questo avrá S=1, R=0 e quindi Q=1. Subito dopo sará S=R=0 e quindi lo stato del FF rimane invariato fino al tempo tB, quando R diventa 1 e quindi Q va a 0. Nell'intervallo di tempo tB - tA in cui Q=1, la porta AND é 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 tB-tA. Il sistema andrá ovviamente calibrato preventivamente. Un dubbio che puó venire é il seguente. Sia l'impulso generato al tempo tA che quello generato al tempo tB 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? Ció é semplice se sostituiamo il FF RS con un JK in cui J=K=1, come mostrato in figura

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 tB manda Q 0 e 1. Un dispositivo molto simile a questo puó essere adoperato se si vuole misurare una frequenza, cioé il numero di impulsi/sec. Il dispositivo é mostrato in figura.

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 di clock ottenuto demoltiplicando per un fattore 106 un segnale di 1 MHZ. In tale circostanza Q=1 per un tempo di 1S. La porta AND sará quindi abilitato per 1 s ed il contatore conterá il numero di impulsi che in tale tempo gli arrivano, cioé la frequenza voluta. Naturalmente occorrerá aggiungere qualcosa al circuito per far sí che alla fine del secondo, il sistema si arresti e consenta la lettura del contatore. Anche se non specificamente relativo ad applicazioni in esperimenti di fisica, incontra ampio utilizzo il contatore decimale illustrato a pag. 66. Ad esempio, nel contatore BCD (Binary Coded Decimal) il numero é espresso come una sequenza di cifre decimali. Ad esempio, il numero 3702 é memorizzato come

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 Ser 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 che segue. L'integrato 74LS93, che consente il conteggio fino a 16, puó essere azzerato, come si é visto, in corrispondenza della cifra binaria corrispondente al n 9. E' peró piú conveniente usare un contatore predisposto per contare in base 10, come il 74LS90.

Nella tabella dei codici ASCII mostrata in figura 1.42, 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 (dove 16). Come visto, in tale rappresentazione, un numero e.g. di 2 bytes (16 bit) é rappresentato da una stringa di 4 carattere esadecimali. Ad esempio, il numero binario: 1010 1101 0110 0011 é rappresentato in esadecimale da: AD63.

1.18 Decodificatori e Display

Per visualizzare un numero binario abbiamo varie alternative. La prima é quella, ossia, di visualizzare ogni singolo bit mediante un LED. Se il LED é accesso il bit corrispondente vale 1, se esso é spento vale 0. Tale soluzione non é pratica per la visualizzazione di numeri con multi 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) é possibile utilizzare il chip 74LS42, che é integrato a 16 pin. Il

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 1.42:

display puó essere effettuato utilizzando dei "display a 7 segmenti", in cui ciascun numero BCD (tra 0 e 9) corrisponde l'accensione degli opportuni segmenti luminosi (LED), come mostrato in figura:

L'identificazione dei segmenti é quella mostrata in corrispondenza dell'8. Il decodificatore BCD a sette segmenti 74LS247, mostrato in figura, é un decoder/driver adatto a comandare i segmenti di un LED. Esso contiene sia la circuitaria che fornisce la decodifica dei quattro bit d'ingresso nelle 7 linee d'uscita, sia quella di alimentazione dei singoli segmenti.

La resistenza da 270 sulle 7 linee d'uscita sono state inserite per limitare la corrente erogata. L'indicatore puó essere spento mandando all'ingresso del 74LS247 la cifra binaria 1111 o, alternativamente, portando a 0 l'ingresso di blauking (pin 4). La decodifica per gruppi di 4 dal binario all'esadecimale puó essere effettuata utilizzando l'integrato 74LS154, che ha 16 uscite, corrispondenti ciascuna ad uno dei 16 simboli da 0 a 15 (F esadecimale). Ciascuna delle 16 uscite puó 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 é 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 (cioé da 0 ad F). Questo display sará quello adoperato in alcune delle esperienze di laboratorio. Le caratteristiche di tale integrato sono mostrate qui di seguito.

La figura che segue mostra uno schema a blocchi del TIL 31. Notiamo la presenza di un "Latch" in ingresso. Questo é essenzialmente un insieme di 4 FF di tipo D. Questo ha lo scopo di "mantenere" i dati presentati in ingresso, per il tempo necessario alla decodifica ed al successivo display a lettura. Piú precisamente, finché il segnale presente sul piedino 5 é basso, l'uscita del Latch seguono l'ingresso (piedini 3, 2, 13, 12). Quando il segnale é alto, le uscite non cambiano piú. 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 (74LS04) indicati in figura.

1.19 Moltiplicazione e divisione in binario

L'operazione di moltiplicazione in binario tra due bit equivale all'AND di essi:

La moltiplicazione tra numeri di piú bit puó essere effettuata con una semplice estrapolazione dell'analogo processo nel sistema decimale. Mostriamo alcuni esempi.

Vediamo che ciascun bit del secondo numero (a partire dalla destra) deve essere moltiplicato per i bit del primo. A tali "moltiplicazioni" potranno provvedere delle porte AND. Il risultato ottenuto dalla moltiplicazione del primo bit andrá sommato al risultato della moltiplicazione del secondo, shiftato a sinistra di una posizione; questo andrá sommato al risultato della moltiplicazione del terzo bit, shiftato a sinistra di due posizioni ... etc. Vediamo cosí che un "moltiplicatore" puó esser ottenuto effettuando sequenzialmente degli AND, degli shift e delle somme. Il circuito mostrato in figura illustra una possibile implementazione di un moltiplicatore di due numeri a 3 bit. Per il risultato della moltiplicazione é ovviamente necessario un registro a 6 bit. Il circuito comprende uno Shift-Register 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 azzera il registro accumulatore, si caricano i tre bit del moltiplicatore nel relativo registro (con Q0 corrispondente al bit meno significativo) ed i tre bit del moltiplicando nei registri Q0-Q2 corrispondenti. Le uscite delle cinque porte NAND contengono a questo punto i prodotti del bit meno significativo del moltiplicatore per i tre bit del moltiplicando. Piú 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 Q0 del moltiplicatore per il bit Q1 del moltiplicando, etc. Le porte a e b sono per il momento al livello logico 0. I prodotti parziali cosí ottenuti sono ora disponibili agli ingressi del sommatore, in B0, B1, B2. Essi sono anche disponibili all'uscita del sommatore (S0, S1, S2) essendo gli altri ingressi (A0, A1, A2) per il momento nulli. Al primo impulso di clock, il prodotto parziale (S0, S1, S2) viene trasferito nei tre FF all'estrema destra del registro accumulatore e contemporaneamente agli ingressi (A0, A1, A2) del sommatore. Il medesimo impulso di clock agisce sul registro moltiplicatore e sul registro moltiplicando nel modo seguente: 1) Il secondo bit del registro moltiplicatore 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 é a livello logico 0. E' facile vedere che ora l'uscita della porta AND indicata con é 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, o tre bit ottenuti sono presenti agli ingressi (B1, B2, B3) del sommatore e quindi risultano giá shiftati a sinistra di una posizione, che é proprio quel che si fa nella moltiplicazione binaria. Ora all'uscita del sommatore comparirá, in (S0, S1, S2, S3, S4) la somma della prima moltiplicazione e della seconda. Il successivo impulso di clock trasferisce tali uscite nei corrispondenti FF dell'accumulatore e quindi degli ingressi (A0, ..., A4) del sommatore. Il medesimo impulso di clock shift il bit piú significativo del registro moltiplicatore (che era a questo punto in FF1) nel registro FF0 e simultaneamente shifta 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 (B2, B3, B4) del sommatore. Questo fornirá ora alle uscite (S0 - S5) la somma finale richiesta é ora disponibile all'uscita del sommatore e sará 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 peró n commercio dei moltiplicatori che effettuano le operazioni in parallelo, con un notevole guadagno in velocitá.

1.20 La divisione in binario

L'idea base di un circuito digitale che effettua l'operazione di divisione tra due numeri interi (positivi) puó 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 possiamo dire che la parte intera del numero che esprime il rapporto é pari al numero delle sottrazioni effettuato, cioé 4. Sottraiamo ora 0.7 da 5, tante volte quante ne occorrono per ottenere un resto inferiore a 0.7:

Ne deduciamo che la prima cifra decimale é 7. 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 puó 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 = La prima sottrazione, effettuata con il metodo del complemento a 2, fornisce:

Abbiamo cioé un bit di riporto (o di overflow), il quale indica che la differenza (uguale a 1010) é positiva. Sottraendo ancora abbiamo:

Abbiamo ancora una differenza positiva (come evidenziato dalla presenza del bit di riporto) uguale a . Un'ulteriore sottrazione fornisce:

cioé un numero positivo (2D). Successivamente si ottiene: Vediamo che ora il bit di riporto é 0, cioé la differenza é negativa. La parte intera del rapporto desiderato é pari al numero di volte che la differenza ha fornito un bit di riporto uguale ad 1, cioé 3D. Un circuito che realizza la funzione desiderata é quello mostrato nella figura che segue. 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 Il dividendo é caricato inizialmente nel registro.

Dopo che la prima somma sará stata effettuata, il riporto C3=1, mentre i registri Q3, Q2, Q1, Q0 contrarranno il risultato della differenza tra dividendo e divisore. Poiché C3=1, il prossimo impulso di clock fará avanzare il contatore di una unitá. Ora il sommatore effettuerá la differenza tra il nuovo contenuto del registro ed il divisore. Se C3 é ancora uguale ad 1, la porta AND sará ancora abilitata ed il contatore avanzerá di una unitá al prossimo impulso di clock. Il processo si ripete fino a quando C3 non sia divenuto uguale a 0, al qual punto il contatore si arresta.

Indice