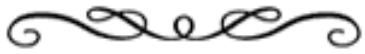
Dispense di elettronica digitale per il corso di LAB 2

Prof. Flavio Fontanelli

Versione 1.4 — 29 marzo 2012



NOTA Questo non è (ancora) un libro di testo

Scopo di queste note è fornire una introduzione relativamente semplice all'elettronica digitale ed alle sue applicazioni alla fisica. Le informazioni sono preliminari, parziali, in sviluppo, probabilmente ci sono inesattezze e punti poco chiari, spero vi servano come guida alla preparazione dell'esame, nulla di più.
Sono reperibili all'indirizzo:
`http://www.ge.infn.it/~fontanel/index.html`

Indice

Capitolo 1

Algebra booleana

Storicamente l'elettronica digitale è nata dalla esigenza di costruire un sistema in grado di eseguire calcoli in modo veloce ed affidabile, in realtà è sotto gli occhi di tutti come il raggiungimento di questo obbiettivo abbia poi permesso di sfruttare le potenzialità del sistema per ottenerne infiniti altri, infatti non esiste oggi elettrodomestico, mezzo di trasporto, strumento scientifico di misura, strumento di diagnostica, ecc. che non sia controllato da un microcalcolatore per ottimizzarne il funzionamento. È pertanto a questo tipo di problematiche legate all'hardware del calcolatore che siamo principalmente interessati, lasciando ad altri corsi i problemi, pure importanti, legati al software.
Il primo problema che occorre risolvere per costruire un sistema di elaborazione in grado di lavorare direttamente su grandezze numeriche è capire come rappresentarle efficientemente.
Si potrebbe pensare di continuare ad usare il sistema di numerazione decimale a tutti ben noto, ad esempio si potrebbero rappresentare i simboli da 0 a 9 con 10 differenti valori di tensione e costruire macchine che siano in grado di operare coerentemente con tali valori di tensione.
Tale sistema sarebbe pero' poco affidabile e complicato a causa della necessità di distinguere rapidamente valori di tensione poco differenti l'uno dall'altro (i componenti fisici invecchiando si modificano alterando il loro comportamento), inoltre un sistema in grado di lavorare con 10 diversi livelli di tensione sarebbe ragionevolmente più complicato di uno capace di distinguerne e trattarne un numero minore. Poichè ciò che ci interessa è la costruzione di un sistema il piu' efficiente ed economico possibile dobbiamo valutare quale tipo di rappresentazione dei numeri ci convenga.

1.1 La rappresentazione dei numeri interi

Un numero intero viene rappresentato da una sequenza di cifre dove la posizione relativa indica il "peso" della cifra, ad esempio 197 indica il numero ottenuto sommando $1 \cdot 10^2 + 9 \cdot 10^1 + 7 \cdot 10^0$, ossia come una somma di potenze di 10 moltiplicate per numeri interi compresi fra 0 e 9.
Possiamo naturalmente usare anche altre "basi", ad esempio, se usiamo la base 8 possiamo scrivere $197 = 3 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0$.

Esercizio 1 *Scrivete 197 usando la base 2.* *(R. 11000101)*

Esercizio 2 *A quale numero corrisponde* 1001_2 *?* *(R. 9)*

Come si vede dall'esercizio precedente se si usa la base 2 si hanno a disposizione 2 soli simboli e la notazione si appesantisce parecchio, la rappresentazione di 197 ha bisogno di 8 simboli invece che di 3.
Una macchina che operasse con numeri espressi in base 2 dovrebbe essere capace di distinguere solo 2 simboli diversi (0 e 1) e quindi sarebbe costituita da molti dispositivi semplici, ognuno dei quali si occuperebbe solo delle

somme 0+0, 0+1, 1+1 con riporto). Si pone perciò il problema di scegliere la base ottimale (in base a criteri che potrebbero essere il costo e l'affidabilità).
Se supponiamo che la complessità ed il costo di un dispositivo siano proporzionali alla base utilizzata (per cui, ad esempio un sommatore elementare in base 10 sarebbe 5 volte piu' complicato dell'analogo sommatore in base 2) si puo' dimostrare che la base più conveniente sarebbe la base 3, questo perche' il sistema userebbe molti sommatori elementari, ognuno dei quali relativamente poco complicato.
L'ipotesi che un dispositivo numerico che lavori in base 3 sia solo del 50% più complicato di uno capace di lavorare in base 2 non è realistica e ciò porta alla conclusione che i sistemi ottimali per il trattamento dei dati numerici devono utilizzare la base 2.
In questa conclusione siamo confortati anche dal fatto che la natura ha scelto la base 2 per la costruzione del nostro sistema nervoso (i neuroni hanno 2 soli stati: normale ed eccitato). Per i dettagli della precedente dimostrazione vedi [5], vol. 2.

1.2 L'algebra booleana

È opportuno, giunti a questo punto, trovare un attrezzo che ci permetta di formalizzare la manipolazione di questi 2 simboli 0 e 1. Ci viene in soccorso un'algebra particolare, inventata da Boole nell'800[1], in tale algebra le grandezze possono assumere solo 2 valori, (solitamente chiamati 0 e 1, oppure falso e vero, che nulla hanno a che fare con i numeri corrispondenti). Alle grandezze dell'algebra booleana vengono attribuite proprietà utili a modellare le operazioni che sui simboli 0 e 1 le macchine dovranno effettuare per agire su numeri rappresentati in base 2.
Sugli elementi dell'algebra booleana si può agire con 3 operatori: OR e AND e NOT (detta anche negazione) che noi indicheremo con $+$, $\cdot$ e $\overline{\ }$. Anche se utilizziamo i 2 simboli delle 2 operazioni dell'algebra elementare, le operazioni che faremo su di essi nulla hanno a che fare con esse. Vediamo quindi gli assiomi dell'algebra di Boole, cercando di limitare la formalizzazione allo stretto necessario. Sia x una variabile booleana, allora, per ogni x vale:

1. $x + 0 = x$
2. $x + 1 = 1$
3. $x + x = x$
4. $x + \overline{x} = 1$
5. $x_1 + x_2 = x_2 + x_1$ Proprietà Commutativa dell'OR
6. $(x_1 + x_2) + x_3 = x_1 + (x_2 + x_3)$ Proprietà Associativa dell'OR
7. $(x_1 \cdot x_2) + (x_1 \cdot x_3) = x_1 \cdot (x_2 + x_3)$ Proprietà Distributiva dell'OR
8. $x \cdot 1 = x$
9. $x \cdot 0 = 0$
10. $x \cdot x = x$
11. $x \cdot \overline{x} = 0$
12. $x_1 \cdot x_2 = x_2 \cdot x_1$ Proprietà Commutativa dell'AND
13. $(x_1 \cdot x_2) \cdot x_3 = x_1 \cdot (x_2 \cdot x_3)$ Proprietà Associativa dell'AND
14. $(x_1 + x_2) \cdot (x_1 + x_3) = x_1 + (x_2 \cdot x_3)$

[1] Quest'algebra fu proposta dal matematico inglese George Boole nel 1854 nell'opera "An investigation of the laws of thought on which are founded the mathematic theories of logic and probabilities".

Si noti che l'ultima proprietà elencata non vale nell'algebra elementare. Da questi assiomi è possibile dedurre i seguenti teoremi:

1. $\overline{(\overline{x})} = x$
2. $\overline{0} = 1$ e similmente $\overline{1} = 0$
3. Teorema di dualità:
   se in una identità booleana si scambiano gli zeri con gli 1 e gli AND con gli OR si ottiene un'altra identità booleana.
4. Teoremi di De Morgan:
   $\overline{x} + \overline{y} = \overline{(x \cdot y)}$
   che applicando il precedente teorema di dualità puo' essere riscritta anche così: (verificare!)
   $\overline{x} \cdot \overline{y} = \overline{(x + y)}$

È utile porre in relazione l'algebra di Boole con la teoria degli insiemi. Consideriamo una porzione del piano e consideriamo tutti i possibili sottoinsiemi della porzione del piano, identifichiamo il simbolo 1 con tutta la porzione di piano prescelta, lo zero con l'insieme vuoto, l'AND e OR con le usuali operazioni di intersezione ed unione dell'algebra degli insiemi ed il NOT con il complemento, allora tutte le precedenti proprietà possono essere facilmente visualizzate, è sufficiente associare ad ogni variabile booleana un'area arbitraria contenuta nel quadrato (il quale verrà identificato con il simbolo 1). In questo modo è ad esempio possibile visualizzare il teorema di De Morgan e convincersi della sua validità (provateci!).
È anche possibile utilizzare gli assiomi dell'algebra booleana per formalizzare la manipolazione delle espressioni logiche verbali, in questo caso ad ogni espressione logica falsa faremo corrispondere il simbolo 0, ad ogni espressione vera faremo corrispondere il simbolo 1, allora potremo per esempio considerare frasi del tipo "Oggi vado a Milano", "Oggi vado a Pavia" e poi combinarle attraverso le congiunzioni AND e OR per ottenere altre espressioni sulla cui verità o falsità è possibile decidere conoscendo quella delle espressioni di partenza, ad esempio potremo costruire frasi del tipo: "Oggi vado a Milano oppure (OR) oggi vado a Pavia" od espressioni un po' più complicate, come la seguente che potrebbe essere utilizzata come base per la progettazione di un allarme: "L'allarme deve suonare se (una portiera viene aperta e (AND) la chiave di accensione non viene inserita) o (OR) viene rotto un finestrino".
Nel nostro caso però utilizzeremo l'algebra booleana per manipolare le cifre dell'aritmetica binaria oppure, più in generale, per lavorare su segnali elettrici che possano assumere solo 2 valori come quelli che derivano dal confronto fra 2 grandezze fisiche, ad esempio un termostato che confrontando la temperatura dell'ambiente con quella impostata decide se accendere o no la stufa.
L'effetto degli operatori AND, OR e NOT sulle variabili booleane può essere facilmente visualizzato in forma tabellare:

z = x AND y

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

z = x OR y

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

y = NOT x

x	y
0	1
1	0

Queste tabelle vengono solitamente chiamate **tabelle di verità**.
In modo analogo vengono definiti AND e OR su un numero qualunque di variabili (la funzione AND risulta vera se e solo se tutti gli argomenti sono veri, l'OR è vero se almeno un argomento è vero).

Esercizio 3 *Scrivere esplicitamente la tabella di verità per AND e OR a 3 variabili*

È possibile definire anche funzioni derivate da queste, ad esempio la funzione NAND (ossia NOT AND) che corrisponde all'applicazione dell'AND e successivamente del NOT, similmente esiste il NOR (NOT OR)

Esercizio 4 *Scrivere esplicitamente la tabella di verità per un NAND a 3 variabili*

Si noti che, grazie al teorema di De Morgan, l'AND e l'OR non sono entrambi indispensabili, l'algebra booleana potrebbe essere costruita usando il solo NAND.

Esercizio 5 *Scrivere espressioni equivalenti a $C = A + B, D = A \cdot B$ usando il solo operatore NAND.*

Gli operatori elementari AND, OR, NOT possono essere combinati variamente costruendo espressioni algebriche simili a tutti gli effetti a quelle a cui siamo abituati nell'algebra elementare, siamo cioè spinti a definire **funzioni** booleane, ad es. $z = f(a, b, c) = \overline{a + c} + bc$.
Equivalentemente la stessa funzione potrà essere descritta da una opportuna tabella di verità:

z= f(abc)

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Esercizio 6 *Verificare l'equivalenza della tabella precedente con l'espressione algebrica.*

Si pone adesso il problema di passare dalla descrizione tabellare a quella algebrica e viceversa. Il passaggio dalla formula alla tabella è banale, come avrete compreso risolvendo l'esercizio precedente, bisogna solo elencare tutti i valori possibili delle variabili indipendenti e pazientemente calcolare le funzioni booleane richieste, con un po' di pratica non ci sono problemi.
Il passaggio inverso è invece piu' complesso, cominciamo definendo le forme canoniche: una funzione booleana può sempre essere espressa come "somma di prodotti" (forma SP) o come "prodotto di somme" (forma PS), vediamo cosa ciò vuol dire: nella forma SP una funzione puo' essere scritta come un or di molti termini elementari ognuno dei quali contiene le variabili indipendenti, eventualmente negate, legate dall'operazione AND, ad esempio: $w = a\,\overline{b} + \overline{a}\,b$, similmente nella forma PS la funzione è espressa come un AND di molti termini elementari ognuno dei quali contiene le variabili indipendenti, eventualmente negate, legate dall'operazione OR, ad esempio: $u = (a + b) \cdot (\overline{a} + \overline{b})$
Le due espressioni precedenti sono equivalenti a tutti gli effetti, il passaggio da una all'altra richiede solo un po' di manipolazioni algebriche (piuttosto noiose...) in cui l'uso del teorema di De Morgan è spesso fondamentale.
Esempio: $(A + B)(\overline{A} + \overline{B}) = A\overline{A} + A\overline{B} + B\overline{A} + B\overline{B} = A\overline{B} + B\overline{A}$

Esercizio 7 *perchè ho potuto cancellare i termini $A\overline{A}$ e $B\overline{B}$?*

Una delle 2 forme, a seconda dei casi, può essere più o meno conveniente.
L'algoritmo per passare dalla forma tabellare alla forma canonica SP può essere descritto in questo modo, non molto formale ma si spera convincente (per una dimostrazione rigorosa si veda [1]):
Si noti per cominciare che la funzione AND fornisce come risultato falso a meno che tutti i suoi argomenti non siano veri.

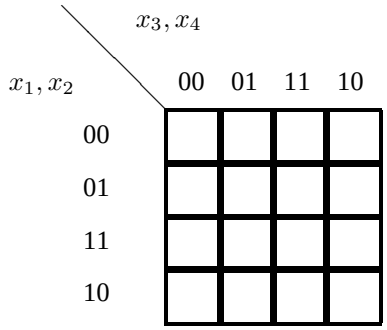
È allora facile scrivere una funzione che sia vera per una sola, specifica combinazione dei suoi argomenti: è sufficiente mettere in AND le variabili stesse, negandole se il loro valore era falso, ad esempio sia $z = z(a, b, c)$ con z sempre falsa, eccettuato il solo caso a=0, b=1, c=0, allora la funzione richiesta si ottiene ponendo: $z = \overline{a} \cdot b \cdot \overline{c}$.
Da questa constatazione si parte per scrivere la funzione usando la forma algebrica SP: si scorre la tabella di verità e si isolano le righe per cui la funzione è vera, per ogni riga si scrive il termine corrispondente come appena spiegato, successivamente si collegano i termini ottenuti con l'operatore OR. Infatti l'OR di n termini è vero se almeno uno dei termini è vero e quindi l'OR dei termini costruiti come spiegato produrrà un valore vero solo dove desiderato.
Esempio: Si consideri la seguente funzione (il cosiddetto OR ESCLUSIVO)

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

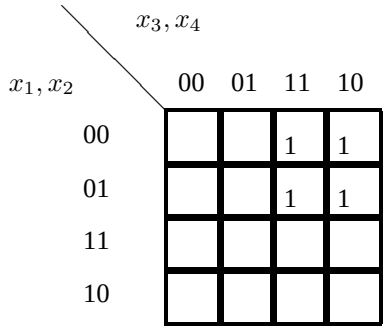
Guardando la tabella ci accorgiamo che solo la seconda e la terza riga sono vere, alla seconda riga corrisponde $\overline{x} \cdot y$, alla terza riga corrisponde $x \cdot \overline{y}$, facendo l'or dei 2 termini si ottiene $z = \overline{x} \cdot y + x \cdot \overline{y}$
Incidentalmente è utile sapere che i singoli termini che realizzano la funzione in forma SP sono spesso chiamati mintermini.
Vediamo ora il metodo utilizzato per realizzare la funzione in forma PS.
Mentre prima si era scritta la funzione come somma dei termini il cui valore era vero solo per un certo valore delle variabili, adesso scriveremo la funzione come prodotto di termini, ognuno dei quali si annullerà in un solo caso. Il ragionamento è assolutamente parallelo a quello utilizzato per la forma SP.
Si parte dalla constatazione che l'OR permette di costruire una funzione sempre vera tranne il caso in cui tutte le variabili siano false, è perciò possibile, negando le variabili selettivamente, ottenere una funzione nulla per un ben determinato valore delle variabili, basterà sostituire ai valori delle variabili nella tabella di verità, agli zeri la variabile e agli uni la variabile negata. Ad esempio se vogliamo una funzione $z = z(abc)$ sempre uguale a 1, tranne il caso $a = 0, b = 1, c = 0$ basterà porre $z = a + \overline{b} + c$.
Adesso scorreremo la tabella cercando le righe per cui il valore della funzione è falso e ne scriveremo i termini relativi nel modo su indicato, come ultimo passo collegheremo i termini trovati con l'operatore AND. I singoli fattori di una espressione PS vengono chiamati *maxtermini*. Ad esempio la funzione OR ESCLUSIVO della pagina precedente, usando la forma PS, diventa $z = (x + y) \cdot (\overline{xy})$.
Questi 2 algoritmi conducono a 2 forme in tutto e per tutto equivalenti ma non necessariamente ugualmente comode, se, ad esempio, prendiamo la tabella di verità dell'AND e, facendo finta di non conoscere il risultato, applichiamo l'algoritmo per il calcolo della forma canonica PS, otteniamo $z = (a+b)(\overline{a}+b)(a+\overline{b})$, tragicamente più lunga della espressione ottenuta usando l'algoritmo per la forma canonica SP: $z = ab$.
Detto in altri termini questi 2 algoritmi permettono di ricavare si un'espressione algebrica che realizza la funzione desiderata, ma di solito la forma ottenuta deve essere "minimizzata". La minimizzazione (semplificazione) delle funzioni booleane può essere ottenuta in diversi modi, come nell'algebra elementare si possono tentare delle semplificazioni sfruttando le proprietà dell'algebra e il teorema di De Morgan, ma questo è un procedimento molto legato all'abilità della persona e soggetto ad errori manuali, si preferiscono perciò di solito metodi formali molto efficaci, i 2 procedimenti più diffusi sono quello di Quine–McCluskey e quello delle mappe di Karnaugh.
Il primo metodo è più adatto ad essere programmato su un calcolatore permettendo la semplificazione di funzioni booleane arbitrariamente complicate in breve tempo, gli interessati lo possono trovare sul [1], il secondo, molto semplice, è adatto alla minimizzazione di funzioni di non più di 6 variabili e lo analizzeremo nel dettaglio.

1.3 Le mappe di Karnaugh

I metodi per la minimizzazione delle espressioni booleane si basano sulla seguente proprietà: $fx + f\overline{x} = f$. Il metodo delle mappe di Karnaugh (vedi [1]) e' un metodo grafico facilmente utilizzabile per minimizzare espressioni con non più di 6 variabili. Si sviluppa in 2 fasi: la compilazione delle mappe e la loro "copertura".
Prima fase: la tabella di verità viene riscritta in forma di matrice, ad esempio la mappa relativa ad una funzione di 4 variabili $f(x_1, x_2, x_3, x_4)$ ha il sequente aspetto:



Ad ogni casella corrisponde un valore della funzione booleana, ad esempio la casella in alto a sinistra corrisponde al valore $f(0,0,0,0)$, mentre la casella in alto a destra corrisponde al valore $f(0,0,1,0)$. Se le funzioni hanno un numero inferiore di argomenti la tabella viene ridotta opportunamente, ad esempio una funzione di 2 variabili corrisponde ad una matrice 2X2 ed una di 3 variabili ad una matrice 4X2 (o 2X4, dipende da come ordinate le variabili). Se invece le variabili sono 5 avremo una prima tabella 4X4 per $x_5 = 0$ ed una seconda per $x_5 = 1$, se le variabili fossero 6, avremo una prima tabella per $x_5 = 0, x_6 = 0$, una seconda per $x_5 = 0, x_6 = 1$, una terza $x_5 = 1, x_6 = 1$ ed una quarta per $x_5 = 1, x_6 = 0$. È essenziale notare un dettaglio: le variabili sono sempre disposte a coppie: 00, 01, 11, 10, notare l'inversione rispetto all'ordine binario consueto, il terzo elemento è 11, non 10. C'è una ragione profonda in questo dettaglio, guardando le mappe si vede che 2 caselle contigue in orizzontale o in verticale sono distinte per il cambio di valore di una sola variabile. È invece totalmente irrilevante l'ordine utilizzato per riportare le variabili sugli assi e la scelta dell'asse, ad esempio, una funzione di 4 variabili A, B, C, D potrà indifferentemente essere rappresentata da una mappa avente AB sull'asse orizzontale e C e D su quello verticale oppure da una mappa avente ad esempio le variabili C ed A sull'asse verticale e B e D su quello orizzontale.
Supponiamo ad esempio di trovarci di fronte alla seguente mappa:



Nella mappa ci sono 4 termini uguali a 1 (le caselle vuote sono poste a 0) che corrispondono ai termini (da sinistra a destra, dall'alto in basso) : $\overline{x_1}\,\overline{x_2}\,x_3x_4 + \overline{x_1}\,\overline{x_2}\,x_3\overline{x_4} + \overline{x_1}x_2x_3x_4 + \overline{x_1}x_2x_3\overline{x_4}$
Se adesso usiamo la proprietà $fx + f\overline{x} = f$ dapprima sui primi 2 termini e successivamente sul terzo e quarto dove x_4 appare nella forma normale ed in quella negata otteniamo che la precedente equazione è in realtà equivalente

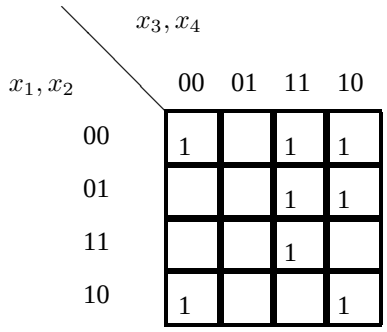
a: $\overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2x_3$, nuovamente possiamo notare che i 2 termini differiscono solo per il termine in x_2, per cui l'espressione in realtà può essere ridotta a $\overline{x_1}x_3$.
Possiamo perciò affermare che tutte le volte che in una mappa di Karnaugh ci sono 2, 4, 8, 16, 32, 64 caselle contigue, disposte a rettangolo, contenenti tutti 1, le possiamo conglobare in un unico termine contenente solo le variabili che non mutano valore. Nell'esempio precedente infatti possiamo dire, guardando le coppie di zeri e uni sugli assi che solo x_1 e x_3 non cambiano valore, in particolare x_0 appare col valore 0, quindi corrisponde alla variabile negata.
Notiamo ancora una cosa: le caselle presenti ai bordi sinistro e destro oppure alto e basso della mappa sono in realtà contigue, nel senso che una sola variabile cambia valore, perciò possono essere raccolte e minimizzate, da un punto di vista topologico dobbiamo pensare che la mappa ha gli estremi coincidenti.
Ad esempio la mappa:

x_1, x_2 \ x_3, x_4	00	01	11	10
00	1			1
01				
11				
10	1			1

corrisponde al termine $\overline{x_2}\ \overline{x_4}$. Ci si convince di questo fatto pensando che l'ordine delle variabili lungo gli assi è arbitrario, una permutazione dell'ordine delle variabili pertanto sposterebbe le variabili dai bordi al centro.
Per poter tenere conto di tutte le possibilità le mappe di 5 o 6 variabili diventano tridimensionali e nel fare le minimizzazioni bisogna cercare gruppi di 1 contigui in tutte e 3 le dimensioni, ad esempio, supponiamo di avere una funzione di 6 variabili Z=Z(A,B,C,D,E,F), per rappresentarla avremo bisogno di 4 mappe 4X4, la prima presenterà i primi 16 valori della tavola di verità, ossia A e B fissi a 0, C,D,E,F che assumono tutti valori possibili, la seconda mappa conterrà i valori della funzione per A=0, B=1, la terza i valori per A=1, B=1, la quarta infine i valori per A=1, B=0. Si noti il solito ordine per i valori da A e B: 00,01,11,10.
Incidentalmente a questo punto è chiaro perche' il metodo funziona fino ad un massimo di 6 variabili, con 7 variabili avremmo bisogno di matrici quadridimensionali . . .
Una volta individuati i blocchi di 1 necessari a "coprire" la mappa, cioè individuato un insieme di blocchi sufficiente a contenere tutti gli 1 **(eventualmente anche con sovrapposizioni)** è necessario scrivere i termini corrispondenti per completare il processo di minimizzazione della funzione ottenendo il risultato espresso nella forma canonica SP.
Per minimizzare una espressione booleana ed ottenerne la forma canonica PS occorre procedere in modo simmetrico, ossia bisognerà

1. isolare nelle mappe i gruppi di **zeri** contigui sufficienti a coprire la mappa.
2. individuare le variabili che non cambiano valore e determinare se sugli assi hanno il valore 0 oppure 1
3. far corrispondere agli zeri sugli assi le variabili ed agli 1 le variabili negate (contrariamente a prima)
4. legare le variabili all'interno del singolo blocco col segno + (OR)
5. mettere in AND i singoli termini trovati.

Ad esempio la mappa:



può essere coperta da 3 termini, tutti parzialmente sovrapposti che conducono alla funzione $\overline{x_2}\,\overline{x_4}+\overline{x_1}x_3+x_2x_3x_4$ in forma SP.
Una analoga operazione per ottenere la forma PS conduce invece alla formula: $(\overline{x_2}+x_3)(x_3+\overline{x_4})(\overline{x_1}+x_2+\overline{x_4})(\overline{x_1}+\overline{x_2}+x_4)$ certamente meno conveniente.

Esercizio 8 *Ricavate la formula precedente relativa alla forma PS.*

1.4 Rappresentazione grafica delle funzioni logiche

In elettronica di solito si ricorre agli schemi elettrici per rappresentare i circuiti, ciò può essere fatto anche per rappresentare espressioni logiche booleane. Basta associare un simbolo grafico ad ogni operatore elementare (AND, OR, NOT) e poi collegare i simboli fra di loro per costruire l'equivalente dell'espressione algebrica. I simboli più comuni sono mostrati in figura1.1. Si noti l'uso del pallino sulle uscite per indicare la negazione.

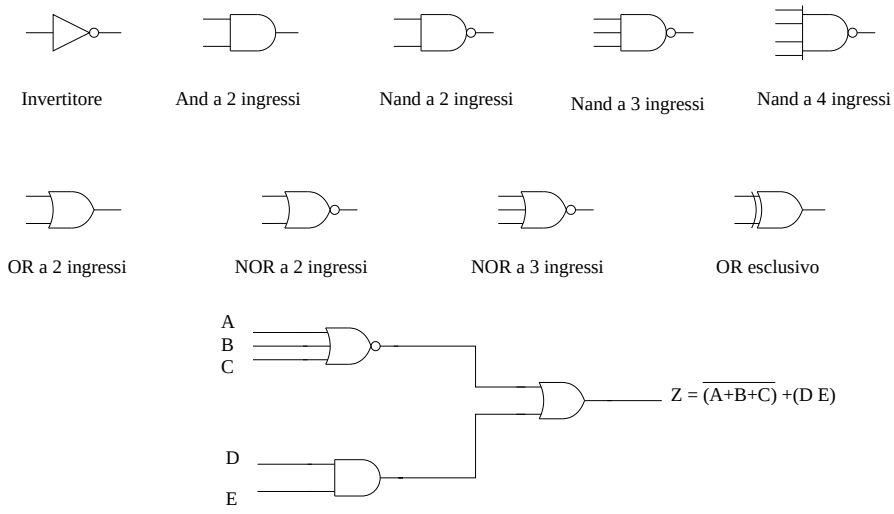


Figura 1.1: Simboli circuitali di alcune funzioni logiche elementari

1.5 Moduli combinatori

Armati delle tecniche algebriche su esposte è possibile progettare circuiti in grado di eseguire operazioni aritmetiche, ad esempio la somma di 2 numeri, infatti la somma di 2 "stringhe" di n bit può essere vista come un

insieme di n+1 funzioni booleane (la somma di 2 numeri di n cifre puo' condurre ad un risultato di n+1 cifre) che semplicemente calcolano il risultato in base ad una tabella di verità che noi forniamo. Seguendo questa linea è possibile progettare diversi "moduli" combinatori che realizzano funzioni utili per la realizzazione di blocchi più complessi: codificatori, decodifiche, selettori, unità aritmetico logiche (ALU).
STUDIARE [1] capitolo 3, Alcuni moduli combinatori.

1.5.1 Codificatori

I codificatori sono circuiti capaci di "tradurre" stringhe di bit da un codice ad un altro, se ne possono immaginare di vari tipi, a seconda dell'applicazione che si ha in mente, ad esempio un codificatore potrebbe avere 2^n ingressi ed n uscite, supponendo che un solo ingresso alla volta possa valere 1 e tutti gli altri debbano valere zero, l'uscita del codificatore potrebbe essere utilizzata per rappresentare in codice binario quale bit in ingresso è diverso da 0. Ecco la tabella di verità per n=2:

x3	x2	x1	x0	y1	y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Esercizio 9 *Ricavare le espressioni booleane per y1 e y0 per il codificatore appena visto, attenzione che molti casi in ingresso non sono ammessi e questo semplifica molto le mappe di Karnaugh che potranno dare risposte arbitrarie di fronte ad ingressi illegali.*

Una variazione del precedente codificatore e' l'aggiunta della priorità: per qualunque configurazione degli ingressi l'uscita rappresenta in codice binario il numero del bit più significativo all'ingresso diverso da zero. A esempio con 4 ingressi la tabella di verità potrebbe essere la seguente dove la X indica un valore qualunque:

x3	x2	x1	x0	En	y1	y0
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	X	1	0	1
0	1	X	X	1	1	0
1	X	X	X	1	1	1

Nel sistema in questione quindi abbiamo i 4 ingressi e le 2 uscite yo e y1, ho aggiunto una uscita supplementare En per distinguere il caso in cui nessun ingresso è diverso da zero da caso in cui lo è il primo. Per realizzare questo modulo occorre sintetizzare 3 funzioni booleane, una per ogni bit, potete facilmente verificare (**Esercizio !**) che si ottiene $E_n = x_0 + x_1 + x_2 + x_3$, $y_0 = x_3 + \overline{x_2} x_1$, $y_1 = x_3 + x_2$.
Questo sistema viene ad esempio utilizzato nei computer col nome di "priority encoder" quando diversi periferici richiedono l'attenzione per stabilire quale deve essere servito per primo.
Un altro esempio di codificatore si ha nel pilotaggio dei display segmentati (quelli usati ad esempio negli orologi digitali o nei display a LED in cui ogni cifra viene disegnata per mezzo di 7 segmenti luminosi), in questi oggetti le singole cifre decimali vengono internamente rappresentate da 4 bit che, per mezzo del codificatore vengono utilizzati per ottenere 7 segnali elettrici, uno per ogni segmento della cifra.

1.5.2 Decodificatori

È possibile immaginare anche degli oggetti che svolgano la funzione inversa ai codificatori: i decodificatori. Ad esempio possiamo immaginare un oggetto con n ingressi $x_{n-1} \cdots x_0$ e 2^n uscite $y_{2^{n-1}} \cdots y_0$ che, per ogni configurazione binaria in ingresso, porti a 1 la linea corrispondente, cioè se le x sono tutte zero, solo y_0 sarà 1, ecc. La tabella di verità per n=2 è la seguente:

x1	x0	y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Esercizio 10 *Calcolare le funzioni booleane che sintetizzano il circuito di decodifica e disegnare il circuito elettrico.*

Esercizio 11 *Progettare un circuito capace di incrementare di 1 i numeri interi compresi fra 0 e 6.*

Esercizio 12 *Progettare un circuito capace di calcolare il quadrato di numeri interi compresi fra 0 e 5.*

1.5.3 Selettori

I selettori, come dice il nome, servono a selezionare uno fra molti ingressi ed avviarlo all'uscita. Saranno perciò dotati di n linee di controllo $c_{n-1} \cdots c_0$, 2^n linee di ingresso $x_{2^{n-1}} \cdots x_0$ e di una sola uscita Y. Y sarà uguale al valore della linea x_i dove i è il valore della stringa di bit $c_{n-1} \cdots c_0$. I selettori si possono disegnare in modo molto semplice usando come sottoblocco una decodifica, lo schema di base è cioè il seguente selettore a 4 ingressi (fig.1.2):

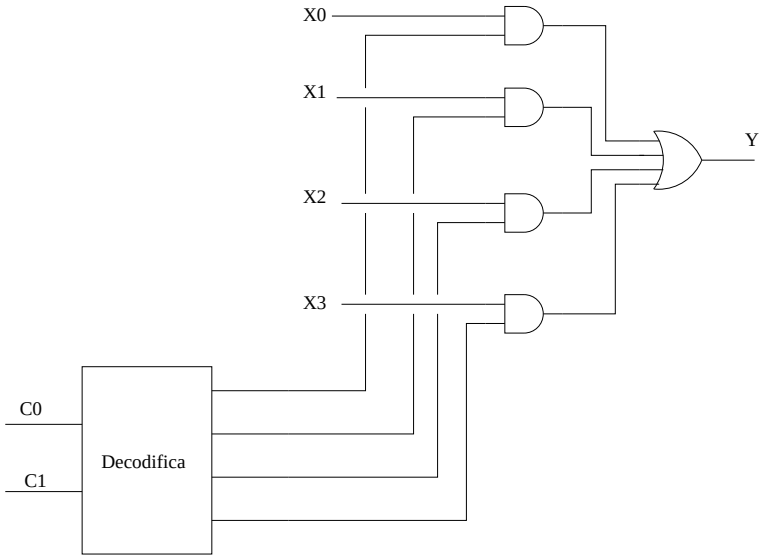


Figura 1.2: Selettore a 4 ingressi

Il funzionamento è il seguente: la decodifica abilita una e una sola porta AND che copia in uscita il segnale in ingresso, l'or terminale mette insieme tutte le possibili sorgenti (ricordare che un AND con un ingresso posto a 0 da sempre un'uscita 0, per cui con questa configurazione solo uno e uno solo dei segnali verà copiato in uscita.

1.5.4 Addizionatori

Supponiamo di avere 2 addendi $X = x_{n-1} \cdots x_0$ e $Y = Y_{n-1} \cdots y_0$, vogliamo calcolare $S = X + Y$ dove $S = s_n \cdots s_0$, possiamo farlo in diversi modi, il modo più ovvio è quello di notare che S è una funzione combinatoria di X e Y e quindi il tutto si riduce a calcolare n+1 funzioni booleane che calcolino gli n+1 bit di S. Ad esempio se n=1 la tavola di verità e la seguente:

x0	y0	s1	s0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Il fatto di essere in grado di calcolare 1+1 non è particolarmente emozionante, quindi viene spontaneo tentare di scrivere la precedente tabella per casi più complicati e, fino ad un certo punto la cosa è possibile, purtroppo però, per un numero di bit ragionevole il problema diventa rapidamente non trattabile, per sommare ad esempio 2 parole di 16 bit occorrerebbe calcolare 17 funzioni booleane e la tabella di verità avrebbe 2^{32} righe, circa 4 miliardi !
Conviene quindi spezzare il problema in somme più semplici, analogamente a quello che facciamo noi quando calcoliamo una somma a mano, ossia è conveniente sommare bit per bit, occorre perciò progettare un sommatore analogo a quello appena visto, con un bit in ingresso in più per il riporto dal bit precedente (meno significativo). Dei 2 bit in uscita quello più significativo deve essere utilizzato come riporto al bit di peso superiore.
La tabella di verità per sommare 2 bit A e B con riporto in ingresso C_{in} e somma S con riporto in uscita C_{out} è la seguente:

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Dalla tabella precedente si ricavano facilmente le espressioni per S e per C_{out}:

$$C_{out} = BC_{in} + AC_{in} + AB$$

$$S = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C_{in}} + A\overline{B}\,\overline{C_{in}} + ABC_{in}$$

Esercizio 13 *Disegnare un sommatore a 3 bit usando "scatole nere" contenenti il precedente sommatore a 1 bit con riporto.*

Se invece di sommare due numeri vogliamo sottrarli possiamo ancora usare il precedente circuito sfruttando il fatto che $A - B = A + (-B)$, perciò il problema si riduce a calcolare la rappresentazione di $-B$ che, per un noto teorema dell'aritmetica binaria, è uguale al complemento di B incrementato di 1.

Esercizio 14 *Disegnare un sottrattore a 3 bit usando "scatole nere" contenenti il precedente sommatore a 1 bit con riporto. L'incremento di 1 potrà essere ottenuto usando intelligentemente i riporti . . .*

Esercizio 15 *Disegnare un circuito capace di sommare o sottrarre 2 numeri di n bit in funzione del valore di un bit di controllo.*

1.6 I ritardi di propagazione

Un punto importante da sottolineare è la incapacità dell'algebra booleana di incorporare il "ritardo" nel calcolo del circuito logico: ogni dispositivo fisico che realizza una funzione logica impiega un certo tempo per effettuare il calcolo, tempo legato non solo alla velocità di propagazione dei segnali, sempre (molto) minore della velocitá

della luce, ma anche alla presenza di capacità e resistenze interne al circuito che, introducendo delle costanti di tempo, riducono la velocità di variazione dei segnali. (Gli effetti induttivi sono quasi sempre trascurabili)
Ad esempio il circuito mostrato in figura 1.3 dovrebbe fornire un'uscita sempre uguale a 0, invece, a causa del ritardo dell'invertitore, una transizione $0 \rightarrow 1$ all'ingresso genera un impulso in uscita di durata confrontabile con il ritardo intrinseco della logica.

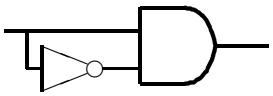


Figura 1.3: Derivatore logico

Esercizio 16 *Studiare nel dettaglio il funzionamento del derivatore logico mostrato sopra.*

Naturalmente gli effetti legati ai ritardi di propagazione del segnale sono solitamente nocivi, non solo per il rallentamento del funzionamento sistema ma, soprattutto, perchè, come nell'esempio precedente, creano impulsi spurii, solitamente molto brevi, che facilmente creano problemi al progettista.

Esercizio 17 *Esaminare i ritardi generati in un sommatore a n bit, dimostrare che il tempo impiegato per effettuare la somma nel semplice schema proposto precedentemente è linearmente proporzionale al numero di bit n.*
Come si potrebbe fare per ridurre il tempo di calcolo?

1.7 Porte logiche 3-state

A volte sarebbe utile poter collegare diverse porte logiche sulla stessa connessione fisica, specie quando la connessione è molto lunga o costosa, oppure si vorrebbe poter trasferire i dati in entrambi i sensi lungo la stessa connessione, come mostrato in figura 1.4

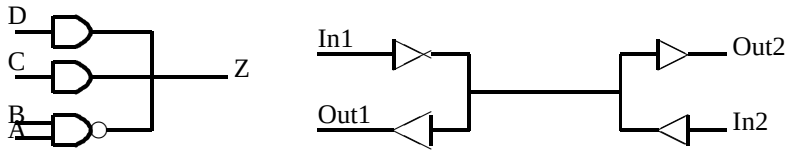


Figura 1.4: Molte porte logiche sulla stessa connessione (Da non fare !)

Purtroppo questo è normalmente **vietato**, infatti ogni porta logica cerca di affermare il suo valore logico (cioè la sua tensione) all'uscita; poichè i valori logici saranno in generale diversi, nasce un conflitto (una specie di corto-circuito) per cui la tensione diventa un valore intermedio fra quella corrispondente allo "0" e quella corrispondente all' "1" con forte passaggio di corrente e danneggiamento dei dispositivi.
D'altra parte ci sono casi in cui questa necessità è molto sentita, pensiamo ad esempio al trasferimento dei dati tra una CPU ed una memoria: per ridurre il numero dei collegamenti (ed il costo) bisogna fare in modo che la stessa connessione CPU–Memoria possa essere utilizzata in entrambi i sensi, anzi, generalizzando, è opportuno, al fine di ridurre i costi e aumentare la flessibilità del sistema, immaginare di avere un unico collegamento fra diversi

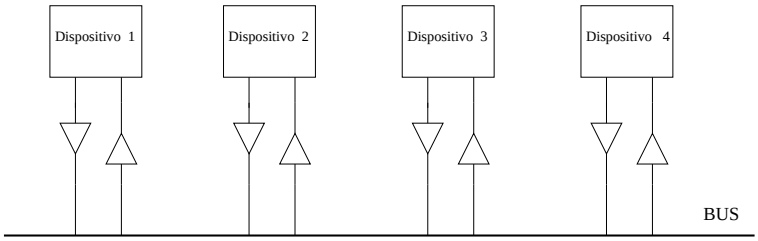


Figura 1.5: Sistema con 4 dispositivi che comunicano attraverso un “BUS”

dispositivi e che, lungo questa connessione, viaggino tutte le informazioni fra coppie qualunque di dispositivi, questo metodo di collegamento prende il nome di “bus” (vedi 1.5).
Ma per poter realizzare quanto sopra è necessario che una sola porta logica per volta possa pilotare i collegamenti applicando il suo livello logico alle connessioni con gli altri dispositivi, le altre porte logiche devono essere inibite. Questa è l’idea alla base delle porte “tri-state”, ossia dispositivi logici che oltre a fornire i classici valori logici, sono anche in grado di “disconnettersi” dal circuito in modo da permettere alle altre porte logiche di funzionare normalmente. Questa terzo valore logico (oltre a “0” e “1”) viene indicato dal carattere Z o dalla sigla “HiZ” che stanno ad indicare che l’output della porta logica si mette in uno stato di “alta impedenza” e quindi si disconnette in pratica dal circuito. Per controllare questa caratteristica le porte “tri-state’ sono fornite di un ulteriore ingresso di controllo, spesso chiamato “Enable”, che agisce sulla circuiteria interna della porta spegnendo tutti i transistor collegati all’uscita e quindi riducendo praticamente a zero l’effetto della porta sugli altri circuiti collegati ad essa. Simbolo di una porta logica “3-state” 1.6).

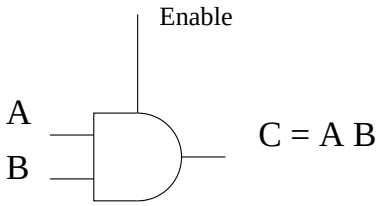


Figura 1.6: Porta logica (AND a 2 ingressi) con “Enable” per la funzionalità “3-state”. C = A B se “Enable” = 1, C = “HiZ” se “Enable” = 0.

Se adesso vogliamo realizzare la trasmissione bidirezionale dei dati come mostrato in figura 1.5 dobbiamo semplicemente sostituire tutte le porte collegate con l’uscita sul bus con porte 3-state e fare in modo che solo una di esse sia abilitata ad ogni istante, questo naturalmente richiede che, comunque, vi sia un coordinamento nel funzionamento dei vari dispositivi collegati al bus, problema complesso che non vogliamo trattare qui.

Capitolo 2

Sistemi sequenziali

Fino a questo momento ci siamo occupati di **circuiti combinatori**, circuiti capaci di fornire un output legato unicamente agli ingressi attuali, ossia un output che non dipende dal valore degli ingressi in istanti precedenti. Questo è insufficiente se vogliamo progettare un calcolatore o comunque un sistema che debba eseguire una serie di operazioni in sequenza, magari variando il tipo di operazione in funzione di risultati intermedi. Occorre perciò pensare a **sistemi "sequenziali"**, ossia sistemi il cui output dipenderà non solo dagli ingressi attuali ma anche dalla storia precedente. Classico esempio è quello della macchinetta distributrice di bevande che esegue un certo numero di operazioni in sequenza, dove le operazioni eseguite dipendono da ciò che ha chiesto l'utente (se ha messo i soldi, quale bevanda ha scelto, se ha chiesto lo zucchero, ecc.) e da ciò che ha preliminarmente già fatto la macchina (prima il bicchierino, poi il caffè, ecc). Con le sole reti logiche combinatorie una macchina del genere non sarebbe costruibile perchè in esse l'output non dipende dalla storia precedente.
Questa parte introduttiva è esposta chiaramente in [1] .
Un esempio di circuito sequenziale può essere illuminante prima di affrontare il problema in modo piu' formale. Si consideri il seguente circuito logico (fig:2.1:

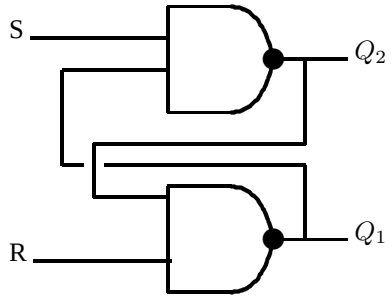


Figura 2.1: Flip–Flop RS

Le uscite del circuito in figura dipendono dai due ingressi R ed S, vi sono pertanto 4 casi possibili, 3 casi sono facilmente calcolabili:

R	S	Q_1	Q_2
0	0	1	1
0	1	1	0
1	0	0	1
1	1	?	?

Il quarto caso, contrassegnato dai ? non dipende solo dagli ingressi come si può facilmente verificare, ponendo, per prova, $Q_1 = 0, Q_2 = 1$ si ottiene una soluzione consistente, ossia i valori logici ipotizzati sono coerenti con le tabelle di verità della logica impiegata, d'altra parte, scambiando il valore di Q_1 e Q_2 si ottiene una soluzione altrettanto valida (non per nulla il circuito è totalmente simmetrico). Si noti che, pur avendo un sistema simmetrico, la soluzione non lo è, ci troviamo quindi di fronte ad un esempio di rottura spontanea della simmetria.
Dobbiamo perciò concludere che le 2 uscite del sistema non sono determinabili in modo univoco conoscendo solo gli ingressi.
L'output del sistema può essere calcolato solo conoscendo la storia precedente degli ingressi, se, ad esempio, gli ingressi hanno avuto la seguente storia: R=1, S=0, successivamente R=1, S =1, l'output sarà $Q_1 = 0, Q_2 = 1$, ossia quello corrispondente alla prima coppia di valori in ingresso, la seconda coppia di ingressi ha solo la funzione di memorizzare lo "stato interno" del circuitino. Il sistema appena visto prende il nome di flip–flop RS (in italiano bistabile) e rappresenta il più semplice circuito sequenziale. Se ci chiediamo cosa distingue il sistema appena visto da un circuito combinatorio possiamo notare che il flip–flop è caratterizzato dalla presenza di "cicli", ossia dalla presenza di percorsi chiusi che partendo da un nodo vi ritornano, ad esempio partendo da una delle 2 uscite è possibile ritornare all'ingresso dell'altro nand, da questo, dopo avere attraversato il nand si arriva sull'altra uscita del circuito che ci invia alla prima porta logica e da questa al punto di partenza.
La presenza dei cicli implica che in qualche modo l'uscita influisce sull'ingresso per cui la conoscenza degli ingressi non fissa in modo univoco le uscite. Quindi possiamo affermare che un sistema sequenziale e' caratterizzato dalla presenza di cicli, ossia sara' sempre possibile rappresentarlo in questo modo (fig:2.2):

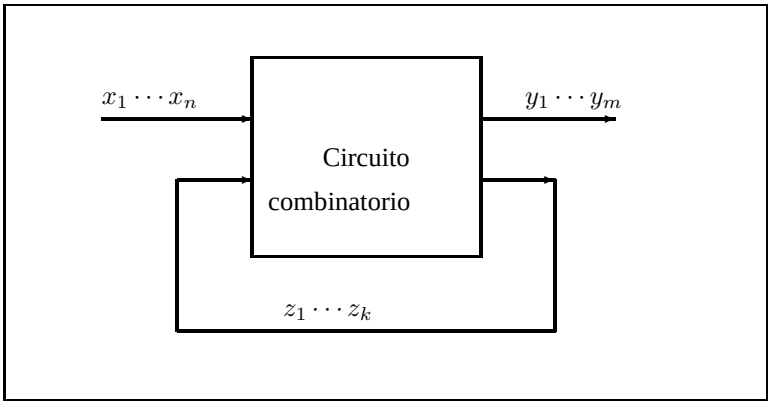


Figura 2.2: Schema a blocchi di un generico sistema sequenziale

Come si vede il sistema è costituito da una "scatola nera" provvista di un certo numero di ingressi e di uscite, inoltre sono stati isolati k cicli (la cui individuazione non è detto che sia univoca). Per calcolare la relazione ingressi–uscite del sistema bisognerà tener conto anche delle variabili di ciclo $z_1 \cdots z_k$, infatti noi stiamo implicitamente assumendo che le z siano funzione delle x e delle z nell'istante precedente, le y invece saranno funzione sia delle z in ingresso che delle x. Le variabili di ciclo vengono usualmente chiamate variabili di stato. Il funzionamento del sistema può essere piuttosto complicato, infatti una variazione degli ingressi può provocare una variazione delle uscite e delle variabili di stato con ulteriore variazione delle uscite e delle variabili di stato che a loro volta producono ulteriori variazioni delle variabili di stato e così via. Possiamo avere a priori un numero anche molto elevato di transizioni prima che il sistema si assesti in uno stato stabile (che non è detto che esista!).

Il comportamento appena citato è non solo complesso, è soprattutto inaffidabile, infatti le transizioni da uno stato all'altro sono condizionate dalle uscite del sistema che sono soggette a vincoli tecnologici, è ad esempio impossibile fare in modo che le variabili di stato e le uscite variino tutte contemporaneamente (in generale dipendono da funzioni booleane differenti e quindi i ritardi potranno essere leggermente differenti, non foss'altro che per il fatto che è impossibile costruire 2 porte logiche assolutamente identiche. Quindi non possiamo essere sicuri che ad una variazione degli ingressi simultanea (ipotesi, per quanto appena detto, assolutamente irrealizzabile) le variabili di stato assumerebbero tutte simultaneamente il nuovo valore e quindi il sistema potrebbe interpretare erroneamente il transiente e fornire nuovi output e nuove variabili di stato sbagliate che lo trascinerebbero in uno stato finale diverso da quello che si sarebbe avuto senza il transiente erroneo.
Il modo più semplice per evitare il problema è quello di spezzare l'anello di retroazione ed introdurre una memoria temporanea come mostrato in figura 2.3:

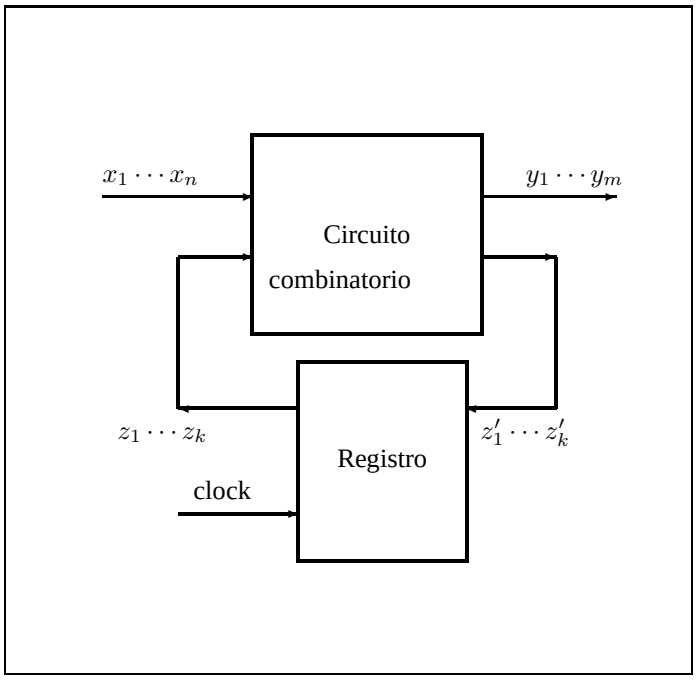


Figura 2.3: Sistema sequenziale sincrono (con registro)

Per costruire un sistema sequenziale occorre quindi un registro che memorizzi lo stato interno del sistema in modo stabile per dare tempo ai circuiti combinatori di calcolare il prossimo stato, viene spontaneo pensare al flip–flop RS introdotto precedentemente, c'è però un problema, il flip–flop RS cambia stato appena variano gli ingressi, quindi non "isola" gli ingressi e le uscite della parte combinatoria, occorre quindi introdurre un comando di sincronismo (clock) per autorizzare le variazioni con un ritmo prefissato e sufficientemente rilassato da non creare problemi con i ritardi presenti. Aggiungiamo pertanto al nostro flip–flop un semplice circuito di separazione fra gli ingressi del flip–flop ed il resto del sistema che permetta le transizioni solo quando il clock è "alto": arriviamo così al clocked RS (fig:2.4):
Questo dispositivo ha la proprietà di accettare i valori R' ed S' ed applicarli al flip–flop RS solo se l'ingresso di clock è 1, altrimenti il flip–flop risulta isolato. Ci sono tuttavia ancora 2 problemi:

- Non è stato eliminato lo "stato proibito" che a questo punto si ottiene se R'=S'=1.
- Il sistema può funzionare male: supponiamo per esempio di volere che il flip–flop inverta lo stato ad ogni impulso di clock, ossia l'uscita Q debba assumere i valori 0,1,0,1,... ad ogni impulso di clock; questo puo' essere ottenuto collegando Q con R' e $\overline{Q}$ con S' (verificare), supponiamo inoltre che il nostro clock sia

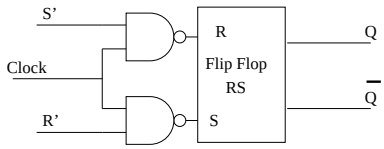


Figura 2.4: Flip–flop “clocked RS”

un segnale ad “onda quadra” che oscilla fra i valori 0 e 1 con tempi uguali, se ora il clock rimane nello stato 1 per un tempo maggiore dei ritardi caratteristici del sistema può accadere che la prima transizione del sistema ne provochi immediatamente un’altra od eventualmente molte altre prima che il clock ritorni a 0. Unica soluzione al problema sarebbe usare un clock asimmetrico in cui l’intervallo temporale per cui il valore del clock è 1 fosse confrontabile con i tempi di propagazione della logica, ma non molto più breve ne’ più lungo, altrimenti anche la prima transizione potrebbe non avvenire per la lentezza dei dispositivi elettronici utilizzati, oppure potrebbero avvenirne 2, per cui questa soluzione è di difficile realizzazione pratica e inaffidabile.

La soluzione dei 2 problemi citati è fornita dal flip J–K master–slave (fig:2.5):

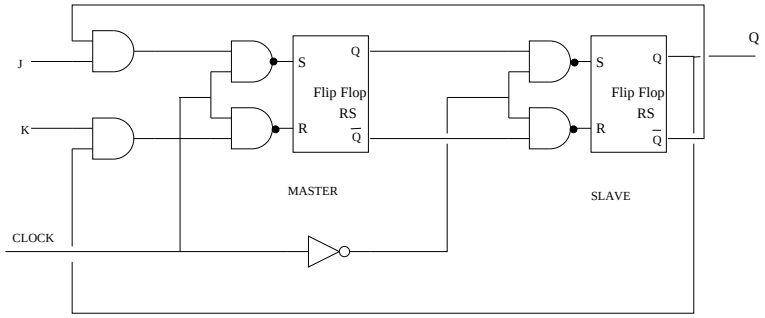


Figura 2.5: Flip-flop master–slave

Come vedete nel sistema ci sono adesso 2 flip–flop RS, il “master” e lo “slave”, lo slave riceve il clock del master negato, questo implica che quando uno dei due flip-flop aggiorna il proprio stato perchè il suo clock è alto, l’altro invece rimane nel suo stato iniziale, in questo modo è impossibile avere più di una transizione per volta. Il funzionamento del circuito è riassunto da questa tabella di verità (la cui verifica è lasciata come **esercizio**, nel farla si tenga conto che lo stato iniziale dello slave dovrà essere coerente con quello del master ossia $Q(slave) = \overline{Q(master)}$):

Flip–flop J–K			
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	X	0
1	0	X	1
1	1	0	1
1	1	1	0

Ossia, detto a parole, se J=K=0 il flip–flop non cambia stato all’arrivo del clock, se J=1, K=0, il flip–flop si posiziona con Q=1 viceversa, se J=0 e K=1, Q diventa 0, il caso in cui J=K=1 viene utilizzato per negare Q. Il

flip–flop JK è una generalizzazione dell'RS, infatti, variando gli ingressi può essere messo nei 2 stati possibili, può essere bloccato nello stato corrente, ma può anche essere costretto ad andare nello stato complementare, questa è una possibilità in più che rimpiazza il cosiddetto stato proibito dell'RS.
L'ultimo flip–flop di cui parleremo è il tipo D, tra l'altro il più usato.
In questo caso abbiamo un solo ingresso, chiamato D, ed una uscita Q, la tabella di verità è banale:

Flip–flop D	
D	Q
0	0
1	1

Una possibile realizzazione (concettuale) di questo dispositivo è mostrata in figura 2.6:

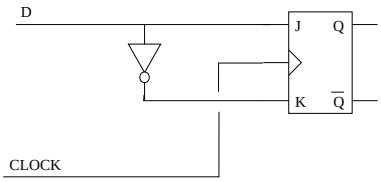


Figura 2.6: Flip–flop D

Come si può notare il flip–flop si limita a "copiare" sull'uscita Q il valore di D presente all'ingresso, attenzione però, questa copia viene effettuata solo in corrispondenza delle transizioni del clock, ossia le variazioni di D non vengono trasferite su Q automaticamente, se così fosse il flip–flop sarebbe inutile, ma solo quando il clock esegue una particolare transizione, ad esempio da 0 a 1. Il dispositivo si comporta cioè da cella di memoria capace di memorizzare un singolo bit, controllata dal segnale di clock, che come avete visto su [1], è pressoche indispensabile per evitare problemi legati alla temporizzazione degli ingressi ed al ritardo della logica.
Nella pratica il flip–flop D è molto diffuso, se volete provarne uno in laboratorio, cercate il modello 74LS74 che come vedrete è in realtà leggermente più complesso, infatti presenta altri 2 ingressi: set e reset, chiamati anche preset e clear la cui funzione è di forzare il flip–flop in uno stato noto, prescindendo dal valore di clock (ingresso asincrono), si noti inoltre la presenza di una ulteriore uscita $\overline{Q}$ che è semplicemente la negazione di Q.
Riassumendo la tabella di verità di un flip–flop di tipo D 74LS74 è la seguente:

$\overline{Set}$	$\overline{Clear}$	D	Clock	Q	$\overline{Q}$
0	0	X	X	1	1
0	1	X	X	1	0
1	0	X	X	0	1
1	1	0	$\uparrow$	0	1
1	1	1	$\uparrow$	1	0

Nota: Set e Clear sono attivi bassi (ossia agiscono quando sono zero, questo spiega la notazione con la sbarra: $\overline{Set}$.
X indica un valore non specificato.
Se asserisco il comando di Clear il flip–flop deve porsi nello stato con Q=0 a prescindere dal valore di D e del clock. Analogamente il Set forza l'uscita Q=1. Si noti che non ha significato asserire Set e Clear contemporaneamente, implicherebbe che il flip–flop dovrebbe obbedire a 2 comandi contraddittorii (prima riga della tabella precedente).
La freccia $\uparrow$ indica la transizione da 0 a 1 del clock.

2.1 Sintesi di un sistema sequenziale

Il progetto di un sistema sequenziale prende le mosse dalle specifiche, ossia dalla descrizione della risposta del sistema ad una sequenza arbitraria di ingressi. Occorre cioè fornire

- Gli ingressi
- Le uscite
- Gli stati interni
- Una descrizione di come sono strutturati gli stati interni
- La descrizione dei valori delle uscite per ogni possibile valore degli ingressi e degli stati interni

La struttura degli stati interni, ossia come l'automa passa da uno all'altro in funzione degli ingressi e il valore degli output possono essere specificati in almeno 2 modi:

- Attraverso una tabella di transizione
- Attraverso un grafo orientato

La tabella di transizione è semplicemente una matrice che reca sulle righe l'elenco degli stati e sulle colonne l'elenco degli input, per ogni combinazione di stato interno / ingresso bisogna specificare nella tabella il valore del prossimo stato e dell'output corrispondente.
Analogamente il grafo orientato è un insieme di "bolle", che rappresentano gli stati, da cui si dipartono degli archi che collegano gli stati fra di loro, su ogni arco viene indicato quale input provoca la transizione e l'output corrispondente, l'arco termina con una freccia per indicare la direzione della transizione.
Supponiamo ad esempio di voler costruire un semplice automa che controlli il livello di un liquido in un recipiente: il contenitore è dotato di 2 sensori, uno per il livello minimo (Smin), uno per il livello massimo (Smax), l'afflusso del liquido è controllato da una elettrovalvola (V) che può essere aperta o chiusa da un segnale logico. L'utente del recipiente potrà prelevare il liquido attraverso un altro rubinetto posto sul fondo su cui il sistema in questione non ha alcun controllo.
Assumiamo che i sensori diano un output 0 se il liquido è al di sotto del livello, 1 se al di sopra, assumiamo altresì che la valvola si chiuda se il suo input di controllo è 0, aperta se è 1. Scopo del sistema da progettare sarà quello di evitare che il livello del liquido scenda sotto il livello Smin, aprendo automaticamente la valvola V e riportando automaticamente il liquido al livello Smax.
Il sistema non può certamente essere realizzato in modo combinatorio, infatti quando il livello del liquido è compreso fra i 2 sensori la valvola può essere aperta o chiusa a seconda della storia precedente.
È chiaro che Smin ed Smax sono i 2 input e V è l'output, un poco più complesso è capire quali sono gli stati interni, anche perchè spesso vi è una certa arbitrarietà. Se analizziamo il comportamento del sistema ci accorgiamo che esso sostanzialmente deve solo tenere la valvola aperta (V=1) o chiusa (V=0), e quando il livello dell'acqua è compreso fra i 2 sensori lo stato della valvola può essere dedotto solo sapendo quale sensore è stato raggiunto per ultimo, non vi sono altre situazioni, si può perciò tentativamente provare a costruire un grafo con 2 soli stati che potremo chiamare "A" e "B", si passerà dallo stato "A" a "B" tutte le volte che Smin passerà da 1 a 0 (recipiente quasi vuoto, apertura valvola), si passerà da "B" ad "A" tutte le volte che Smax passerà da 0 a 1 (recipiente quasi pieno, chiusura della valvola).
Su ogni arco di collegamento fra i due stati indicheremo il valore degli ingressi e, separati da una sbarra (/), il valore degli output.
Dopo avere disegnato il diagramma di stato occorre tradurlo in una *tabella di transizione*, la quale contiene le stesse informazioni del grafo precedente espresse in altro modo, la tabella contiene, in funzione dello stato corrente e degli ingressi, il valore del prossimo stato.
La tabella di transizione del nostro esempio sarà:

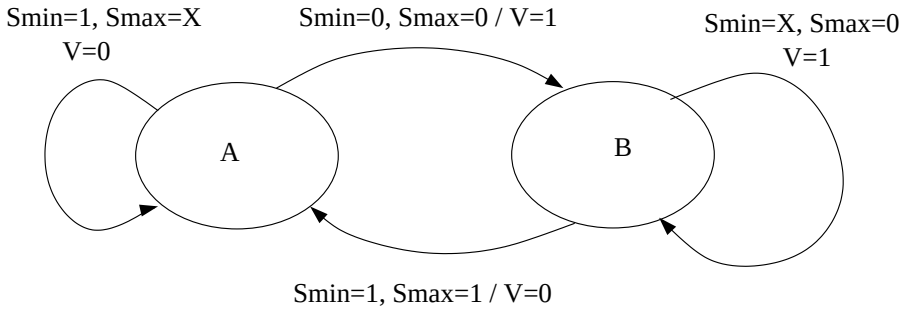


Figura 2.7: Grafo dell'automa per il controllo della elettrovalvola

Tabella di transizione

Stato corrente	Smin, Smax			
	0 0	0 1	1 0	11
A	B, 1	X, X	A, 0	A, 0
B	B, 1	X, X	B, 1	A, 0

A questo punto è opportuno assegnare ad ogni stato una stringa di bit arbitraria, di solito si assegnano le stringhe corrispondenti ai numeri interi, usando il minor numero di bit possibile, se, ad esempio, abbiamo 5 stati, potremo assegnare ai 5 stati le stringhe: 000, 001, 010, 011, 100. Si noti che si sono usati 3 bit, con cui è possibile distinguere 8 stati diversi (da 000 a 111), più generalmente il numero dei bit deve essere tale da soddisfare la seguente disuguaglianza: $2^N \geq$ numero degli stati, dove N è il numero dei bit occorrenti.
Per il nostro esempio N = 1 poichè abbiamo solo 2 stati, potremo perciò arbitrariamente associare ad A la stringa 0 e a B la stringa 1, incidentalmente, con questa scelta il valore dello stato coincide col valore delle uscite e questa è una semplificazione. È inoltre opportuno dare un nome ai bit di stato, ad esempio y_0, y_1, ecc., nel nostro caso abbiamo un solo bit, chiamiamolo y.
L'ultima cosa da fare è costruire le funzioni booleane che, in funzione degli ingressi e dello stato corrente, calcolino il prossimo stato e le uscite. Le informazioni sono contenute nella tabella di transizione, nel nostro caso, particolarmente semplice, dovremo calcolare una sola funzione booleana, quella relativa al prossimo stato. Sostituendo ad A e B il loro valore binario la tavola di verità per il calcolo del prossimo stato assume la forma:

Tabella per il calcolo del prossimo stato

Stato corrente	Smin, Smax			
y	0 0	0 1	1 0	11
0	1	X	0	0
1	1	X	1	0

A questo punto abbiamo le specifiche delle funzioni booleane per il calcolo del prossimo stato, possiamo quindi compilare le mappe di Karnaugh:

y \ Smin,Smax	00	01	11	10
0	1	X	0	0
1	1	X	0	1

La funzione relativa al prossimo stato (y') risulta essere:

$$y' = \overline{S_{min}} + y\,\overline{S_{max}}$$

Nota: la scelta delle transizioni è sempre un po' arbitraria, si sarebbe ad esempio potuto decidere di sostituire le 2 X nella seconda colonna con 2 zeri per essere certi che in caso di informazioni incoerenti provenienti dai sensori di livello (e quindi probabile guasto) il rubinetto sia comunque chiuso, ottenendo quindi:

$$y' = \overline{S_{min}}\;\overline{S_{max}} + y\,\overline{S_{max}}$$

Il registro viene solitamente ottenuto usando dei flip-flop di tipo D.
Il disegno del circuito elettrico corrispondente è mostrato in figura 2.8.

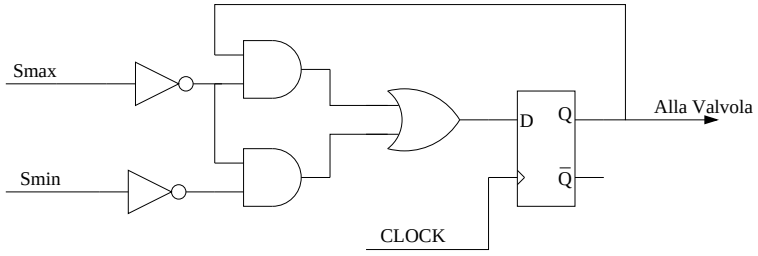


Figura 2.8: Schema automa controllo elettrovalvola

Esercizio 18 *Si supponga di disporre di 2 cisterne fornite di 2 sensori di livello come nel caso precedente e di 2 tubi in uscita, ognuno dotato di elettrovalvola. Quando una delle 2 cisterne è vuota gli utilizzatori vengono automaticamente collegati all'altra mentre la prima si riempe. Progettate un automa che controlli i 2 ingressi e le 2 uscite.*
Quando avrete studiato il capitolo sui circuiti programmabili potrete verificare il funzionamento del vostro progetto usando "Palasm".

Una distinzione importante (e sottile) a proposito dei sistemi sequenziali è quella riguardante la macchine di Mealy e di Moore. Un automa è una macchina di Mealy se le uscite sono funzione dello stato interno e degli ingressi; un automa è una macchina di Moore se le uscite sono funzione solo dello stato interno. Evidentemente le macchine di Mealy rappresentano il caso più generale possibile, le macchine di Moore sono un sottocaso più semplice. Per comprendere la differenza fra i 2 tipi di sistemi dobbiamo osservare le uscite che, di solito, saranno inviate ad un altro sistema digitale, ci dobbiamo perciò chiedere quando le uscite siano valide e possano essere utilizzate: nel caso di Mealy ogni variazione degli ingressi provoca variazioni in uscita, perciò, se noi vogliamo essere sicuri di quello che leggiamo dobbiamo farlo nel momento dell'arrivo del segnale di sincronismo, infatti quando il sistema

compie una transizione gli ingressi DEVONO essere stabili, se non è così il sistema è stato mal progettato e non potrà funzionare. Nel caso di Moore invece le uscite sono funzione solo dello stato e quindi una variazione degli ingressi non risulta visibile in output fino a che il sistema non transisce ad altro stato e quindi il valore delle uscite può essere utilizzato durante tutto il lasso di tempo fra una transizione del clock e la successiva.
Si può dimostrare che ogni problema sequenziale può essere risolto sia da una macchina di Mealy che da una macchina di Moore, naturalmente il grafo che ne descriverà il funzionamento sarà diverso nei 2 casi, le uscite saranno però identiche. Di solito un automa di Mealy richiede meno stati di un automa di Moore, tuttavia la temporizzazione è più delicata.

2.2 Secondo esempio di sintesi di un sistema sequenziale

Vogliamo progettare un sistema di allarme per un locale con cassaforte. Il sistema dovrà controllare i segnali provenienti da 2 sensori (uno sulla porta di ingresso, l'altro su una seconda porta che dà accesso alla cassaforte).
Nel caso di apertura, anche solo momentanea proveniente dalla prima porta, il sistema deve lasciar passare un intervallo di tempo predefinito (per poter disinserire l'allarme) e poi attivare la sirena.
Se invece viene aperta (anche solo momentaneamente) la seconda porta, qualunque sia lo stato della prima porta, si deve attivare immediatamente la sirena.
Il sistema deve essere "sicuro" ossia, in caso di dubbio, deve suonare l'allarme.
Il temporizzatore potrà essere ottenuto usando un contatore a 3 bit.
Inoltre vi sarà un ingresso di reset che sarà l'unico modo per bloccare sia il temporizzatore che la sirena, si suppone che questo sia anche il bottone utilizzato dalle persone autorizzate per accedere alla cassaforte ...
Questo esempio è decisamente più complicato di quello della cisterna, in realtà è opportuno costruirlo usando 2 sistemi sequenziali:

- il primo è l'automa che, in funzione degli ingressi e del temporizzatore, controlla la sirena
- il secondo è il temporizzatore, realizzato con un contatore a 3 bit, quando arriva al conteggio massimo (7) fornisce un segnale al primo blocco circuitale.

Il diagramma a blocchi è mostrato in figura 2.9

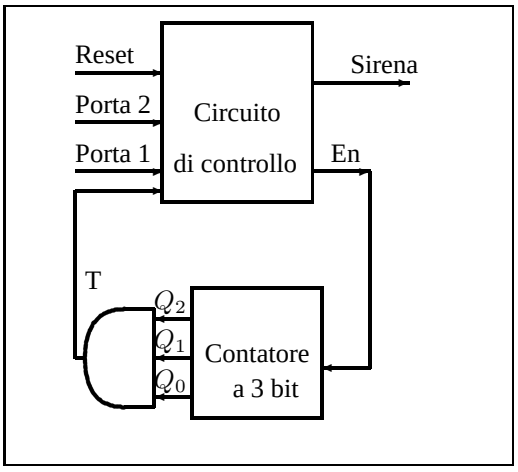


Figura 2.9: Schema a blocchi del "Sistema di allarme"

Come si vede abbiamo utilizzato un contatore a 3 bit le cui uscite Q_0, Q_1, Q_2 sono poste in AND per decodificare la condizione "1 1 1" che corrisponde al massimo conteggio.

Il progetto del contatore è standard, (vedi comunque prossima sezione) possiamo scrivere le 3 equazioni usando le tecniche generali per il progetto dei contatori:

$$Q_0' = \overline{Q_0}En + Q_0\overline{En}$$
$$Q_1' = (\overline{Q_1}Q_0 + Q_1\overline{Q_0})En + Q_1\overline{En}$$
$$Q_2' = (\overline{Q_2}(Q_0Q_1) + Q_2\overline{(Q_0Q_1)})En + Q_2\overline{En})$$

L'indicazione del raggiungimento del massimo conteggio (111 ossia 7 decimale) si ottiene con l'equazione

$$T = Q_0Q_1Q_2$$

.
Dobbiamo adesso progettare l'automa di controllo che avrà come ingresso le 2 porte (P_1, P_2), il reset R, l'output del contatore T, in output avremo la sirena S e l'abilitazione del contatore En.
Cerchiamo di individuare gli stati che il nostro sistema dovrà necessariamente avere:

- Uno stato **I**niziale in cui il sistema si porta quando riceve il comando di reset (nessuno ha aperto da lungo tempo le porte, non ci sono allarmi, il temporizzatore è inattivo)
- Uno stato di **P**reallarme (è stata aperta la porta 1, il contatore ha iniziato a contare ma il tempo non è ancora scaduto)
- Uno stato di **A**llarme: il temporizzatore è arrivato in fondo e/o la porta 2 è stata aperta.

Possiamo quindi disegnare il grafo orientato (2.10) che mostra i 3 stati e le transizioni, la notazione sarà quella già introdotta: su ogni arco di collegamento fra 2 stati indicheremo gli ingressi, una sbarra (/) quindi le uscite, nel nostro caso avremo Porta1, Porta2, Temporizzatore / Sirena, Enable.
Il reset verrà aggiunto a mano alla fine.

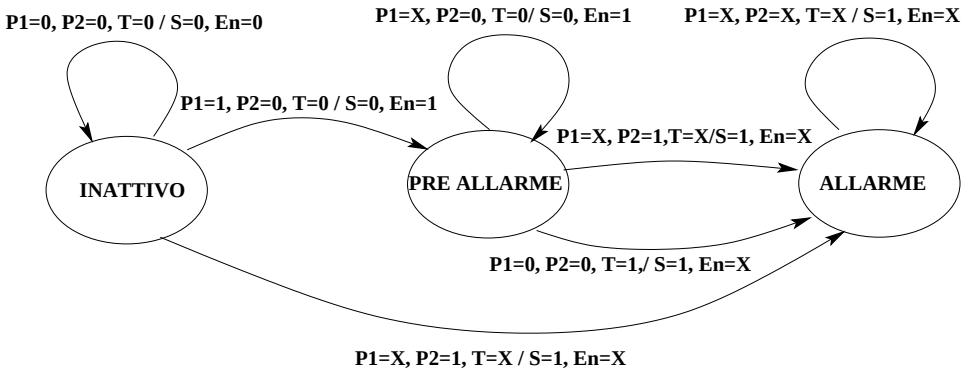


Figura 2.10: Grafo del sistema di allarme

Esercizio 19 *Aggiungete un quarto bit al contatore precedente.*

Esercizio 20 *Analizzate il grafo e controllate il significato di ogni transizione*

Dal grafo è possibile dedurre immediatamente la tabella di transizione che, in funzione dello stato corrente (**I**nattivo, **P**reallarme, **A**llarme) e degli ingressi (P_1, P_2, T), mostrerà lo stato successivo del sistema e delle 2 uscite (Sirena ed Enable):

$P_1, P_2, T \rightarrow$	000	001	010	011	100	101	110	111
Stato $\downarrow$								
I	I, 0 0	X	A, 1 X	A,1 X	P, 0 1	X	A, 1 X	A, 1 X
P	P, 0 1	A, 1 X	A, 1 X	A, 1 X	P, 0, 1	A, 1 X	A, 1 X	A, 1 X
A	A, 1 X	A, 1 X	A, 1 X	A, 1 X	A, 1 X	A, 1 X	A, 1 X	A, 1 X

Adesso dobbiamo attribuire ad ogni stato una stringa di bit univoca (ma totalmente arbitraria), siccome abbiamo 3 stati ci bastano 2 bit, ad esempio possiamo porre Inattivo = 00, Preallarme = 01, Allarme = 10. I 2 bit di stato li chiameremo Y_1 e Y_0. La tabella precedente pertanto diventa:

$P_1, P_2, T \rightarrow$	000	001	010	011	100	101	110	111
Stato								
$Y_1, Y_0 \downarrow$								
00	00, 0 0	X	10, 1 X	10,1 X	01, 0 1	X	10, 1 X	10, 1 X
01	01, 0 1	10, 1 X	10, 1 X	10, 1 X	01, 0, 1	10, 1 X	10, 1 X	10, 1 X
10	10, 1 X	10, 1 X	10, 1 X	10, 1 X	10, 1 X	10, 1 X	10, 1 X	10, 1 X

Come al solito le X indicano un valore non specificato, potremo porlo uguale a 0 o a 1 come più ci conviene.
La tabella precedente sostanzialmente ci fornisce i valori del prossimo stato $(Y_1', Y_0'$, S, En, possiamo estrarre le 4 funzioni booleane e riscriverle sotto forma di mappa di Karnaugh. Ecco la mappa per Y_1':

T=0

Y_1, Y_0 \ P_1, P_2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	X	X	X	X
10	1	1	1	1

T=1

Y_1, Y_0 \ P_1, P_2	00	01	11	10
00	X	1	1	X
01	1	1	1	1
11	X	X	X	X
10	1	1	1	1

Da cui $Y_1' = Y_1 + P_2 + T$.
Analogamente per Y_0':

T=0

Y_1, Y_0 \ P_1, P_2	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	X	X	X	X
10	0	0	0	0

T=1

Y_1, Y_0 \ P_1, P_2	00	01	11	10
00	X	0	0	X
01	0	0	0	0
11	X	X	X	X
10	0	0	0	0

Da cui facilmente si ottiene $Y_0' = Y_0\overline{P_2}\,\overline{T} + \overline{Y_1}P_1\overline{P_2}\,\overline{T}$.

Analogamente per S (la sirena):

T=0

Y_1,Y_0 \ P_1,P_2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	X	X	X	X
10	1	1	1	1

T=1

Y_1,Y_0 \ P_1,P_2	00	01	11	10
00	X	1	1	X
01	1	1	1	1
11	X	X	X	X
10	1	1	1	1

Da cui facilmente si ottiene $S = P_2 + T + Y_1$.
Analogamente per En:

T=0

Y_1,Y_0 \ P_1,P_2	00	01	11	10
00	0	X	X	1
01	1	X	X	1
11	X	X	X	X
10	X	X	X	X

T=1

Y_1,Y_0 \ P_1,P_2	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10	X	X	X	X

Da cui si ottiene $En = Y_0 + P_1$.
Dobbiamo adesso aggiungere il comando di reset, se vogliamo che il reset sia, ad esempio, attivo quando è basso dovremo semplicemente porlo in AND con le espressioni logiche ottenute, supponendo di chiamarlo R, avremo:

$$Y_1' = (Y_1 + P_2 + T)R$$

$$Y_0' = (Y_0\overline{P_2}\,\overline{T} + \overline{Y_1}P_1\overline{P_2})R\overline{T}$$

$$S = (P_2 + T + Y_1)R$$

$$En = (Y_0 + P_1)R$$

.
Analogamente per il contatore:

$$Q_0' = (\overline{Q_0}En + Q_0\overline{En})R$$

$$Q_1' = ((\overline{Q_1}Q_0 + Q_1\overline{Q_0})En + Q_1\overline{En})R$$

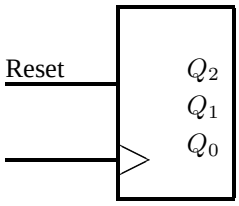
$$Q_2' = ((\overline{Q_1}(Q_0Q_1) + Q_2\overline{(Q_0Q_1)})En + Q_2\overline{En}))R$$

.

Esercizio 21 *Disegnate lo schema elettrico dettagliato del sistema di allarme. Provate a seguirne il funzionamento in alcuni casi tipici.*
Quando avrete imparato l'uso del programma Palasm potrete simularne il funzionamento al calcolatore

2.3 I contatori binari

I contatori binari sono sistemi sequenziali le cui N uscite (interpretate come numero binario senza segno) cambiano secondo le regole dell'aritmetica binaria, ossia vengono incrementate ad ogni impulso di clock. Una rappresentzione a blocchi di un contatore binario a 3 bit può essere la seguente:



Come si può notare abbiamo l'ingresso di clock, un ingresso di reset utile per forzare il contatore nello stato iniziale e le 3 uscite Q_0, Q_1, Q_2. Ovviamente le uscite seguiranno la sequenza:

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0
0	0	1
.	.	.
.	.	.
.	.	.

e così via all'infinito.

Esercizio 22 *Progettate un contatore a 3 bit usando i metodi standard*

Come risulta dal precedente esercizio la progettazione di un contatore binario è piuttosto noiosa e diventa rapidamente intrattabile, provate a progettare un contatore a 5 o 6 bit (anche senza reset) per rendervene conto, occorre perciò trovare un algoritmo generale che ci permetta di fare ciò in modo agile. Consideriamo la numerazione binaria e chiediamoci quando un bit cambia durante un operazione di incremento, la risposta è semplice, un bit cambia quando si ha il riporto dai bit meno significativi, ossia quando tutti i bit meno significativi sono uguali ad 1. In tutti gli altri casi i bit resteranno inalterati. Tradurre queste affermazioni in logica booleana è facile:

$$Q_n = \overline{Q_n}(Q_{n-1}Q_{n-2}\cdots Q_2Q_1Q_0) + Q_n\overline{(Q_{n-1}Q_{n-2}\cdots Q_2Q_1Q_0)}$$

L'espressione precedente è costituita da 2 blocchi, il primo complementa il valore di Q_n quando tutti i bit precedenti sono uguali ad uno (questo controllo si ottiene con il blocco delle Q poste in AND), il secondo blocco, che sfrutta la funzione NAND ha lo scopo di mantenere stabile il valore di Q_n in tutti gli altri casi.
Seguendo lo stesso approccio possiamo aggiungere la funzione reset: $Q_n =$ [La formula precedente] Reset.
Ossia è sufficiente mettere il nuovo valore di Q_n in AND con il segnale di reset per azzerarlo sbrigativamente.
Analogamente possiamo trattare il caso del contatore a decremento, sempre leggendo la tabella precedente, questa volta dal basso verso l'alto, noteremo che un bit cambia quando si ha un "prestito", ossia quando tutti i bit meno significativi sono uguali a zero.

$$Q_n = \overline{Q_n}\,\overline{(Q_{n-1} + Q_{n-2} + \cdots + Q_2 + Q_1 + Q_0)} + Q_n\,(Q_{n-1} + Q_{n-2} + \cdots + Q_2 + Q_1 + Q_0)$$

Anche in questo caso la formula è costituita di 2 parti ottenute seguendo la falsariga dei ragionamenti precedenti: la prima parte provvede a complementare il valore di Q_n quando tutti i bit meno significativi sono uguali a zero, la seconda parte mantiene fisso il valore di Q_n in tutti gli altri casi.
A questo punto possiamo aggiungere un selettore per disegnare un contatore capace di contare in entrambe le direzioni:

$$Q_n = [\text{Formula per incrementare}]D + [\text{Formula per decrementare}]\overline{D}$$

Esercizio 23 *Progettate un contatore a 5 bit capace di contare in entrambe le direzioni, di azzerarsi, di inizializzarsi ad un valore fornito da 5 opportuni ingressi e di arrestare il conteggio. Le 4 funzioni (conteggio avanti, indietro, inizializzazione e pausa) potranno essere scelte tramite 2 opportuni ingressi di selezione. Il reset si potrà ottenere come già mostrato. L'unico punto nuovo è l'inizializzazione, non dovrebbe essere difficile . . .*

2.4 I registri a scorrimento (Shift Register)

Un sistema sequenziale molto usato è il cosiddetto registro a scorrimento, normalmente chiamato con termine anglosassone "shift register". Lo schema di base è mostrato in figura 2.11. Come si vede è costituito da una sequenza più o meno lunga di flip–flop collegati sequenzialmente (4 nell'esempio), ad ogni impulso di clock il dato presente sull'ingresso a sinistra viene memorizzato dal primo flip–flop mentre i dati presente sui successivi flip–flop scorrono verso destra e vengono imagazzinati dal flip–flop successivo. É una specie di "coda" anzi, se vogliamo, possiamo considerare il registro a scorrimento come la realizzazione hardware della struttura dati della coda (chiamata anche FIFO ossia first in – first out per ovvie ragioni).

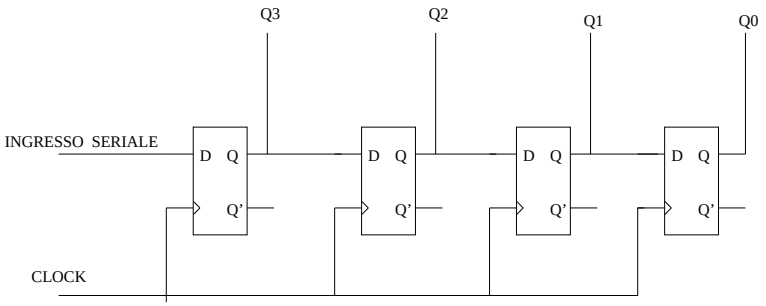


Figura 2.11: Schema di base di uno shift register a 4 stadi

I registri di questo tipo possono essere utilizzati in vario modo: come memorie temporanee (se i dati da memorizzare sono strutturati su parole di parecchi bit bisognerà immaginare di averne parecchi in parallelo), oppure per trasformare dati trasmessi serialmente (ossia dati che vengono trasmessi un bit alla volta) in dati "paralleli", sarà sufficiente collegare la sorgente dei dati all'ingresso seriale e aspettare un certo numero di impulsi di clock per ritrovarsi sui flip–flop i dati caricati pronti all'uso. Una estensione importante di questi oggetti è lo shift register bidirezionale (vedi figura 2.12) in cui opportuni selettori permettono di decidere se lo scorrimento dei dati avviene verso destra o verso sinistra.

Esercizio 24 *Disegnare uno shift register che abbia le seguenti funzioni: scorrimento a destra, a sinistra, caricamento parallelo, pausa. ossia aggiungete la possibilità di caricare in parallelo tutti i flip–flop e la pausa, ossia il mantenimento dei dati nel flip–flop in cui sono. Suggerimento: modificare lo schema di base del registro bidirezionale ampliando il blocco selettore.*

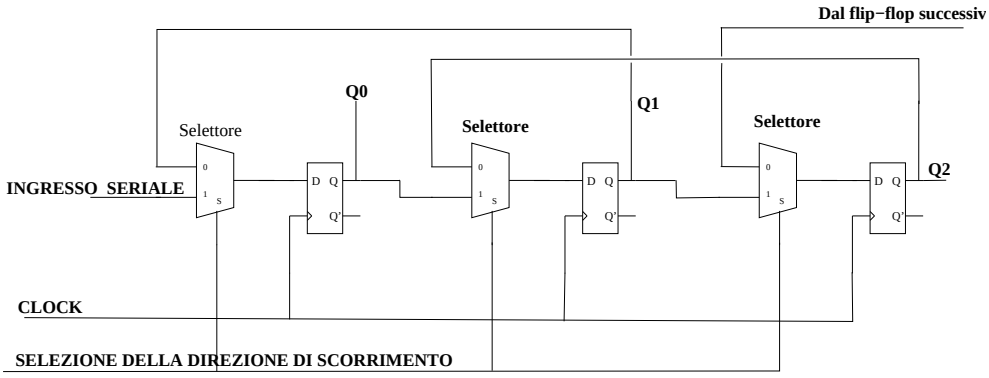


Figura 2.12: Schema di base di uno shift register bidirezionale a 3 stadi

2.5 I circuiti di memoria

Gran parte dei sistemi digitali è occupata da circuiti che hanno il compito di memorizzare delle informazioni in forma binaria, ad esempio nei moderni microprocessori più di metà dell'area di silicio disponibile è utilizzata per realizzare la memoria "cache" della CPU. Analizziamo pertanto i vari tipi di memoria e i principi fisici su cui si basa la memorizzazione dei dati, limitandoci a priori alle memorie a semiconduttore, escludendo quindi metodi pure estremamente interessanti e diffusi come la registrazione su supporto magnetico (hard disk e similari) oppure ottico (cdrom).
Le memorie possono essere classificate in base a diversi criteri: funzionalità, metodo di accesso e meccanismo di memorizzazione. La prima distinzione funzionale riguarda le memorie a sola lettura (ROM ossia Read Only Memory) e le memorie a lettura–scrittura.
Le ROM sono circuiti *combinatori* forniti di un certo numero di ingressi (n) (il cosiddetto indirizzo), e di un certo numero di uscite (i dati, di solito raggruppati in blocchi di 8 bit, ossia di byte), applicando agli ingressi una stringa di n bit è quindi possibile scegliere uno dei 2^n dati contenuti in memoria che può essere letto in uscita. I dati devono essere scritti in fase di fabbricazione e non possono essere modificati.
Le ROM sono usate per memorizzare informazioni che non devono essere modificate durante la vita dell'apparato, ad esempio il BIOS di un PC è spesso memorizzato in una ROM (una EPROM in realtà).
Esistono anche tecnologie per costruire ROM modificabili "poche" volte, si veda a questo proposito il capitolo successivo sui circuiti programmabili.
Le memorie a lettura–scrittura sono fornite degli stessi ingressi di indirizzo e uscite per i dati come le ROM, hanno in più un segnale, spesso denominato $R/\overline{W}$ che specifica se si intende scrivere un nuovo dato all'indirizzo specificato o leggere un dato inserito precedentemente, i dati possono essere scritti o letti usando le stesse connessioni per mezzo di porte bidirezionali con tecnologia tri-state precedentemente introdotta.
Il metodo più comune di accesso alle memorie a semiconduttore e quello casuale: ogni bit di memoria può essere letto o scritto indipendentemente dagli altri (in gergo sono delle RAM ossia Random access memory), esistono però memorie ad accesso sequenziale che realizzano in hardware la struttura software della coda, ossia la prima parola scritta è anche la prima che può essere letta, in gergo vengono chiamate FIFO ossia first in – first out.

2.5.1 Struttura di una memoria

Internamente le memorie sono solitamente costruite secondo lo schema di figura 2.13.

Come si può notare le celle di memoria (di qualunque tipo siano) sono organizzate a matrice quadrata, parte dell'indirizzo viene utilizzato per selezionare una riga, l'altra parte seleziona invece la colonna, ciò viene fatto per diverse ragioni, la più ovvia delle quali è che in questo modo i circuiti di decodifica risultano molto semplificati. La struttura della singola cella di memoria (e del circuito amplificatore che adatta i livelli logici utilizzati nella

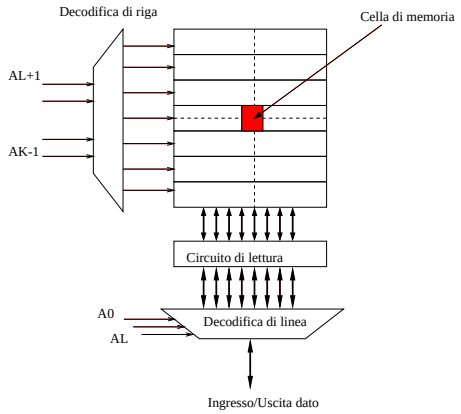


Figura 2.13: Schema a blocchi di una memoria generica

cella di memoria a quelli esterni) dipende dal tipo di memoria. Per le memorie a sola lettura è sufficiente un semplice diodo, in altri casi si usano transistor MOSFET a gate "flottante" (vedi oltre il capitolo sui dispositivi e circuiti logici e quello sui circuiti programmabili, PROM ed EPROM in particolare).
Le memorie RAM utilizzano sostanzialmente 2 tecnologie: le RAM cosiddette *statiche* usano un flip–flop, mentre le cosiddette dinamiche memorizzano il dato sotto forma di carica elettrica racchiusa in un condensatore:

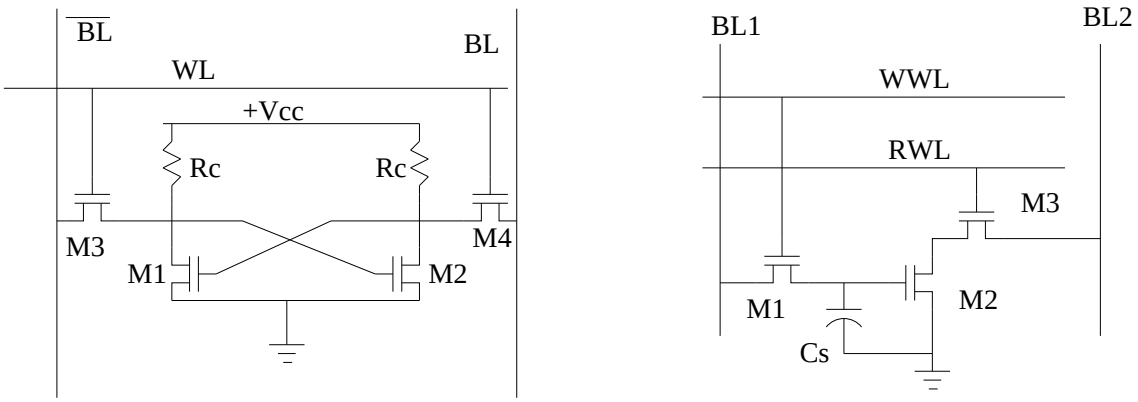


Figura 2.14: Schema della cella di memoria RAM (Statica e Dinamica)

È facile riconoscere nello schema a sinistra di figura 2.14 la classica struttura del flip–flop RS in cui i 2 invertitori sono semplicemente 2 MOSFET ad arricchimento che vengono forzati in uno dei 2 stati utilizzando la linea WL (selezione di riga) e le 2 linee BL e $\overline{BL}$ (selezione di colonna). Si notino le resistenze di carico R_c, per minimizzare la dissipazione di potenza esse devono avere valori elevatissimi, dell'ordine del TeraOhm $(1T\Omega = 10^{12}\Omega)$, però a questo punto per riuscire a leggere il bit senza perturbare la celletta ed in tempi ragionevoli, cioè pochi nanosecondi (ci si ricordi delle capacità parassite e dei circuiti RC), è indispensabile ridurre il valore delle resistenze di carico durante la fase di lettura, si può fare ciò abilitando contemporaneamente le 2 linee BL (la cosiddetta fase di precarica) per collegare il drain dei 2 mosfet a 2 resistenze di carico molto più basse ed al circuito di lettura.
Nelle RAM *dinamiche* invece l'informazione viene memorizzata come carica elettrica immagazzinata nella capacità di un mosfet. La cella di memoria dinamica mostrata nella figura 2.14 funziona nel seguente modo: il dato sotto forma di un appropriato livello di tensione viene applicato alla linea BL1, la linea di scrittura (WWL) viene alzata per portare il mosfet M1 in conduzione, subito dopo WWL viene riabbassata intrappolando la carica nel

condensatore CS, la lettura viene fatta alzando la linea RWL, sulla linea BL2 si potrà ora leggere un livello logico basso o alto a seconda se il transistor M2 è in conduzione o no. Poichè la carica in Cs tende a disperdersi a causa dell'isolamento non perfetto del silicio queste memorie devono essere periodicamente lette e riscritte (in pratica ogni pochi millisecondi!). Vi sono anche altri modi (più efficienti) per realizzare celle elementari che usano un solo transistor MOSFET invece dei 3 precedenti, poichè il loro funzionamento è piuttosto complesso e delicato non le tratteremo qui (anche se sono le più diffuse).

Capitolo 3

Dispositivi e circuiti logici (cenni)

NOTA: Le note relative al funzionamento dei dispositivi elettronici sono molto sommarie e semplificate, la precisione è stata a volte sacrificata alla compattezza. Non sono sufficienti, da sole, per preparare l'esame, devono essere integrate con gli appunti delle lezioni e con la parte relativa di [1].
In questo capitolo ci occuperemo delle tecnologie utilizzate per realizzare le funzioni logiche elementari. I simboli logici 0 e 1 sono solitamente rappresentati da livelli di tensione, uno standard molto utilizzato è quello chiamato TTL dal nome della famiglia logica di circuiti integrati che lo adottò per prima. Secondo questo standard 0 è rappresentato da una tensione compresa fra 0 e 0.4 V e 1 è rappresentato da una tensione compresa fra 2.4 e 5 V. Naturalmente esistono altri standard e quello appena citato comincia a diventare un po' obsoleto, è tuttavia quello che useremo in laboratorio. Si noti che esiste un intervallo proibito di tensioni: nessuna ingresso o uscita di circuito logico dovrà mai avere una tensione compresa fra 0.4 e 2.4 V, è solo consentito attraversare rapidamente questo intervallo durante un cambiamento di livello logico. Similmente gli ingressi dovranno essere collegati a livelli logici corretti, è perciò vietatissimo lasciarli scollegati, incidentalmente questo è uno degli errori più frequenti. Vediamo adesso come realizzare le funzioni logiche elementari, occorre perciò introdurre i dispositivi a semiconduttore che sono alla base dell'elettronica digitale moderna. Non darò qui alcuna spiegazione dei meccanismi fisici di funzionamento dei dispositivi, mi limiterò ad illustrarne il comportamento considerandoli in pratica come scatole nere aventi certe proprietà, questo è tutto quello che serve per capire il funzionamento delle reti logiche ed anche, almeno fino ad un certo punto, per progettarle.

3.1 Il diodo

Il diodo e' un dispositivo fornito di 2 soli terminali che ha la proprietà di lasciar passare la corrente in una sola direzione, i 2 terminali sono denominati anodo e catodo, applicando una tensione positiva all'anodo rispetto al catodo si ha un passaggio di corrente che cresce molto rapidamente all'aumentare di questa tensione, viceversa se l'anodo è polarizzato negativamente la corrente inversa è piccolissima (dell'ordine del nanoampere) e praticamente trascurabile. La legge che fornisce la relazione fra tensione e corrente ai capi del diodo è la seguente: $I = I_0(\exp(qV/KT) - 1)$. q è la carica dell'elettrone, V la differenza di potenziale fra anodo e catodo, K è la costante di Boltzmann e T la temperatura assoluta. Se V è negativa la corrente inversa è I_0, questo è un parametro legato a come è stato progettato il diodo. Come si vede, per una **polarizzazione diretta** la corrente aumenta esponenzialmente, questo implica che l'intervallo dei valori di V per cui la corrente non ne' piccolissima ne' eccessiva (ricordatevi dell'effetto Joule) è molto limitato, in prima approssimazione V può essere considerata costante $\approx 0.65V$.

3.2 Il transistor bipolare

Il transistor bipolare può essere visto visto come un amplificatore di corrente, se facciamo circolare una piccola corrente nel circuito di ingresso, nel circuito di uscita (opportunamente polarizzato) circola una corrente molto maggiore. I transistor sono dotati di 3 connessioni: emettitore (il terminale comune ai circuiti di ingresso e di uscita), base (ingresso), collettore (uscita). La relazione tra la tensione base–emettitore e la corrente di base è quella già vista del diodo, perciò la base sarà sempre leggermente positiva rispetto all'emettitore (Vbe circa 0.65 V). Una tipica configurazione (di principio) è mostrata in figura3.1:

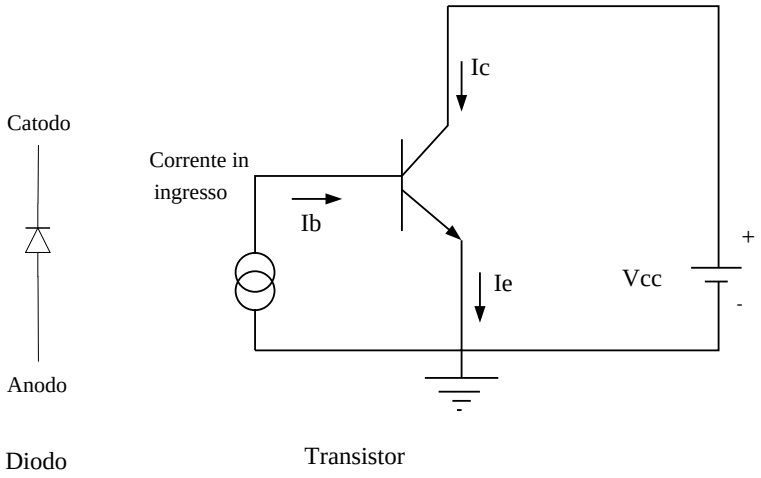


Figura 3.1: Simboli circuitali del diodo e del transistor (con polarizzazione)

Sulla sinistra è montato il generatore di corrente in ingresso collegato fra la base e la massa, il segnale amplificato si ottiene polarizzando *positivamente* il collettore rispetto all'emettitore, in queste condizioni fluisce una corrente Ic proporzionale alla corrente di base ma molto maggiore (ordine di grandezza circa 100 volte maggiore, dipende dal transistor).
Abbiamo così ottenuto che le variazioni (piccole) di tensione all'ingresso diventino variazioni di corrente di base (in prima approssimazione il circuito di ingresso, ossia la giunzione base–emettitore è equivalente ad un diodo polarizzato direttamente), queste variazioni di corrente vengono amplificate dal transistor e diventano variazioni di corrente di collettore, per poterle sfruttare semplicemente dobbiamo trasformarle in variazioni di tensione, per questa ragione tra collettore ed alimentazione positiva viene posta la cosiddetta resistenza di carico ai cui capi e' disponibile il segnale amplificato.
A questo punto è possibile disegnare i circuiti logici elementari, OR, AND, NOT.

Come si vede i circuiti OR ed AND possono essere costruiti con soli diodi, il funzionamento è molto semplice: esaminiamo ad esempio il circuito OR: l'uscita deve essere a "1" se almeno un ingresso è a "1". Infatti se tutti gli ingressi sono a 0 Volt, l'uscita sarà pure a 0 Volt (non c'è alcun generatore di tensione !), se almeno un ingresso ha una tensione positiva e molto maggiore della caduta tipica ai capi di un diodo (circa 0.65 V) questa tensione si propagherà in uscita attraverso il diodo ad essa collegato che risulta polarizzato direttamente. Si noti che i diodi il cui ingresso è a 0 V sono polarizzati inversamente e quindi non conducono, isolando gli ingressi fra di loro. Si noti altresì che il valore del logico 1 in uscita risulta abbassato di circa 0.65 V, questo impedisce di collegare in cascata molti circuiti di questo tipo.
La spiegazione del funzionamento del circuito AND è lasciata come *esercizio*.
Il funzionamento del circuito NOT, detto anche invertitore è molto semplice: se l'ingresso è a 0V non fluirà alcuna corrente nel circuito base–emettitore, quindi anche la corrente di collettore sarà nulla e la tensione in uscita pari a quella di alimentazione, se invece avremo una corrente di base sufficiente la corrente di collettore sarà limitata

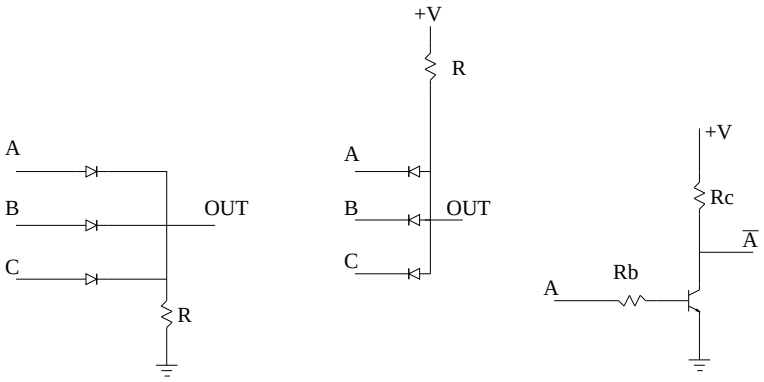


Figura 3.2: Circuiti OR AND NOT

dalla resistenza di carico posta in serie ad esso e la tensione in uscita sarà molto vicina a zero, tipicamente 0.1 V circa, in queste condizioni (corrente di collettore limitata esternamente e tensione emettitore-collettore quasi nulla) il transistor si dice saturato.

3.3 Il transistor ad effetto di campo MOSFET

Il transistor più utilizzato nei circuiti logici moderni è il cosiddetto MOSFET. La sua struttura è mostrata in fig.3.3: su una piastrina di silicio debolmente drogato P (chiamata substrato) vengono formate 2 regioni di tipo N chiamate "source" e "drain", sopra queste regioni viene fatto crescere un sottile strato isolante di biossido di silicio (SiO_2), ricoperto successivamente di uno strato conduttore, questo strato costituisce il cosiddetto "gate".

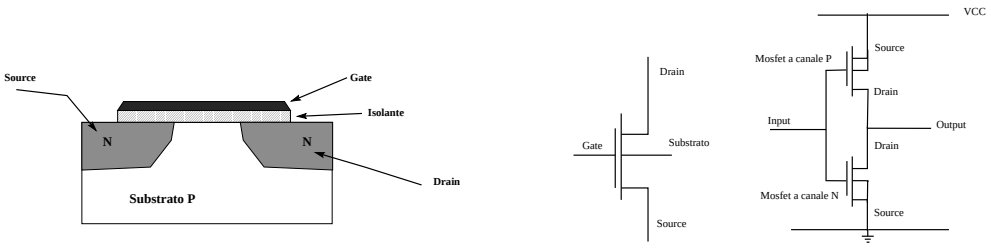


Figura 3.3: Mosfet a canale N (di profilo), suo simbolo circuitale ed inverter CMOS

Normalmente il source è collegato, insieme al substrato, al polo negativo, il drain, attraverso un carico, al polo positivo; il gate è l'elettrodo di controllo, se applichiamo ad esso un potenziale sufficientemente alto si ha un passaggio di corrente tra source e drain tanto maggiore quanto maggiore è questo il potenziale. In pratica ciò significa che se la differenza di potenziale tra source e drain è sufficiente il mosfet si comporta come un interruttore controllato in tensione (confronta con il transistor bipolare che viene utilizzato come interruttore controllato in corrente).
La ragione di questo comportamento è facile da capire a livello qualitativo: in mancanza di una differenza di potenziale tra source e gate non si può avere passaggio di corrente tra source e drain, infatti tra il substrato ed il drain si trova una giunzione P–N, se il drain è positivo, questa giunzione è polarizzata inversamente, quindi non si ha conduzione.

Se ora applichiamo un potenziale positivo al gate creiamo un campo elettrico tra gate ed il silicio sottostante che attirerà le cariche negative presenti (anche nel silicio drogato P vi sono portatori negativi, naturalmente molto pochi), se il campo elettrico è sufficientemente alto queste cariche formeranno un sottile strato sotto al gate che collegherà sorce e drain, ossia si formerà un "canale" costituito di silicio in cui i portatori di carica sono elettroni, ossia di tipo N, pertanto source e drain costituiranno le estremità di una struttura tutta di tipo N in cui la corrente può scorrere liberamente.
Una analisi quantitativa del dispositivo permette di calcolare la relazione tra la corrente di drain e le tensioni applicate:
Per tensioni source–drain basse (molto minori della tensione source–gate) si ha:

$$I_D = k_n((V_{GS} - V_T)V_{DS} - \frac{V_{DS}^2}{2})$$

k_n è una costante che dipende dal materiale utilizzato e dalla geometria del mosfet, per un dispositivo tipico, con spessore dell'isolante pari a 20 nm, $K_n = 80\mu A/V^2$.
Al crescere della tensione drain–source non si può più trascurare questa caduta di tensione rispetto alla tensione di gate, il campo elettrico sotto al gate non sarà più costante, ma diminuirà spostandosi dal source verso il drain, aumentando così la resistenza del canale.
In queste condizioni il flusso di corrente tenderà ad un valore costante, indipendente cioè dalla tensione source–drain:

$$I_D = \frac{k_n^{'}}{2}\frac{W}{L}(V_{GS} - V_T)^2$$

In analogia con i 2 tipi di transistor: npn e pnp, è possibile costruire anche mosfet a canale P, invertendo tutti i drogaggi, ossia, partendo da un substrato N, creare source e drain drogando P, ecc.
Un mosfet a canale P avrà il source collegato al polo positivo ed il drain collegato al polo negativo ed entrerà in conduzione quando il gate sarà sufficientemente negativo rispetto al source.
I 2 tipi di transistor permettono di costruire in modo molto semplice un invertitore (vedi fig:3.3), se la tensione in ingresso è circa zero solo il mosfet a canale P è in conduzione e quindi l'uscita sarà ad un livello logico alto, viceversa, se la tensione in ingresso è "alta" (prossima al valore della tensione di alimentazione), il mosfet a canale N sarà acceso e il mosfet a canale P spento, per cui la tensione in uscita sarà prossima a zero.
Questa tecnologia prende il nome di CMOS (Complementary Mosfet) ed è attualmente molto usata, offre infatti molti vantaggi:

- Tecnicamente semplice da costruire, si utilizzano solo mosfet, niente resistenze o diodi.
- Lo spazio richiesto sul silicio per costruire un invertitore è molto limitato, consentendo di costruire circuiti integrati molto complessi.
- Il consumo statico di potenza (ossia senza variazioni degli ingressi) è praticamente zero perchè uno dei 2 mosfet è sempre spento, si ha passaggio di corrente solo durante le transizioni per caricare–scaricare le capacità parassite (e quindi il consumo del sistema è proporzionale alla frequenza di clock).

Usando la tecnologia CMOS è particolarmente semplice progettare anche porte logiche 3–state (vedi 1.7). Come si ricorderà in una porta 3–state è possibile spegnere tutti i transistor dello stadio di uscita al fine di disconnetterla completamente dal circuito a cui è collegata, questo si può ottenere ad esempio come mostrato in fig. 3.4: attraverso i 2 segnali complementari En e $\overline{En}$, che provvedono ad accendere o spegnere contemporaneamente i 2 MOS collegati (uno a canale N ed uno a canale P).

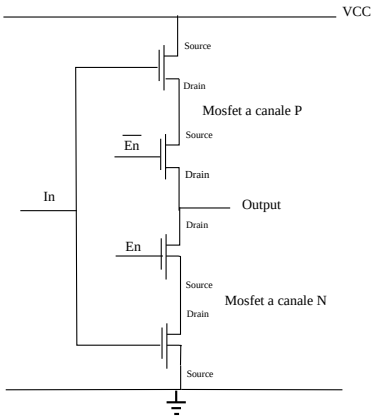


Figura 3.4: Invertitore 3–state in tecnologia CMOS

Analogamente è possibile progettare le altre funzioni logiche, ad esempio in fig. 3.5 è mostrato un NOR a 2 ingressi.

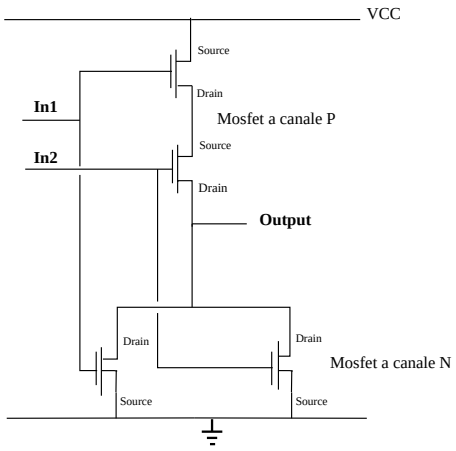


Figura 3.5: NOR a 2 ingressi in tecnologia CMOS

Esercizio 25 *Disegnate un circuito OR a 3 ingressi in tecnologia CMOS.*

Esercizio 26 *Disegnate un circuito NAND a 2 ingressi in tecnologia CMOS*

Capitolo 4

I circuiti programmabili

La costruzione di un sistema digitale anche non molto complesso richiede un numero elevato di funzioni logiche elementari (and, or,not) e di blocchi precostituiti (flip–flop, ALU, comparatori digitali, registri, shift–register, contatori, ecc) collegati nei modi più vari, è una situazione molto diversa da quella dell'elettronica analogica dove esistono gli amplificatori operazionali che costituiscono il mattone fondamentale per la realizzazione di moltissimi circuiti.
In queste pagine vogliamo introdurre il concetto di circuito programmabile, analizzare la tecnologia che ne permette la realizzazione, avere una panoramica delle architetture più comuni e delle possibili applicazioni. La trattazione è volutamente semplice, anche a costo di lievi inesattezze.
Si intende per circuito programmabile (PLD) un dispositivo (digitale) integrato che può essere modificato dall'utente al fine di realizzare la rete logica desiderata. Il costruttore fornisce cioè un dispositivo generico contenente funzioni logiche 'elementari' che possono essere interconnesse dal progettista a piacere. Le funzioni logiche elementari da cui partire possono essere,ad esempio,porte logiche, flip-flop, rom.
Iniziamo un breve esame delle tecnologie che consentono la creazione di collegamenti fra due punti all'interno del circuito integrato.
Storicamente i primi circuiti programmabili apparsi sul mercato furono le PROM, ossia le memorie programmabili a sola lettura. Lo scopo di tali circuiti è semplice: permettere al progettista/costruttore di sistemi digitali di costruirsi rapidamente e a basso costo le ROM di cui può avere bisogno. In effetti prima dell'avvento delle PROM l'unica possibilità era quella di rivolgersi ad un fabbricante di circuiti integrati, specificando il contenuto della ROM e chiedere la preparazione del circuito integrato con le specifiche richieste, questa operazione è estremamente costosa e lunga, giustificata solo per oggetti prodotti in grande serie, da evitare nella fase di costruzione del prototipo, dove ogni errore o modifica comporta danni sia in termini di tempo che di denaro. Le PROM sono invece memorie a sola lettura in cui una semplice procedura permette di alterare stabilmente le connessioni interne al fine di memorizare i dati voluti. Vediamo uno schema generale di ROM:
Poichè il numero di connessioni è enorme lo schema elettrico deve essere presentato in modo appropriato:
i triangoli in alto a sinistra dello schema sono dei "buffer", ossia degli oggetti che si limitano a separare il mondo esterno dalla circuiteria interna al circuito programmabile, sono forniti di 2 uscite per avere sia il segnale originario, sia la copia negata.
La colonna di AND in centro è costituita di molti blocchi uguali, l'unica linea orizzontale presente su ogni funzione logica in realtà è un abbreviazione per un AND con molti ingressi, le connessioni verso i segnali di ingresso è indicata dalle crocette.
Analoga simbologia viene utilizzata per la riga di OR in basso a destra, anche qui le funzioni hanno molti ingressi e le connessioni verso gli AND è indicata dalla crocetta.
La ROM è quindi costituita di 2 parti: una matrice di AND che ha lo scopo di decodificare l'indirizzo ed una matrice di OR che provvede a trasferire in output i bit dell'indirizzo selezionato, è questa seconda matrice di OR che deve essere modificata per alterare i dati memorizzati.

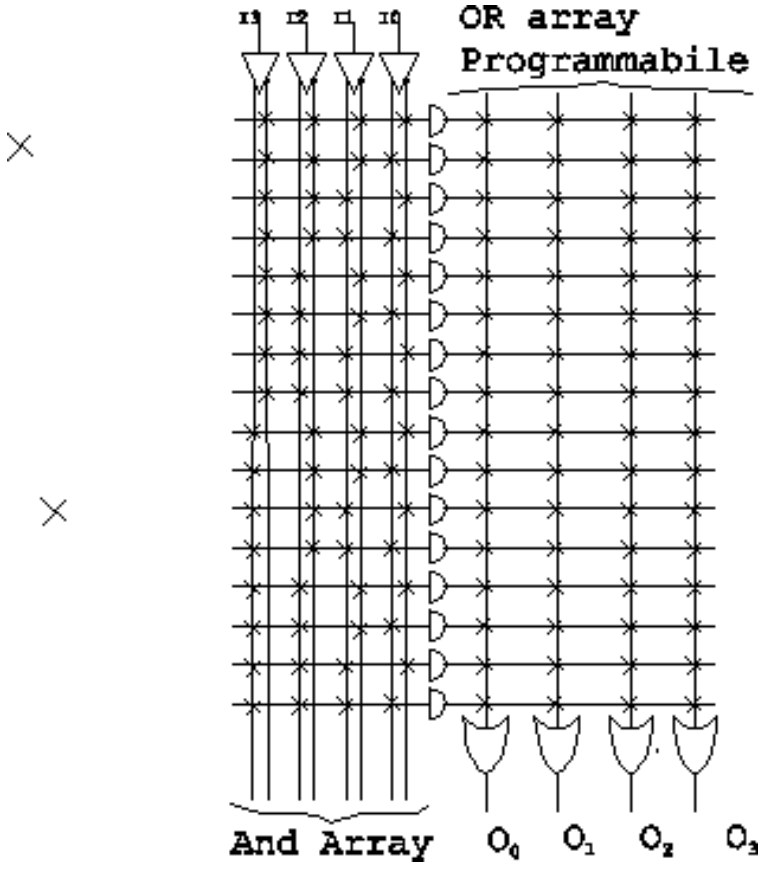


Figura 4.1: Schema semplificato di un circuito ROM

Una piccola parte di questa matrice è mostrata qui sotto: I diodi sono posti fra le linee di indirizzo decodificate

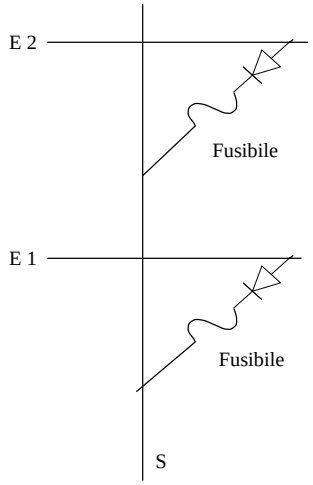


Figura 4.2: Circuito OR a diodi con connessione programmabile

(e1 ed e2) e l'uscita S, in serie ai diodi le linee ondulate rappresentano dei "fusibili" ossia delle connessioni molto fragili che possono essere distrutte selettivamente da un passaggio eccessivo di corrente che li vaporizza. Inizialmente quindi tutte le connessioni sono presenti ed il diodo garantisce semplicemente il flusso unidirezionale della corrente per evitare interferenze fra e1 ed e2 che si verificherebbero se i diodi fossero sostituiti da semplici conduttori. Il fusibile è costituito da una breve connessione (3-5 micron) larga circa 1 micron di una lega a base di titanio e tungsteno che può essere distrutta da una corrente circa 6 volte maggiore di quella normale di funzionamento. Il circuito viene cioè acquistato fornito di tutte le connessioni, prima di essere utilizzato deve essere inserito in un "programmatore" che, sotto la guida di un computer (tipicamente un PC), provvede ad applicare ai terminale dell'integrato gli impulsi di lunghezza ed ampiezza adeguata per la distruzione dei fusibili desiderati.
La tecnologia dei fusibili può essere variamente modificata, vale la pena di citare la tecnica dell' "antifusibile". Come il nome suggerisce l'antifusibile è una connessione "mancante" che può essere creata da un opportuno impulso di tensione. Il principio fisico su cui si basa è quello della perforazione di un sottilissimo (90 angstrom) strato di dielettrico isolante (nitruro di silicio) con l'applicazione di un breve impulso (circa 1 millisecondo) di circa 16 Volt di ampiezza.
Attualmente la tecnica più diffusa è però quella relativa alla programmazione dei transistor MOS, esistono sostanzialmente 3 tecnologie importanti:

- Memorie EPROM con transistor MOS "a gate flottante": Ricordiamo brevemente il funzionamento di un transistor MOS "normale" : Esso è costituito da una piastrina di silicio opportunamente drogata, la conduzione sulla sua superficie viene controllata da un elettrodo (gate o porta) che modula la conducibilità del semiconduttore, in termini intuitivi si può pensare che la carica elettrica presente sul gate influenzi il passaggio della corrente nel silicio attraverso un effetto elettrostatico.

  Nel caso di nostro interesse il transistor mos è fornito di 2 elettrodi di controllo (gate) sovrapposti, uno funziona come nel caso precedente, l'altro è scollegato dal resto del circuito ed inizialmente non ha alcun effetto sul passaggio di corrente nel MOS, se tuttavia si inietta una carica negativa sul secondo gate che (sottolineo) è completamente circondato da isolante (gate flottante), il campo elettrico generato dalla carica intrappolata impedisce ai circuiti di controllo di agire sul transistor e di portarlo in conduzione.

  In questo modo viene immagazzinato un bit di informazione. Per cancellare l'informazione, il circuito integrato deve essere sottoposto ad una intensa sorgente di luce ultravioletta che ha la proprietà di rendere debolmente conduttore il silicio e quindi permette agli elettroni accumulati nel gate flottante del MOS di

defluire verso gli elettrodi e ripristinare la condizione iniziale. A tale scopo i circuiti sono forniti di una finestrella di quarzo che dopo la programmazione deve essere ricoperta da una etichetta opaca per evitare che la luce ambiente che sempre contiene una componente UV alla lunga cancelli i dati. La scrittura di un dato richiede di solito alcuni millisecondi per ogni parola indirizzata.

- Memorie EEPROM: simili alle precedenti, la cancellazione viene però effettuata da opportuni impulsi elettrici che per effetto tunnel fanno rifluire la carica elettrica intrappolata verso gli altri elettrodi del transistor, svuotando il gate flottante, la scrittura di un dato richiede di solito alcuni millisecondi per ogni parola indirizzata.
- Memorie FLASH: simili alle EEPROM dal punto di vista dell'utilizzatore, anche il principio di funzionamento è molto simile, la differenza è sostanzialmente tecnologica, sono caratterizzate da tempi di scrittura più rapidi (pochi microsecondi)

Il numero di scritture e cancellazioni per tutte e 3 le tecnologie è comunque *LIMITATO*, nel caso più favorevole (FLASH) è attualmente dell'ordine del milione o più. È naturalmente sempre possibile memorizzare i dati in celle SRAM, cioè in flip-flop che hanno il vantaggio di poter essere letti e riscritti infinite volte ma hanno il difetto non trascurabile di dimenticare tutto se manca l'alimentazione, perciò talvolta l'integrato viene fornito con una pila al litio per garantire la conservazione dei dati.
Se osserviamo lo schema generale di una PROM vediamo che sono distinguibili 2 parti distinte: una prima matrice di AND che ha la funzione di decodificare tutti gli indirizzi ed una matrice di OR in cui è materialmente scritta l'informazione, solo la matrice di OR è programmabile dall'utente.
Ci si può chiedere se non sia possibile rendere programmabile **anche** la matrice di AND, in effetti questo è stato fatto ed ha dato origine ad una nuova famiglia di componenti programmabili che vanno sotto vari nomi commerciali (PAL, GAL, PLA, LCA, FPGA, ecc).
L'idea di base iniziale fu quella di rendere programmabile la matrice di AND e fissare la matrice di OR, si aveva a questo punto un circuito combinatorio abbastanza generale, che poteva funzionare anche in modo sequenziale grazie alla possibilità di riportare alcune uscite agli ingressi (feedback).
Il risultato, a questo livello, non è però molto dissimile da una PROM opportunamente programmata, anzi, poichè la matrice di AND è di dimensioni molto più limitate, apparentemente il circuito è soltanto una versione più limitata (e magari economica) di quanto già visto, tuttavia è importante notare che questi circuiti sono solitamente molto più veloci delle PROM. Ciò che però rende estremamente interessanti questi circuiti è la possibilità di integrarvi anche componenti sequenziali, ossia flip-flop: la configurazione più semplice a cui si arriva è cioè un insieme di funzioni combinatorie i cui risultati confluiscono in un registro di flip-flop di tipo D, le uscite dei flip-flop sono quindi disponibili in uscita, ma possono anche essere riportate all'ingresso in base al concetto di feedback già visto. Il risultato è insomma un sistema che può essere facilmente configurato come automa finito assolutamente generale.
Come esempio forniamo qui (fig. 4.3) lo schema logico di una PAL 16R6: Con questo tipo di architettura è impossibile costruire funzioni solo combinatorie degli ingressi e dello stato interno, questo difetto è superato dalle PAL "versatili" , ad esempio la 22V10 in cui esistono selettori programmabili che permettono di scavalcare il flip-flop e portare in output direttamente il risultato della funzione combinatoria (fig. 4.4).

I circuiti "versatili" di questo tipo sono sufficientemente generali da poter essere utilizzati per la costruzione di un qualsiasi processore ed hanno l'innegabile vantaggio che la correzione di un errore di progetto può essere effettuata in modo semplice e rapido, spesso senza alterare il circuito stampato, con ovvi vantaggi, specialmente durante la fase di costruzione del prototipo.
Naturalmente i circuiti visti fin'ora sono piuttosto semplici, la suddetta 22V10 contiene 10 flip–flop, per costruire un sistema completo possono essere necessari da alcune centinaia o fino a molte migliaia di "blocchi logici" elementari, fortunatamente l'evoluzione della tecnologia ha permesso di inserire nello stesso circuito molte più funzioni del tipo già visto, ma a questo punto si offrono sostanzialmente 2 alternative architetturali:

- L'inserimento di "molti" circuiti piuttosto semplici, ossia la messa a disposizione di centinaia o migliaia di elementi costituiti sostanzialmente da 1-2 flip-flop ed alcune funzioni combinatorie arbitrarie (programmabili) con un numero di ingressi limitato (minore di 10 ad es.) La programmazione può essere effettuata

in vari modi, in alcuni casi il dispositivo può essere programmato una sola volta bruciando dei fusibili, in altri casi il circuito può essere riprogrammato diverse volte, arrivando alla famiglia "LCA" di XILINX che può essere riprogrammata infinite volte poichè la configurazione interna è mantenuta dentro una SRAM che viene caricata in fase di inizializzazione. Quest'ultimo caso permette al limite di avere un circuito che viene riconfigurato a seconda delle esigenze del momento del sistema per effettuare operazioni diverse. Il progetto può essere sviluppato sia pianificando nel dettaglio l'uso di ogni singola cella elementare con ovvia difficoltà e lentezza di progetto, ma altrettanto ovvia ottimizzazione del circuito, sia disegnando lo schema in modo tradizionale e usando librerie per accedere ai blocchi funzionali di cui si può aver bisogno (contatori, shift-register, ecc) o usando linguaggi di alto livello (ad es. Verilog o VHDL). Il difetto maggiore di questo secondo tipo di approccio è connesso con lo scarso controllo che il progettista ha sulla traduzione della descrizione simbolica in un circuito digitale, è perciò difficile, ad esempio, prevedere i tempi di propagazione del segnale all'interno del circuito, infatti lo schema o la descrizione simbolica del circuito dovrà essere compilata da un sistema di CAD che cercherà di far corrispondere nel modo migliore possibile i componenti disegnati dal progettista con quelli disponibili e di connetterli cercando i percorsi più corti, ("placement" e "routing"), può perciò accadere che i ritardi di propagazione interni, sommandosi, provochino malfunzionamenti difficili da correggere vista anche l'impossibilità di misurare, ad esempio con un oscilloscopio, cosa accade veramente dentro al sistema[1].

Questo tipo di problemi viene affrontato simulando al calcolatore il funzionamento del sistema per verificare che la realizzazione a cui si è pervenuti sia accettabile.

- L'inserimento di circuiti tipo PAL22V10 o similari in numero più limitato, con la possibilità di interconnetterle attraverso una matrice centrale di commutazione (CPLD) (fig. 4.5). In questo caso è molto più facile tenere sotto controllo i ritardi poichè i blocchi logici e la matrice di interconnessione danno luogo a ritardi ben noti, per cui il tempo di propagazione totale del segnale è sostanzialmente la somma all'interno della matrice più il ritardo intrinseco del blocco.

[1] Nota: la propagazione del segnale elettrico all'interno del circuito integrato avviene ad una velocità molto inferiore a quella della luce, infatti, a causa della resistenza dei conduttori (molto spesso silicio opportunamente drogato) e delle capacità parassite il conduttore deve essere trattato come un complicato circuito RC (per la precisione come una linea di trasmissione dissipativa)

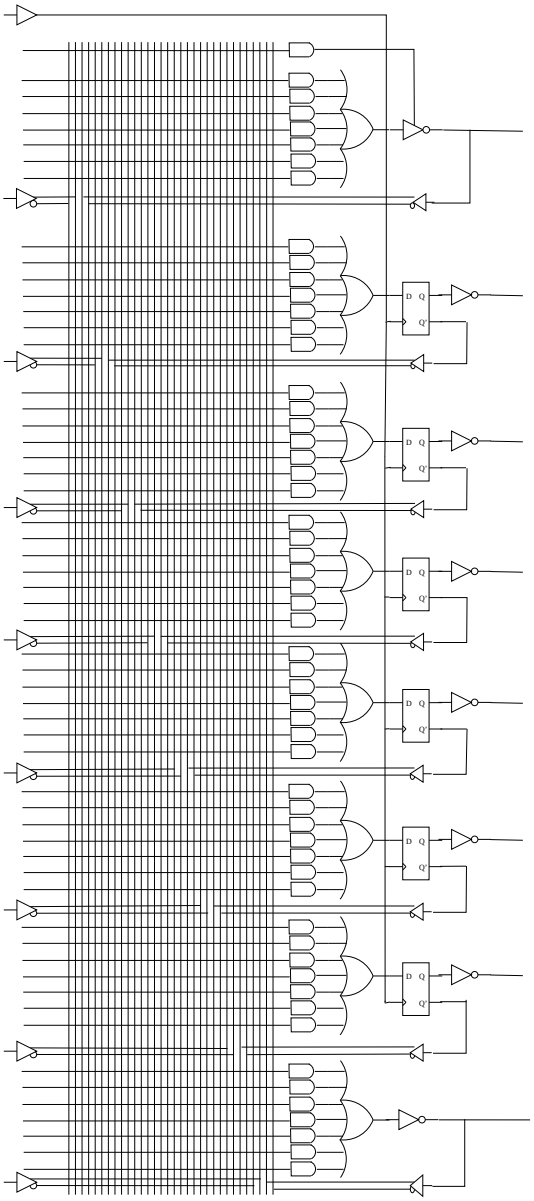


Figura 4.3: Circuito programmabile “16R6”

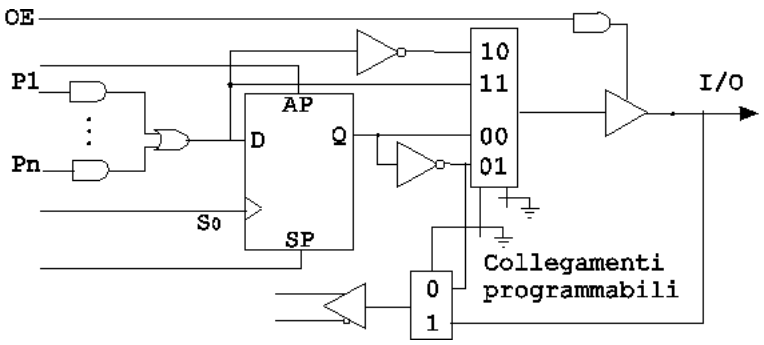


Figura 4.4: Blocco elementare di una “PAL” versatile come la 22V10

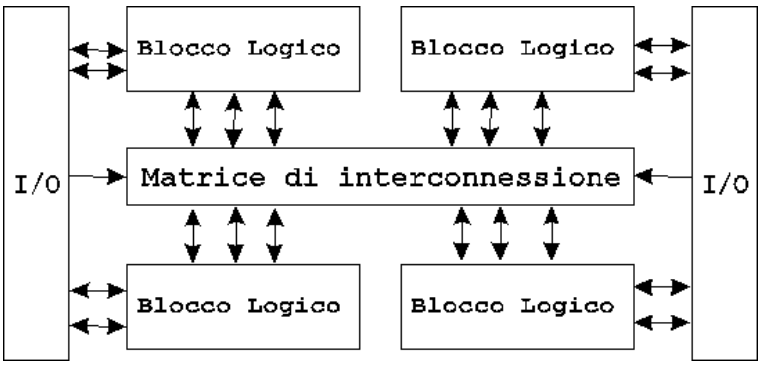


Figura 4.5: Schema a blocchi di un tipico “CPLD”

Capitolo 5

Conversione Analogico–digitale e digitale–analogica

I sistemi digitali visti fin'ora sono caratterizzati dall'avere ingressi e uscite binarie, se vogliamo elaborare grandezze variabili con continuità (ad esempio la tensione fornita da un microfono o da una termocoppia) dobbiamo trovare un modo per tradurle in sequenze di bit che possano essere successivamente processate, inoltre avremo bisogno di un oggetto capace di tradurre una sequenza di bit (cioè un numero binario) in una tensione per poter pilotare amplificatori, attuatori meccanici, ossia, in generale, dispositivi il cui ingresso sia una tensione o corrente variabile in modo pressochè continuo.
I dispositivi che risolvono questi problemi prendono il nome di convertitori analogico–digitali (ADC ossia analog to digital converter) e convertitori digitali–analogici (DAC digital to analog converter). I primi (ADC) hanno un ingresso analogico (di solito una tensione) ed una uscita digitale che rappresenta su N bit un numero proporzionale al valore della tensione in ingresso. I secondi (DAC) hanno invece un ingresso digitale su N bit ed un'uscita analogica su cui è presente una tensione proporzionale al numero presente in ingresso.

5.1 I Digital to analog converter

Iniziamo ad esaminare questi ultimi, la relazione funzionale tra l'ingresso digitale n e la tensione analogica in uscita V_a è legato al tipo di codifica adottata, di solito viene usata la classica rappresentazione binaria, per cui $V_a = q \cdot n$ dove q è il cosiddetto passo di quantizzazione, ossia la minima variazione di tensione ottenibile in uscita e ho indicato con n il numero rappresentato dagli N bit in ingresso.. La relazione è lineare, la tensione in uscita è proporzionale a n. Analizzando le caratteristiche tecniche dei DAC la prima cosa che si nota è che evidentemente la tensione in uscita varia a salti (poichè l'ingresso è un numero intero) e quindi, fissato l'intervallo entro cui si vuole generare V_a, per avere una risoluzione elevata dobbiamo rappresentare n con un numero sufficientemente elevato di bit. È possibile costruire DAC da un minimo di 6/8 bit fino ad un massimo di almeno 16 bit in oggetti commerciali facilmente reperibili, faccio notare che un dac con una risoluzione di soli 8 bit permette di controllare l'uscita con una risoluzione di una parte su $2^8 = 256$, mentre un dac a 16 bit consente una risoluzione di una parte su $2^{16} \approx 65500$ quindi elevatissima.
Altra caratteristica importante dei DAC è la velocità di conversione, ossia l'intervallo di tempo che passa dall'applicazione del nuovo codice binario ingresso al momento in cui l'uscita raggiunge il valore desiderato *entro la precisione richiesta*, ossia, di solito entro la risoluzione del sistema. La velocità di funzionamento del DAC è quindi limitata come al solito dalle costanti di tempo RC del sistema ed evidentemente un DAC ad elevata risoluzione avrà bisogno di un maggior numero di costanti di tempo per raggiungere il valore asintotico entro la precisione richiesta e sarà quindi inevitabilmente più lento. A seconda delle applicazioni il tempo di conversione può variare tra circa un nanosecondo (DAC a pochi bit con elevata dissipazione di potenza per poter ridurre le costanti di tempo) e diversi millisecondi (DAC a 20 bit a bassissimo consumo).

È naturalmente molto importante che la caratteristica ingresso–uscita del DAC sia lineare, ossia che la tensione in uscita sia rappresentata da una legge del tipo $V_{out} = q * n$. Purtroppo questo ovviamente non accade, è impossibile che i componenti impiegati (resistenze, deviatori, operazionali, ecc.) siano ideali, ad esempio le resistenze si discosteranno dal valore nominale, i deviatori avranno una resistenza non nulla in chiusura e saranno solo approssimativamente equivalenti ad una resistenza infinita se aperti, gli amplificatori operazionali avranno sempre una tensione di offset ed un guadagno limitato. La relazione ingresso–uscita finisce per diventare una relazione complicata, rappresentabile con un'equazione del tipo:

$$V_{out} = V_0 + q' * n + q_1 * n^2 + q_2 * n^3 + \cdots$$

dove V_0 rappresenta una tensione di offset, $q' \neq q$ l'errore sul "guadagno" e gli altri termini q_1, q_2 gli errori di linearità del DAC che quindi è opportuno che siano piccolissimi, idealmente zero.
A questo proposito la linearità di un DAC viene specificata dal produttore fornendo 2 quantità: la *linearità integrale e differenziale*. La linearità integrale è definita come il massimo scostamento fra la caratteristica reale del DAC e quella ideale, è insomma il massimo errore che ci può capitare nell'uso del dispositivo. La nonlinearità differenziale è invece una misura della uniformità degli incrementi di tensione tra un valore in ingresso ed il successivo, ossia un DAC ideale dovrebbe aumentare l'uscita di un valore costante se l'ingresso viene incrementato di 1, questo non accade e quindi viene specificato di quanto questo incremento è diverso da quello ideale, si noti che una non linearità differenziale maggiore di un "quanto" (cioè dell'incremento minimo) implica che la caratteristica del DAC può essere non monotona, ossia all'aumentare di n può accadere che l'uscita temporaneamente cali, per questo motivo i produttori sottolineano sempre se l'output del DAC è monotono.

5.2 DAC con rete a scala

La realizzazione dei DAC è fondata sul principio che è possibile usare variabile binarie per azionare interruttori analogici (ad es. dei MOSFET) e controllare correnti che, opportunamente sommate, possono fornire una corrente proporzionale ad n, la corrente può essere facilmente trasformata in una tensione ad essa proporzionale.
Lo schema di base per un DAC a 5 bit è mostrato in figura:5.1

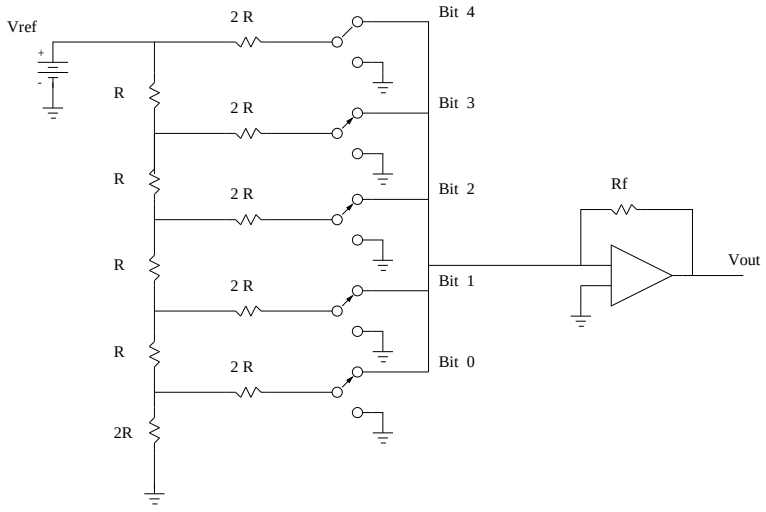


Figura 5.1: DAC a 5 bit

È costituito di 4 blocchi:

- Generatore di tensione di riferimento (Vref), la tensione in uscita sarà proporzionale a questo generatore, è perciò importante che sia ben nota e stabile.

- Una rete di resistenze costituita da 2 soli valori : R e 2R, il valore esatto delle resistenze non è importante, solo i rapporti relativi devono essere stabili e precisi.
- Gli interruttori che hanno la funzione di deviare la corrente fornita dalla rete resistiva o verso massa o verso il nodo sommatore.
- Il circuito sommatore delle correnti realizzato (ad esempio) intorno ad un operazionale in configurazione invertente per avere bassissima impedenza di ingresso.

Il funzionamento del sistema può essere facilmente compreso guardando la figura:5.2

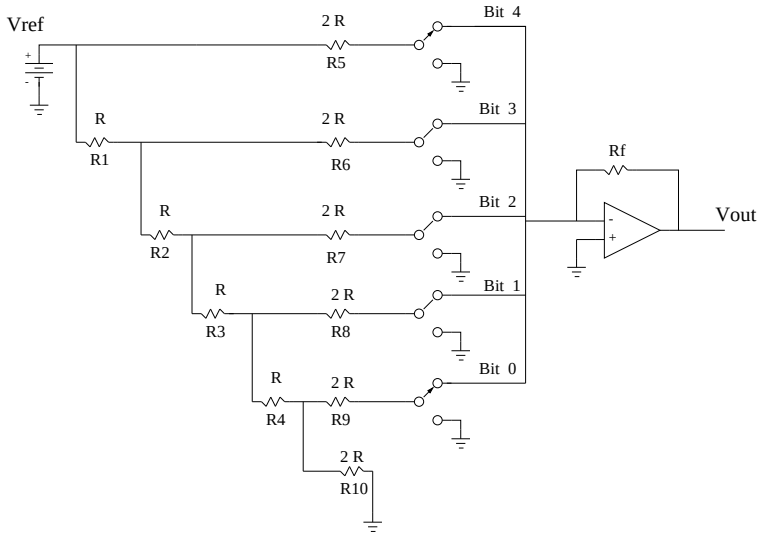


Figura 5.2: Suddivisione delle correnti in una rete R–2R

Si noti prima di tutto che le resistenze R5, R6, R7, R8, R9, R10 sono collegate a destra a un punto di massa, reale se il deviatore è rivolto verso il basso, massa virtuale creata dall'amplificatore operazionale in configurazione invertente se rivolto verso l'alto.
Guardiamo adesso le resistenze R9 ed R10, hanno lo stesso valore e sono collegate in parallelo (per il discorso precedente sulle masse), saranno quindi attraversate dalla stessa corrente, diciamo I_0, questa corrente fluisce verso l'operazionale se il deviatore è rivolto verso l'alto (ossia il bit meno significativo (bit 0) della parola è 1) dando origine ad una tensione in uscita $Vout = -R_f I_0$. Quindi R9 ed R10 sono equivalenti ad una unica resistenza di valore R, percorsa da una corrente $2\ I_0$, perciò anche R4 sarà percorsa dalla stessa corrente $2\ I_0$ ed, essendo posta in serie a $(R9 \| R10)$, questo blocco di 3 resistenze sarà equivalente ad una unica resistenza 2 R percorsa da una corrente $2\ I_0$.
Adesso possiamo ripetere il ragionamento precedente per R8 che, essendo collegata in parallelo al blocco precedente ed avendo lo stesso valore 2 R, sarà percorsa dalla stessa corrente, ossia $2\ I_0$, che, attraverso il deviatore fornirà il contributo relativo al bit 1, chiamiamo I1 questa corrente che, se il bit 1 è uguale a 1, fluirà verso l'operazionale fornendo un contributo di tensione in uscita pari a $-R_f\ I1$.
Ripetiamo il ragionamento per R7 che si trova in parallelo al blocco precedente $R3 + (R8 \| (R4 + (R9 \| R10)))$ ottenendo che R7 è percorsa da una corrente $I_2 = 4\ I_0$. Il sistema funziona insomma dividendo la corrente iniziale fornita da Vref in 2 parti uguali ad ogni passo della rete sfruttando le proprietà delle resistenze serie e parallelo, l'operazionale funziona come una massa virtuale (pozzo di corrente) per dare una

$$V_{out} = -R_f(I_0[bit0] + I_1[Bit1] + I_2[bit2] + I_3[bit3] + I_4[bit4])$$

dove $I_4 = 2I_3, I_3 = 2I_2, I_2 = 2I_1, I_1 = 2I_0$ e naturalmente $I_4 = V_{ref}/(2R) = 16I_0$.

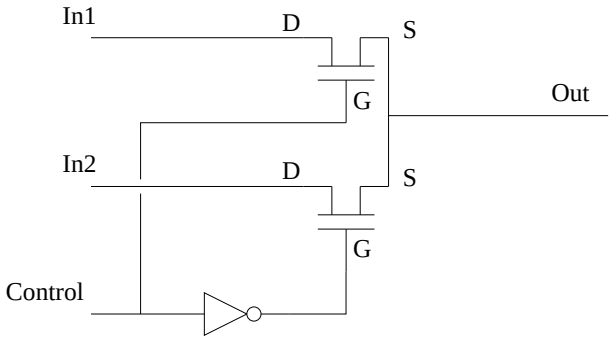


Figura 5.3: Deviatore a Mosfet

I deviatori necessari per la commutazione delle correnti possono essere realizzati in vario modo, ad esempio con dei mosfet (vedi figura 5.3).
A causa dell'invertitore solo uno dei 2 mosfet ha una tensione di gate "alta", perciò solo quello conduce ed è equivalente ad un cortocircuito, l'altro, avendo una tensione di gate prossima a 0 V si comporta come un circuito aperto.

5.3 DAC ottenuti con la tecnica del "pulse width modulation" (PWM).

Ci sono molti casi in cui si vuole controllare una tensione in modo abbastanza rozzo, non si hanno particolari esigenze di velocità e si devono limitare i costi. In questi casi il DAC può essere realizzato usando una tecnologia sostanzialmente tutta digitale. Definiamo prima di tutto il cosiddetto "duty cycle", consideriamo l'onda quadra mostrata in figura 5.4, il segnale oscilla tra i valori V_1 e V_2, sia T il periodo. La tensione ha il valore V_2 per il tempo αT ed il valore V_1 per il tempo $(1-\alpha)T$. α è chiamato "duty cycle" (letteralmente ciclo di lavoro), evidentemente esso è un numero compreso fra 0 e 1, di solito scritto come percentuale, rappresenta la parte del periodo per cui il valore di tensione è "alto".

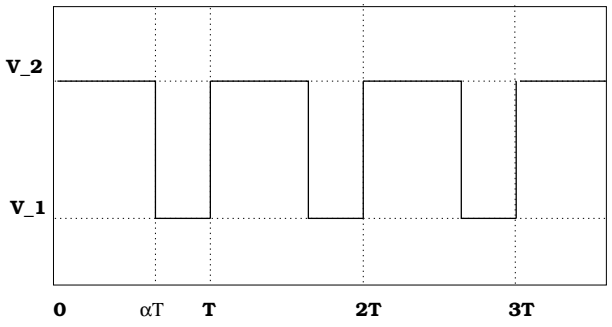


Figura 5.4: Onda quadra

Un duty cycle del 50% corrisponde ad un'onda quadra simmetrica. È facile vedere che il valore medio della tensione dell'onda quadra (su molti periodi) è legato in modo lineare al duty-cycle: $< V >= V_1 + \alpha(V_2 - V_1)$ ed in particolare è proporzionale ad esso se $V_1 = 0$.

Esercizio 27 *Verificare la formula precedente.*

Se ora applichiamo la nostra onda quadra ad un filtro passa basso (ad esempio un semplice "RC") otterremo in uscita la sola componente continua presente nel segnale di ingresso, ossia il valore $< V >$ calcolato precedentemente. La tecnica PWM per ottenere una tensione continua usa proprio questa idea: l'onda quadra a duty cycle variabile viene prodotta con sistemi puramente digitali, sostanzalmente sfruttando un clock piuttosto veloce, un contatore ed un comparatore, l'uscita del comparatore viene successivamente filtrata dal filtro passa basso che può essere realizzato da uno più circuiti "RC" la cui costante di tempo sia stata scelta molto maggiore del periodo T dell'onda quadra:

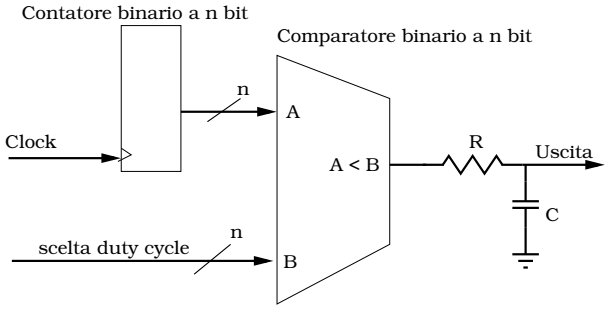


Figura 5.5: Schema di massima per la generazione di un'onda quadra con duty-cycle variabile.

L'ovvio pregio di questo sistema è la semplicità ed il basso costo: può essere costruito usando quasi soltanto componenti digitali, senza alcun bisogno di resistenze di precisione e commutatori elettronici. Se il contatore binario ha molti bit la risoluzione può essere abbastanza elevata, il principale difetto è la presenza nella tensione di uscita di una componente oscillante legata all'attenuazione non infinita del circuito passa basso, questo "rumore" può essere ridotto a livelli trascurabili ma solo aumentando la costante di tempo del filtro e quindi rallentando la risposta del sistema.

Esercizio 28 *Stabilire la relazione matematica tra il duty-cycle desiderato ed il numero da impostare all'ingresso del comparatore.*

Esercizio 29 *Progettare un DAC in tecnologia PWM con una risoluzione di 4 bit usando i circuiti programmabili che avete utilizzato in laboratorio (22V10), il tempo di conversione sia 0.001 s. Calcolare la frequenza del clock.*

Esercizio 30 *Continuazione del precedente. Se invece si volesse costruire un DAC a 10 bit, quale dovrebbe essere la frequenza di clock ? Tenere conto che l'oscillazione della tensione in uscita deve essere minore del quanto di risoluzione del DAC.*

5.4 Gli Analog to digital converter (ADC)

Gli ADC sono dispositivi in grado di fornire in uscita un numero proporzionale al valore della tensione in ingresso. Possono essere costruiti in molti modi a seconda se si vuole privilegiare la velocità di conversione, la risoluzione (ossia il numero di bit con cui viene fornito il risultato), il costo, ecc. Esamineremo nel prosieguo le topologie più comuni:

5.4.1 ADC a rampa semplice

In questo tipo di ADC viene generata una tensione linearmente crescente nel tempo (una rampa appunto) e, simultaneamente, viene fatto partire un contatore, la tensione di rampa viene confrontata con la tensione incognita e quando la tensione di rampa supera la tensione incognita (supposta costante per il momento) il conteggio viene arrestato.

Esercizio 31 *Dimostrare che il conteggio finale è proporzionale alla tensione incognita.*

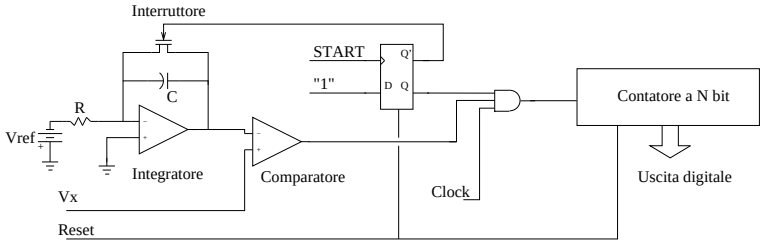


Figura 5.6: Schema di un ADC a rampa semplice

Questo tipo di dispositivi è molto semplice ed economico, tuttavia soffre di alcuni difetti: è tendenzialmente lento, il tempo di conversione è evidentemente proporzionale al massimo conteggio, la precisione è legata a molti parametri: tensione di riferimento, frequenza di clock, resistenza e capacità di integrazione, offset del comparatore, ecc. (che possono anche variare col tempo), perciò il convertitore a rampa semplice può essere usato solo in sistemi a bassa risoluzione, in realtà, allo stato attuale della tecnologia, conviene usare uno dei tipi successivi.

5.4.2 ADC a doppia rampa

L'ADC a doppia rampa sfrutta sostanzialmente lo stesso principio del tipo precedente ma, grazie ad una doppia integrazione è immune a quasi tutti i problemi citati sopra.

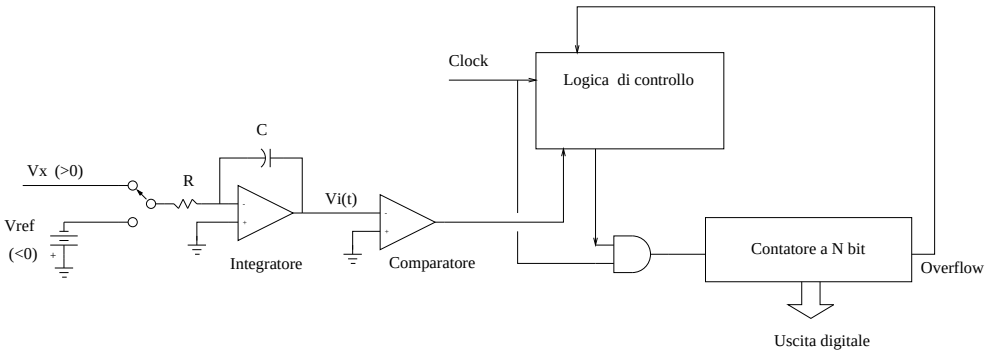


Figura 5.7: Schema a blocchi di un ADC a doppia rampa

Il processo di misura della tensione incognita viene sviluppato in 3 fasi:

- Supponendo che inizialmente il condensatore di integrazione sia completamente scarico, durante la prima fase l'ingresso dell'integratore viene collegato alla tensione incognita (supposta positiva), all'uscita dell'integratore avremo perciò una rampa di tensione positiva $V_i(t) = -\frac{V_x t}{RC}$, perciò il comparatore avrà un valore logico in uscita alto ed il clock farà incrementare il contatore, il sistema resta in questo stato fino a che il contatore non avrà compiuto un ciclo completo ritornando a zero, con N bit questo accadrà dopo 2^N impulsi di clock, ossia per $t_1 = \frac{2^N}{F_c}$. (Fc è la frequenza del clock). La tensione all'uscita dell'integratore a questo punto è evidentemente $V_i(t_1) = -\frac{V_x 2^N}{F_c RC}$.

- La logica di controllo riceve l'overflow del contatore e provvede a collegare la tensione di riferimento V_{ref} (che deve essere negativa) all'integratore, inizia perciò una fase in cui rampa di tensione cambia pendenza e, crescendo linearmente, tende verso 0:

$$V_i(t) = -\frac{V_x 2^N}{F_c RC} - \frac{V_{ref}(t-t_1)}{RC}$$

  Nulla cambia fino a che la rampa non riattraversa il valore 0.

- A questo punto la logica di controllo provvede a bloccare il contatore che mostra il risultato, infatti, imponendo $V_i(t) = 0$ nella precedente si ottiene: $0 = -\frac{V_x 2^N}{F_c} - V_{ref}(t-t_1)$ e notando che $(t-t_1) = \frac{n}{f_c}$ dove n è il conteggio finale del contatore si ottiene: $n = -\frac{V_x 2^N}{V_{ref}}$.

La formula precedente ci dice che il conteggio finale è proporzionale a V_x e dipende solo da V_{ref}, perciò eventuali lente variazioni nel tempo di F_c, R, C sono ininfluenti. Si può dimostrare che anche eventuali offset del comparatore e dell'operazionale sono ininfluenti. Si noti inoltre che se la tensione in ingresso varia il risultato finale è la media del suo valore durante il periodo di integrazione. L'ADC a doppia rampa è quindi un sistema ad elevata risoluzione il cui principale difetto è la lentezza della conversione (dell'ordine di frazioni di secondo).

5.4.3 ADC ad approssimazioni successive

È una architettura di adc molto usata perchè permette di ottenere elevate risoluzioni (anche 16 bit) ed elevate velocità di conversione (dell'ordine del microsecondo).
Il principio di funzionamento è molto semplice ed intuitivo: si confronta la tensione incognita con quella generata da un DAC fino ad ottenere la miglior approssimazione usando il metodo di bisezione.

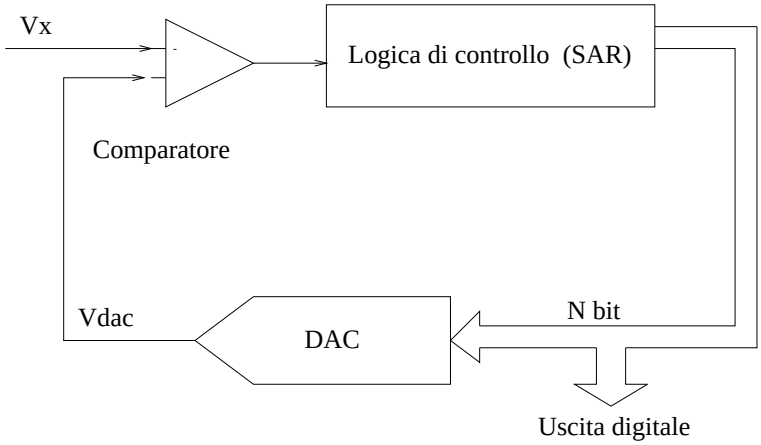


Figura 5.8: Schema a blocchi di un ADC ad approssimazioni successive

Consideriamo un ADC a 8 bit, inizialmente la logica di controllo fornirà al DAC un valore intermedio fra il valore minimo (00000000) ed il valore massimo (11111111), ossia 10000000. Il comparatore confronterà il valore di tensione corrispondente con Vx, la logica di controllo agirà di conseguenza, se Vx risulterà maggiore di Vdac, nella fase successiva, volta a determinare il secondo bit, verrà inviato al DAC il valore 11000000, ossia il primo bit non sarà più toccato e verrà aggiunto un altro bit a 1 immediatamente a destra del precedente (e quindi con un peso relativo 1/2), se invece risulterà Vx minore di Vdac il primo bit sarà azzerato e verrà provato comunque il bit successivo, ossia il secondo tentativo sarà 01000000. Questa procedura di bisezione verrà ripetuta 8 volte fino alla definizione del risultato completo. La logica di controllo, spesso chiamata SAR ossia successive approximation register, è un sistema sequenziale che non è il caso di dettagliare qui, una possibile realizzazione è mostrata in [5].

5.4.4 ADC Flash

Gli ADC Flash sono, come dice il nome, i più veloci, permettono di eseguire conversioni in tempi dell'ordine di 1 ns. Sono però complicati, richiedono, come vedremo, molti componenti attivi e quindi dissipano molta potenza, sono perciò limitati a basse risoluzioni, tipicamente 8 bit.
Uno schema semplificato è mostrato in figura:5.9

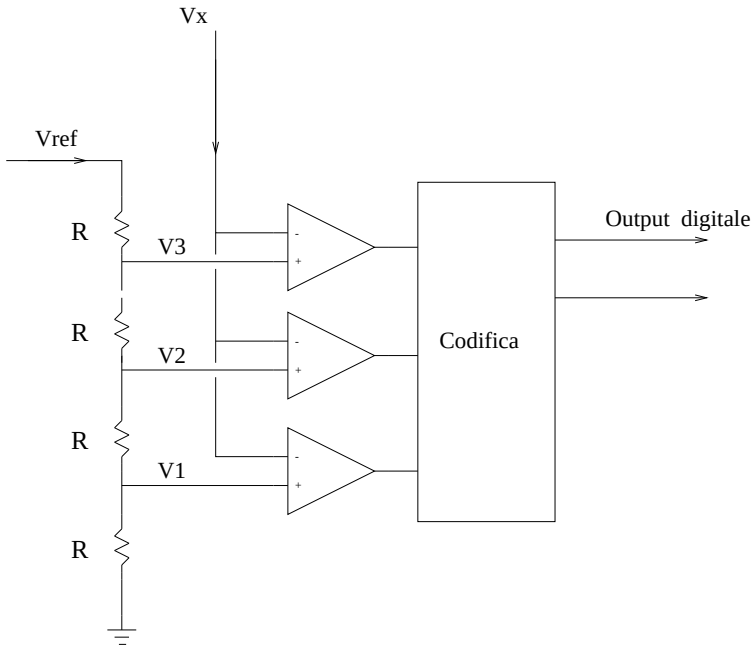


Figura 5.9: Schema a blocchi di un ADC Flash

Il principio di base di questi convertitori è quello di confrontare la tensione incognita con una successione di tensioni uniformemente spaziate tra il valore minimo ed il massimo. Il confronto viene eseguito da opportuni comparatori analogici, ossia da amplificatori differenziali molto veloci la cui uscita è un valore logico (0 – 1). A questo punto un circuito combinatorio opportuno esamina la stringa di zeri e di uni e la codifica nel valore binario corrispondente.
La successione di tensioni uniformemente spaziate viene ottenuta con un partitore costituito di 2^N resistenze uguali. Ciò significa che per un adc da 8 bit occorrono 256 resistenze uguali e 255 comparatori oltre alla logica di codifica che con 256 ingressi provvede a fornire gli 8 bit in uscita.
L'adc mostrato in figura è un dispositivo a 2 bit, infatti la tensione incognita Vx viene confrontata con 4 possibili intervalli di tensione: minore di V1, tra V1 e V2, tra V2 e V3, maggiore di V3.

5.4.5 ADC Sigma Delta

Gli ADC Sigma Delta sono apparsi sul mercato relativamente di recente ma hanno ottenuto una grande popolarità grazie alla loro altissima risoluzione (anche 24 bit, ossia una parte su 16 milioni!!) e basso costo.
Uno schema a blocchi è mostrato in figura:5.10
L'idea di base di questi oggetti è di generare dapprima una stringa di zeri e di uni in cui il numero di uno è proporzionale al valore di Vx e successivamente, attraverso un algoritmo di media (filtro decimatore), ottenere il valore digitale corrispondente a Vx.
Come appare in figura, da Vx, viene dapprima calcolato (Vx - Vdac), questo output viene quindi integrato nel tempo e questo valore viene quantizzato con un ADC ad un bit, ossia un semplice comparatore che confronta

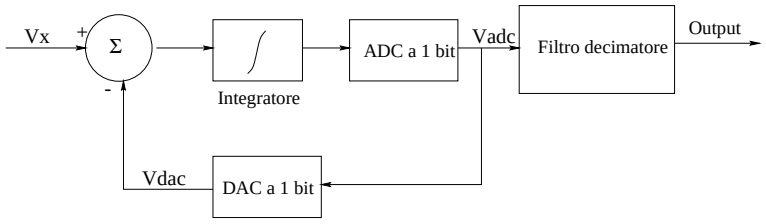


Figura 5.10: Schema a blocchi di un ADC Sigma – Delta

l'ingresso con una tensione intermedia tra il valore minimo e massimo accettato dall'adc e fornisce un output logico 0 – 1, quest risultato viene inviato sia al filtro (di cui parleremo dopo) sia viene inviato al dac ad 1 bit che lo riconverte in una tensione analogica (ossia Vmin oppure Vmax).
Per comprendere meglio il funzionamento di questa prima parte proviamo a fare un esempio, supponiamo che inizialmente l'integratore fornisca un output nullo, per cui l'adc ad 1 bit fornisce 0 logico ed il dac ad 1 bit fornisca Vmin (ad es. 0 V).
A questo punto se Vx è maggiore di 0, il modulo sommatore indicato con la Σ dà in output $V_x - V_{dac} = V_x$, questa viene integrata ed all'adc arriva una tensione linearmente crescente $V_x t$, prima o poi quindi l'adc ad un bit fornirà in output 1 logico e il dac darà Vmax, ora dal sommatore avremo la tensione $(Vx - Vdac) < 0$ per cui l'output dell'integratore sarà una rampa decrescente, l'adc dopo un po' darà zero e similmente il dac.
A questo punto il processo ricomincia con la fase di rampa crescente, all'infinito, è facile vedere che la durata delle 2 fasi crescente e decrescente è legata al valore di Vx (un po' come nell'ADC a doppia rampa).
Ora interviene il filtro decimatore che processa questa stringa in ingresso e ne estrae un codice binario che rappresenta Vx. Questo filtro decimatore è un po' il cuore del sistema e non può essere trattato in modo semplice con gli strumenti a disposizione, a livello intuitivo tuttavia il suo effetto è quello di campionare gli 1 in ingresso, fornendo un risultato tanto più "alto" quanto maggiore è la permanenza temporale dell'adc ad 1 bit nel livello alto.
Gli ADC di questo tipo sono molto comuni ed economici perchè possono essere costruiti usando quasi soltanto tecnologie digitali moto ben sviluppate ed economiche, ma hanno qualche difetto: sono relativamente lenti, spesso solo qualche decina di acquisizioni al secondo, soffrono spesso di problemi di linearità (ossia il legame ingresso–uscita è solo approssimativamente una retta) che riducono anche di 4/5 bit il valore della risoluzione dichiarato dal fabbricante, a causa del loro principio di funzionamento non è facile stabilire il legame temporale tra il momento in cui un valore di tensione viene applicato all'ingresso ed il momento in cui il dato digitale in uscita ne rappresenta una stima ad alta risoluzione.

5.4.6 Caratteristiche degli ADC

Nella scelta di un ADC si devono tenere presenti le seguenti caratteristiche:

- Risoluzione (ossia quanti bit), mai meno di 8, 16–18 in ADC di alta classe ad approssimazioni successive, fino a 24 nei Sigma–Delta.
- Velocità di conversione. Da circa 1 ns fino a frazioni di secondo, di solito gli ADC più veloci hanno risoluzioni più basse di quelli lenti, ADC veloci e ad alta risoluzione (ad esempio conversione in 10 ns e 14 bit di risoluzione) sono piuttosto costosi.
- Linearità integrale. Teoricamente la relazione tra Vx e l'output digitale (chiamiamolo N) dovrebbe essere una relazione lineare: $N = \frac{V_x}{q}$, in realtà non è così , se Vx è zero è possibile avere un piccolo offset in uscita, q avrà un valore leggermente diverso da quello desiderato (errore di scala o di guadagno), inoltre la relazione sarà solo approssimativamente lineare, la relazione ingresso–uscita sarà distorta e la caratteristica reale si allontanerà leggermente dalla retta, la massima distanza tra la retta e la caratteristica reale prende il nome di non–linearità integrale, in un ottimo ADC non è mai superiore a 1–2 quanti (dove qui per quanti si intende la minima differenza di tensione in ingresso che provoca una variazione di 1 in uscita, ossia q.

- Linearità differenziale. In un ADC ideale l'incremento di tensione in ingresso che provoca un incremento di una unità in uscita deve essere costante, in pratica questo non succede, può accadere che per avere un incremento di uno in uscita, l'ingresso possa variare di molto meno di un quanto, oppure di un valore più elevato, in un buon ADC questo errore è sempre un po' inferiore ad un quanto. Si noti che un errore superiore ad un quanto implica che di fronte ad un incremento dell'ingresso l'uscita potrebbe calare! Si veda la figura 5.11 per maggiori chiarimenti.

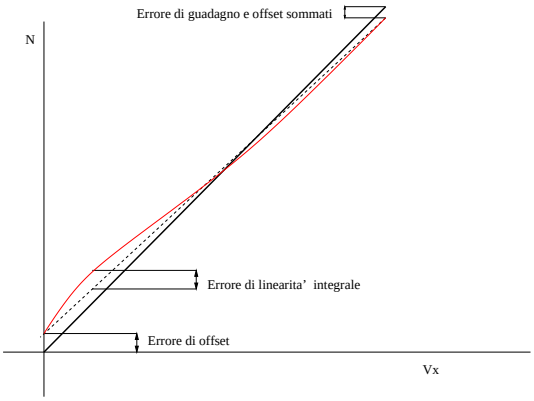


Figura 5.11: Esempio di caratteristica ingresso–uscita di un ADC con l'indicazione degli errori.

5.4.7 Sample and hold

Alcuni ADC, in particolare il tipo ad approssimazioni successive, richiedono che il valore della tensione in ingresso non muti durante la conversione, in effetti nel caso appena citato è facile capire che il calcolo dei singoli bit avviene iterativamente dopo aver preso una decisione per i bit precedenti in base al valore dell'ingresso, se questo muta anche le decisioni precedenti dovrebbero cambiare (ma è troppo tardi), per cui il valore fornito dal dispositivo risulta sbagliato. Per ovviare a questo problema, ed anche per conoscere con precisione a quale istante è riferito il valore fornito dall'ADC, si usa anteporre all'ADC un dispositivo chiamato Sample and Hold (campiona e mantieni) che ha lo scopo di memorizzare il valore analogico di tensione e mantenerlo costante anche se l'ingresso varia. Il principio di funzionamento è mostrato in figura, sostanzialmente ci sono 3 blocchi: un amplificatore operazionale collegato come inseguitore di tensione (guadagno unitario) che isola la memoria analogica dal circuito esterno, il secondo blocco è costituito dal mosfet che viene utilizzato come interruttore e dal condensatore C che svolge il compito di memoria analogica, il terzo blocco è costituito da un scondo operazionale collegato come inseguitore di tensione che isola il condensatore dal successivo ADC, è ovviamente essenziale che questo operazionale abbia altissima impedenza di ingresso, di solito vengono usati dispositivi con stadio di ingresso a FET o MOSFET.

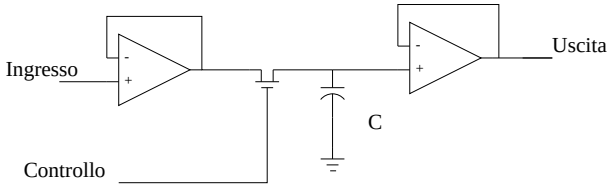


Figura 5.12: Schema di base di un sample and hold

Per una discussione più approfondita sulla progettazione di un sample and hold e sulle precauzioni da osservare

nella scelta del condensatore si veda [4], può essere utile notare che esistono versioni integrate di questi dispositivi, ad esempio LF398 prodotto da National.

Capitolo 6

Microcontrollori

Non faccia l'hardware ciò che può fare il software

Da quanto visto fin'ora risulta evidente che un sistema digitale, opportunamente accompagnato da interfacce e sistemi di conversione analogico–digitali e viceversa è in grado concettualmente di svolgere un qualsiasi compito, tutto dipende in ultima analisi dal sistema sequenziale (automa) che deve prendere le decisioni (cioe' definire le uscite in funzione degli ingressi). Chi ha provato a risolvere qualche esercizio tra quelli proposti durante il corso si è certamente accorto che la progettazione di un automa con una dozzina di stati e pochi ingressi è all'incirca il massimo che si riesca progettare con i metodi sviluppati durante il corso, questo tipo di sistemi può quindi essere impiegato quando le decisioni debbano essere prese in tempi ridottissimi ma non siano troppo complicate, per realizzare sistemi più complessi esistono tecniche opportune quali ad esempio la microprogrammazione (molto ben illustrata in [1]). Con la microprogrammazione è possibile costruire automi complessi a piacere e infatti in questo modo possono essere progettate le unità centrali dei calcolatori (CPU), il prezzo da pagare è comunque sempre quello della difficoltà di progetto per un sistema non banale.
Naturalmente se il progettista di un sistema digitale potesse disporre di una qualche forma di CPU semplificata al massimo, facilmente programmabile, sufficientemente veloce e a basso costo tutti i problemi potrebbero essere affrontati da un altro punto di vista.
Fino agli anni 70 la realizzazione di una CPU era un problema complicato e costoso che richiedeva l'utilizzo di svariati componenti distinti (CPU + memoria RAM + ROM di programma + periferiche + circuiti accessori) che ne limitavano l'uso a sistemi complessi e/o costosi. Nel 1975 fu introdotto sul mercato dalla Intel un microprocessore (siglato 8048) che era costituito da un solo circuito integrato con le seguenti prestazioni:

1. RAM 64 B (Notate 64 **bytes**!)
2. ROM 1 kB (ROM, quindi NON riscrivibile, veniva programmata una tantum dal produttore su specifiche del cliente)
3. Un timer
4. gestione dell'interrupt
5. velocità di calcolo: 0.73 MIPS (milioni di istruzioni al secondo)

Evidentemente era un sistema di calcolo limitato che fu però molto utilizzato: calcolatrici tascabili, console di gioco, gestione di una delle prime auto computerizzate, gestione della tastiera del PC IBM, ecc. Successivamente fu prodotta una versione migliorata con l'EPROM al posto della ROM denominata 8748 che quindi poteva essere programmata dall'utente e riutilizzata molte volte riducendo quindi considerevolmente i costi di sviluppo e di produzione, specialmente in caso di piccole serie. Col passare degli anni ovviamente i progressi della tecnologia permisero di costruire oggetti con caratteristiche migliori a prezzi sempre più bassi. Attualmente in un solo circuito integrato è possibile incorporare non solo **tutti** gli elementi precedenti con magari più memoria RAM (anche molte

decine di kB), ma anche memoria Flash EEPROM che lo stesso programma in esecuzione può modificare, un adc con relativo multiplexer, un dac, spesso realizzato con la tecnica PWM e qualche porta per lo scambio dei dati con altri sistemi (porta seriale, porta USB, porta ethernet, ecc.). Viene quindi spontaneo chiedersi se molti dei problemi affrontati fin'ora non possano essere risolti più semplicemente attraverso una cpu dedicata utilizzata come *automa universale*.

6.1 Struttura di un microcontrollore

Come già detto il microcontrollore è una CPU minimale e completamente contenuta all'interno di un circuito integrato, allo stato attuale della tecnica la maggior parte dei microcontrollori è costituita da:

1. una CPU a 8/16 bit, con architettura più o meno complessa, a seconda del tipo di applicazione per cui è previsto l'oggetto, ad esempio può essere presente un processore hardware per il calcolo del prodotto di due numeri interi, se questo è assente il prodotto dovrà essere calcolato attraverso un semplice algoritmo di somme ripetute (come farebbe un essere umano con carta e penna) che evidentemente richiede più tempo di calcolo. Sempre a proposito dell'architettura della CPU dobbiamo anche distinguere due possibili scelte di struttura ed uso della memoria: i normali PC usano la cosiddetta architettura di Von Neumann, ossia un unico blocco di memoria RAM è utilizzato per contenere sia i dati che i programmi ed esiste un unico "bus" che collega la RAM con la CPU, questa scelta permette di suddividere la memoria fra dati e programmi in modo ottimale riducendo la quantità di memoria totale che deve essere acquistata, svantaggio di questa scelta è l'impossibilità di accedere contemporaneamente alle istruzioni del programa ed ai dati da elaborare, riducendo la velocità del sistema. Nella cosiddetta architettura di Harvard ciò non succede, la memoria dati e quella riservata alle istruzioni sono fisicamente distinte, la cpu vi accede attraverso due bus diversi, diventa pertanto agevole eseguire una istruzione e contemporaneamente leggere e decodificare la successiva. I microcontrollori che useremo noi adottano proprio l'architettura di Harvard.

2. una memoria di programma (solitamente una Flash-EEPROM) da almeno un chilobyte (ma si arriva anche ad alcune centinaia),

3. una RAM (solitamente poche centinaia/migliaia di bytes)

4. una FLASH-EEPROM per conservare dati che non devono essere cancellati quando il sistema viene spento (ma che comunque il programma in esecuzione può modificare),

5. porte digitali di ingresso/uscita per lo scambio dei dati col mondo esterno (il numero di queste porte è di solito molto limitato, si va da pochi bit fino alla decina di bytes, il limite è sostanzialmente legato al numero di "pin" del contenitore), vale la pena notare che abbiamo sia porte "parallele" per lo scambio di dati in modo assolutamente generale ed anche per il controllo di apparati elettrici (rele', motori elettrici, alimentatori), che porte specializzate per utilizzare un certo protocollo (porte USB, seriali, ethernet, ecc.).

6. dispositivi di vario tipo a seconda dell'uso per cui è previsto il microcontrollore, ad esempio temporizzatori, "watchdog"(letteralmente cane da guardia) ossia un dispositivo che "resetta" il sistema se questo non esegue correttamente il programma, generalmente realizzato attraverso un contatore binario che provoca il reset se arriva all'overflow del conteggio e che il programma deve ricordarsi di azzerare periodicamente, comparatori analogici, convertitori analogico-digitali, ecc.

6.2 Sistemi di sviluppo

Parte integrante per la progettazione di un sistema a microcontrollore è il cosiddetto sistema di sviluppo, esso è costituito da diversi componenti, sia hardware che software, necessari per sviluppare il programma che dovrà essere eseguito dalla CPU:

1. il cross-compilatore indispensabile per scrivere il programma che dovrà essere eseguito, solitamente vengono utilizzati sia il linguaggio assembler, sia il linguaggio C o il Basic.
2. simulatore software per il debug del codice (pura simulazione software)
3. il programmatore con l'opportuno software per riversare il codice compilato nel dispositivo.
4. "In circuit emulator": dispositivo hardware che "emula" il microcontrollore, ossia lo sostituisce completamente e permette la verifica in modo facile del funzionamento del sistema completo, cioè anche dell'hardware esterno, permettendo al contempo l'esame dello stato della CPU e della memoria.

6.3 Un esempio: il PIC16F628 della Microchip

Questo processore è venduto in un contenitore da cui fuoriescono solo 18 piedini, 2 sono riservati all'alimentazione (da 3 a 5V.), gli altri 16 possono essere utilizzati a scelta dell'utente, ma rispettando alcune limitazioni imposte dal progettista, ad esempio il piedino 4 può essere usato come reset del processore oppure come uno degli 8 bit della "porta A", mentre i piedini 7 e 8 possono essere usati come secondo e terzo bit della "porta B" oppure come ingressi della porta seriale, perciò il software che viene eseguito dovrà provvedere a "informare" il processore di come verranno utilizzate le varie risorse disponibili, questo lo si ottiene scrivendo nei registri interni del processore delle opportune sequenze di bit.
Il cuore del processore è una CPU a 8 bit di tipo "RISC"[1], ha a disposizione 224 bytes di memoria RAM, ma una parte di questa memoria (circa 30 bytes) è destinata a funzioni particolari sia a livello software che hardware; per distinguerla dalla memoria RAM classica, usata dal programma in esecuzione per allocare le variabili, questi byte di memoria vengono chiamati registri, usando una dizione in auge nei vecchi calcolatori quando la RAM era realizzata usando tecnologie molto differenti (e più lente) da quelle utilizzate per i circuiti della CPU. Tutte le scelte relative alla configurazione dei bit sono effettuate scrivendo l'opportuno valore nei "registri" interni del processore e naturalmente affinchè il sistema funzioni, il progettista deve avere programmato correttamente tutti i registri che modificano la funzionalitá delle risorse da lui utilizzate o che possono interferire con esse, ad esempio per decidere se un bit della "porta A" è un ingresso o un'uscita digitale bisogna agire sul registro "TRISA", porre a zero un bit di TRISA implica che il corrispettivo bit della porta A sarà usato come output, viceversa porre lo stesso bit a uno significa che il bit corrispondente è usato come ingresso. La CPU ha un solo registro di lavoro (chiamato W), le istruzioni elementari operano fra una parola di memoria (o uno dei registri speciali di configurazione o di input/output) e il registro W.

6.3.1 Le Istruzioni

Le istruzioni riconosciute dal processore sono solo 35 e vengono eseguite in un solo ciclo di memoria (ossia 4 periodi di clock) tranne le istruzioni di salto (goto) che richiedono il doppio del tempo a causa della inutilità in questo caso della pipeline interna, infatti il processore, mentre esegue una istruzione, acquisisce sempre la successiva per aumentare la velocità di calcolo, questo trucco permette di raddoppiare la velocità ma non funziona se l'istruzione letta in anticipo è successiva ad un "goto", in questo caso il processore non può utilizzare l'istruzione in memoria, ma deve leggere la nuova istruzione ad un diverso indirizzo.
Possiamo raggruppare le istruzioni in 3 gruppi:

- **istruzioni che operano su byte, ad esempio**
  - ADDWF somma il registro W con un qualsiasi altro registro (o byte della RAM) e scrive il risultato in W o nella RAM iniziale[2].

[1] RISC= Reduced instruction set computer è una architettura del computer in cui il processore è in grado di eseguire un numero relativamente limitato di istruzioni, ma molto efficientemente, si contrappone all'architettura "CISC"= Complex instruction set computer che sceglie una filosofia opposta

[2] il risultato viene scritto in W o nella parola della RAM indirizzata a seconda se l'istruzione in linguaggo assembler termina con ", W" oppure ", F", ad esempio l'istruzione *ADDWF 30, F* somma il contenuto di W al byte di memoria RAM il cui indirizzo esadecimale è 30 dove scrive pure il risultato, questa convenzione è usata da tutte le istruzioni (per cui è significativa).

  - MOVF che copia il contenuto di un registro o byte di memoria in W.
  - RLF e RRF per ruotare circolarmente a sinistra o a destra un byte concatenandolo con il bit di "carry" del registro di stato.
  - SUBWF che sottrae W da F.
  - IORWF calcola l' OR inclusivo di W ed un byte di memoria.

- **istruzioni che operano su bit, ad esempio**
  - BSF pone a 1 un bit a piacere dei registri o della RAM.
  - BTFSC salta l'istruzione successiva se un certo bit dei registri o della ram e zero, questo tipo di istruzione serve per realizzare istruzioni del tipo if (condizione) goto ...

- **istruzioni che operano con costanti e di controllo , ad esempio**
  - ADDLW somma una costante a W.
  - CALL esegue una routine.
  - GOTO ovvio...
  - RETURN per uscire da una routine, ecc.

L'elenco completo delle istruzioni è contenuto nei manuali liberamente scaricabili dal sito Microchip la cui lettura (170 pagine a cui si aggiungono note tecniche, manuali di uso generali, manuali per l'uso del software, ecc.) dà un'idea di come la complessità dell'hardware si sia adesso trasferita sul software! Fortunatamente occorre leggere solo le parti utili per il lavoro che si intende svolgere, raramente più di poche decine, almeno per iniziare.

6.3.2 Lo Stack

Un'ulteriore zona di memoria a disposizione della CPU e continuamente utilizzata è chiamata **"stack"**, ossia mucchio, pila. Dal punto di vista software è una struttura per memorizzare dati in modo sequenziale, è del tipo "last in, first out", ossia l'ultimo dato scritto è il primo che viene estratto in lettura. Lo stack è utilizzato dalla CPU per gestire le chiamate a funzioni attraverso l'istruzione *call*: quando viene chiamata una funzione il processore copia automaticamente nello stack l'indirizzo della istruzione successiva (l'indirizzo di ritorno a cui dovrà continuare il programma al termine dell'esecuzione della funzione) e modifica un puntatore, lo "stack pointer" per tenere conto di quanti indirizzi vi sono stati scritti, inoltre pone in un altro registro interno, il "program counter", l'indirizzo della prossima istruzione da eseguire, cioè la prima istruzione della funzione.
Al termine della funzione l'istruzione return provvede a ripristinare il valore del program counter estraendolo dallo stack e ad aggiustare lo stack pointer. Poichè lo stack di questo processore può contenere al massimo 8 indirizzi bisogna stare attenti a non "annidare" (cioè inserire una dentro l'altra) troppe chiamate a funzioni.

6.3.3 L'Input–output

Un'attività fondamentale per tutti i processori è lo scambio di informazioni col mondo esterno, ossia la gestione dell'input/output, spesso abbreviato in "I/O". Nel nostro PIC 16F628 le operazioni di input output possono riguardare sia le due "porte" A e B, sia la maggior parte degli altri periferici (porta seriale, timer, generatore di tensione programmabile, ecc.). L'input/output può assumere aspetti più o meno complessi, nel nostro processore tutti i periferici vengono visti come byte di memoria e quindi sono controllati scrivendo e leggendo per mezzo della istruzione **"movf"**, non esistono cioè istruzioni particolari per scrivere sui o leggere dai "periferici", essi sono visti dal software come byte di memoria, l'hardware provvede a differenziare in modo assolutamente trasparente per il programmatore tra la vera memoria RAM e i periferici. Questo metodo di gestione dell'input output viene chiamato *"memory mapped I/O"*.
La caratteristica principale delle operazioni input–output (che ne complica la gestione) è quella di essere asincrone (ad esempio è impossibile prevedere quando l'utente batterà un tasto della tastiera o premerà un bottone della

tastiera del telecomando del televisore), il modo più semplice per tener conto di questo fatto è quello di far controllare continuamente dalla CPU il periferico da cui ci aspettiamo dei dati in modo da essere pronti a gestirli appena arrivano. Questa tecnica è chiamata "polling" ed ha evidentemente un difetto: è molto inefficiente, illustriamo il punto con un esempio: supponiamo di utilizzare il microcontrollore per svolgere due compiti: misurare una volta al secondo una tensione ed in parallelo controllare un altro dispositivo. Se vogliamo misurare periodicamente, in istanti di tempo ben definiti, una tensione per mezzo di un ADC, è necessario utilizzare un timer, potremmo leggere ripetutamente il timer per decidere quando è il momento di acquisire il risultato dell'ADC, ma se facessimo questo rischieremmo di non avere più risorse sufficienti per controllare gli altri dispositivi perchè la nostra CPU sarebbe totalmente assorbita dal compito di leggere il timer per capire se è passato un secondo o no, quindi non riuscirebbe a fare nient'altro anche se in fondo non starebbe facendo quasi nulla!
La soluzione di questo dilemma si chiama **"interrupt"**, come dice il nome l'uso dell'interrupt implica che la CPU svolge normalmente i suoi compiti, quando un periferico deve ricevere o fornire dati (ad esempio il timer segnala che è giunto il momento di leggere l'ADC oppure il mouse è stato mosso) la CPU interrompe l'esecuzione di ciò che sta facendo, esegue un programma per gestire i nuovi dati, dopo di che ritorna al suo programma iniziale. Evidentemente questo è un metodo molto più efficiente, ma anche più complesso, è infatti necessario interrompere il programma in esecuzione e poi riprenderlo dal punto in cui era stato interrotto senza che esso ne venga disturbato (a parte l'ovvio rallentamento), cioè non deve essere alterato in alcun modo lo stato della CPU (registri interni e registro di stato[3]) e la memoria utilizzata dal programma principale. Il nostro microcontrollore si comporta in questo modo (peraltro utilizzato con minime modifiche da quasi tutti i processori): all'arrivo dell'interrupt la CPU completa l'istruzione in corso di esecuzione, inibisce ulteriori interrupt da altri periferici azzerando un bit di abilitazione nel registro di configurazione degli interrupt (INTCON), salva sullo stack il "program counter" ossia il registro che contiene l'indirizzo in memoria della prossima istruzione ed il controllo della CPU viene passato ad una routine che nel nosto caso **deve** iniziare ad un indirizzo prefissato (4 nel nostro caso). La routine esegue il compito assegnatole e quindi ritorna al programma precedente recuperando l'indirizzo dallo stack attraverso l'istruzione speciale **retfie**= "return from interrupt" che provvede a riabilitare automaticamente gli interrupt.
Si noti che è compito della routine di interrupt non alterare lo stato interno della macchina durante le sue operazioni affinchè al suo ritorno il programma che veniva eseguito precedentemente "non si accorga" di essere stato interrotto! Questo implica copiare il registro W ed il registro di stato in una zona riservata della RAM.

6.4 Un esempio: riprogettiamo l'esperienza per la misura della velocità del suono

La misura della velocità del suono richiede un cronometro in grado di misurare tempi anche molto inferiori al millesimo di secondo con buona risoluzione (qualche microsecondo), il sistema deve iniziare il conteggio all'arrivo di un segnale di *start* ed arrestarsi all'arrivo del segnale di *stop*, la misura dell'intervallo di tempo deve essere visualizzata in modo conveniente su un opportuno display. Abbiamo risolto il problema durante il corso sfruttando un circuito integrato 22V10, in esso abbiamo inserito due automi: un contatore a 8 bit ed un automa di controllo che abilita il contatore al momento dello start e lo blocca al momento dello stop. Adesso proveremo a risolvere il problema usando un microcontrollore tipo PIC 16F628 prodotto da Microchip. La scelta di questo produttore è legata al fatto che questi componenti sono molto diffusi, il software da utilizzare è gratuito e l'hardware per la programmazione del chip è molto economico.
L'idea di base e' di usare un timer contenuto nel microcontrollore per misurare l'intervallo temporale con buona risoluzione. Il minimo intervallo temporale misurabile è pari al tempo impiegato dal sistema per eseguire una istruzione elementare, ossia quattro periodi del clock del sistema, il nostro microcontrollore potrebbe lavorare con un clock a 20 MHz, fornendo quindi una risoluzione temporale pari a 200 ns (cioè il tempo necessario per eseguire una istruzione), ma noi abbiamo deciso di limitare la frequenza di clock a 4 MHz, la risoluzione temporale a questo punto è pari a $1\mu s$, ampiamente adeguata alle nostre esigenze.

[3] Il registro di stato contiene informazioni sul risultato dell'ultima istruzione eseguita, ad esempio se il risultato è zero, se c'è stato un overflow, ecc., queste informazioni vengono usate di solito per modificare il flusso del programma, cioè eseguire salti condizionati (come la struttura "if–then–else" dei linguaggi di alto livello).

Il sistema farà partire il timer all'arrivo del segnale di start e lo arresterà all'arrivo dello stop, il risultato fornito dal timer è un numero binario che deve essere convertito in numero decimale e mostrato su un qualche tipo di display. Il microcontrollore scelto contiene in effetti 3 timer al suo interno: TIMER0,1,2. Ogni timer è realizzato attraverso un contatore binario completamente controllato via software. Il TIMER1 in particolare è un contatore a 16 bit, gli altri due sono invece contatori a 8 bit, quindi più scomodi da usare per avere un fondo scala sufficiente (il software potrebbe comunque tenere conto dei numerosi overflow dei contatori a 8 bit ed estendere quindi il fondo scala ma sarebbe evidentemente una complicazione). Decidiamo quindi che l'arrivo del segnale di start genererà un interrupt che farà immediatamente eseguire una procedura il cui unico compito sarà quello di far partire il timer previamente azzerato. Ovviamente tra il momento dell'arrivo del segnale di start e l'effettivo inizio del conteggio del timer passeranno pochi microsecondi. Questo tempo morto è evidentemente una limitazione del nostro sistema, d'altra parte lo spazio percorso dal suono in $\approx 10\mu s$ è chiaramente trascurabile, è tuttavia importante che questo ritardo sia costante, se variasse introdurremmo un errore sistematico. Proprio il fatto che la partenza del timer sia legata ad un interrupt ci garantisce che l'errore sia minimo (al massimo l'incertezza sul ritardo introdotto dall'esecuzione di una istruzione elementare, ossia $1\mu s$. Per il segnale di stop possiamo sfruttare una caratteristica del timer in questione che può essere utilizzato in "capture mode", ossia all'arrivo del segnale di stop la logica hardware del microcontrollore provvede automaticamente a copiare il conteggio del timer in una opportuna memoria interna senza introdurre ritardi od errori sistematici.
Il risultato binario deve poi essere convertito attraverso un opportuno algoritmo in 4 cifre decimali che possono essere mostrate[4] usando 4 display a led del tipo "seven segment" in cui le cifre decimali sono rappresentate attraverso l'accensione di un numero variabile di led opportunamente disposti. Lo schema elettrico del cronometro è mostrato in fig:6.1
Si noti il circuito stabilizzatore di tensione costruito intorno all'integrato 7805; il diodo posto in serie all'alimentazione è solo una protezione contro le inversioni di polarità. Protezioni analoghe sono state poste sui due ingressi start e stop, qualunque tensione negativa o maggiore di 5v viene limitata dai 2 diodi. Per motivi "storici" l'ingresso di start è "attivo alto", quello di stop è "attivo basso". Il pulsante del reset fa ripartire il programma del microcontrollore dall'inizio dopo avere effettuato una misura. La frequenza di clock del sistema viene generata da un oscillatore che utilizza un risuonatore a quarzo di elevata precisione (migliore dello 0.1%).
Una nota particolare meritano i 4 display, il microcontrollore non ha sufficienti terminali per alimentarli tutti e 4 contemporaneamente, occorrerebbero infatti 7 * 4 = 28 segnali in uscita, perciò i 4 display devono essere multiplexati, ossia accesi in rapida successione, in questo modo sarà possibile collegare in parallelo i 7 segmenti anodici dei 4 display e collegare i 4 catodi individualmente al processore. Poichè la corrente in un display può facilmente superare il limite massimo erogabile dal microcontrollore (25 mA), il pilotaggio del display dal lato catodico è affidato ad un transitor controllato a sua volta dal processore. Si noti l'uscita del bit A4, poichè questo bit della porta è collegato ad un circuito del tipo "open drain" si è dovuta aggiungere una resistenza di carico (invece di quella di limitazione di corrente.
Naturalmente quello indicato non è l'unico modo per realizzare un cronometro con questo processore, si sarebbe potuto, ad esempio, usare il "capture mode" del timer sia per catturare lo start che lo stop, tuttavia questo avrebbe richiesto degli ulteriori circuiti esterni per convogliare sia lo start che lo stop sullo stesso ingresso del processore, inoltre sarebbe stato possibile aumentare il fondo scala aggiungendo un contatore software incrementato ad ogni overflow del contatore hardware, sarebbe stato utile a questo punto mostrare più cifre inventandosi qualche trucco o modificando l'hardware (ad esempio aggiungendo un display intelligente esterno). Tutto questo viene lasciato come esercizio.

Esercizio 32 *Modificare il circuito di start e stop in modo da usare il solo bit B3 (piedino 9) del processore per controllare il cronometro, i due segnali di start e stop provenienti dal trasmettittore e dal ricevitore saranno fusi in un solo segnale che il software dovrà poi utilizzare intelligentemente.*

Esercizio 33 *Dopo avere risolto l'esercizio precedente modificare il software in modo opportuno in modo che la misura dell'intervallo temporale sia fatta sfruttando integralmente il "capture mode" del timer 1.*

[4] Con 16 bit in realtà occorrerebbero 5 cifre decimali perchè il massimo numero rappresentabile con 16 bit è $2^{16} - 1 = 65535$, tuttavia con 4 cifre possiamo contare fino a $9999\mu s$ ossia $0.01s$, ampiamente sufficienti.

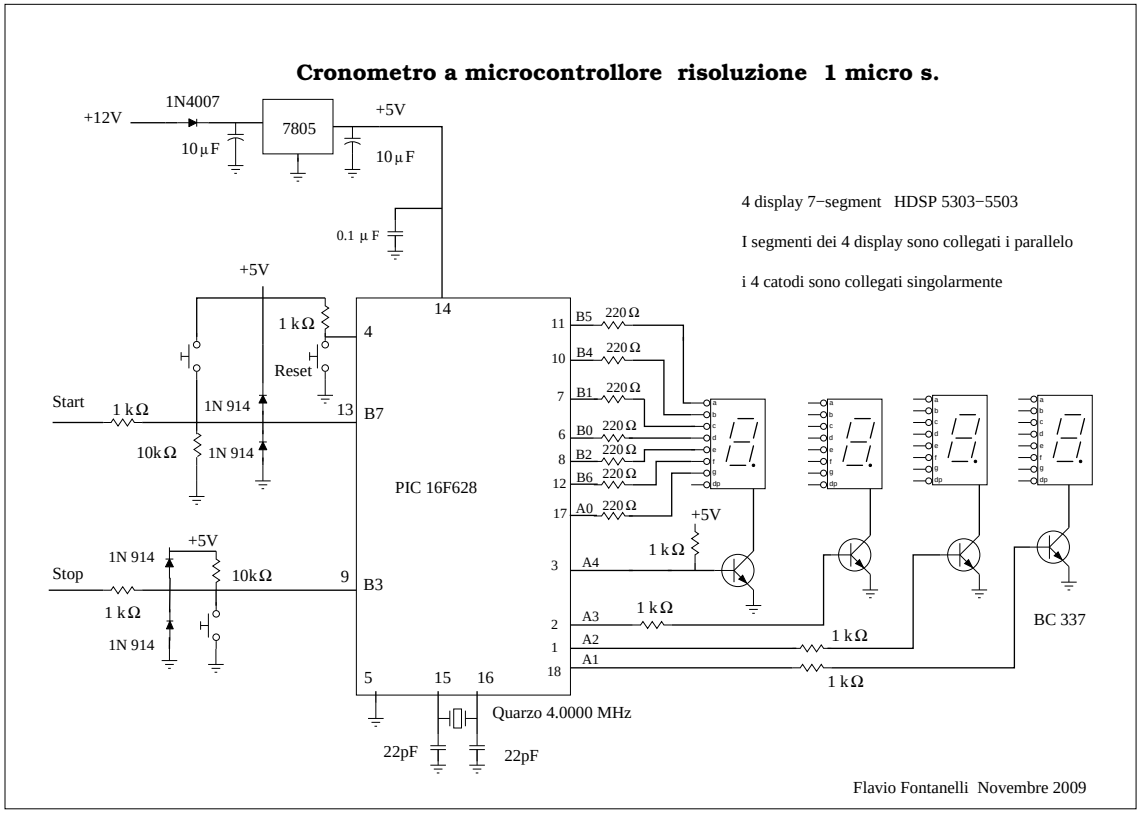


Figura 6.1: Schema del cronometro a microcontrollore

Esercizio 34 *Sfruttando il compilatore C fornito con il sistema di sviluppo riscrivere il programma di controllo del cronometro. Confrontare i 2 codici e misurarne le differenze (lunghezza del codice, occupazione di memoria, funzionalità, ecc)*

La versione assembler del programma è presente sul sito da cui avete scaricato queste dispense (file timersuono.asm)
Bisogna notare che non è sempre necessario scrivere programmi in assembler, lo stesso programma per la gestione del cronometro poteva essere scritto piu' semplicemente in un linguaggio di alto livello come il C, una possibile versione e' mostrata di seguito:

```
//timer per esperienza misura della velocita' del suono
// costruita intorno ad un pic16f628   Versione 1.1  gennaio 2010
#include <htc.h>
#include <pic.h>
#define bitset(var, bitno) ((var) |= 1UL << (bitno))
#define bitclr(var, bitno) ((var) &= ~(1UL << (bitno)))
#define testbit(data,bitno) ((data>>bitno)&0x01)

__CONFIG(PWRTEN & LVPDIS & WDTDIS & UNPROTECT & MCLREN & LVPDIS & XT & BOREN);
//   equivalentemente  __CONFIG(0x3D61);

#define _XTAL_FREQ      4000000 // il clock e' a 4 MHz
#define  start          7
```

```
#define  stop           3
// ricordati PORTA, 5 e' clear
#define  migliaia       4
#define  centinaia      3
#define  decine         2
#define  unita          1
void converti();
void manda();
void B2_BCD();

char tmp, tmpout;    // variabile di lavoro, chiunque la puo' utilizzare
char temp, H_byte, L_byte;
char mig, cent, dec, unit;
char start_ok;      // quando arriva start -> 1
int contatore;
main(){
// qui inizia il programma
inizia:
  OPTION =0;
  CMCON = 7;     // disabilitiamo i comparatori
  TRISA = 0x20;  // porta A pilota il catodo dei display e segmento A
  TRISB = 0x88;  // porta B usata per start, stop e per  bit display (out)

// registro OPTION: resistori di  pull-up individuali,
// interrupt on falling edge di rb0,
// tmr0 incrementato da clock interno, incremento sul fronte di salita
// prescaler assegnato a tmr0, rapporto 1/2.
// queste opzioni sono quasi tutte inutili per noi perche' non usiamo timer0

  PR2 =  0XFF;          //timer2 period register
  PIE1 = 0;             // disabilitiamo tutti gli interrupt
  PCON = 8;       // 4MHz, brown out detect
  VRCON= 0;     // non uso tensione di riferimento
  T1CON =0;             // timer1 per ora bloccato
  T2CON =0;   // timer 2 non usato
  PIR1  =0;   // non serve, azzerato per sicurezza
  CCP1CON =4;     // usero' il "capture mode"
  TMR1L =6; // tengo conto del ritardo alla partenza
  TMR1H =0;
  CCPR1L =  CCPR1H=0;   // registro del capture mode =0 per sicurezza
  INTCON =8;      // disabilito tutti gli interrupt tranne port change

  PORTA =0;       // azzero porta A e B
  PORTB =0;

  start_ok = 0;
  converti();
  bitset (PORTA, unita);    // accendi solo il display delle unita'
  manda();

  while (testbit(PORTB, start))  // aspetto che sia start =0
```

```
  {
    __delay_ms(1);
  }
  __delay_ms(1);  // debouncing

  while (!testbit(PORTB, stop))  // aspetto che sia stop = 1
  {
    __delay_ms(1);
  }
  __delay_ms(1); // debouncing

// adesso siamo pronti a lavorare: abilito interrupt e annullo i flag
  INTCON = 0x88;    // disabilito tutti gli interrupt tranne port change
  // aspetto che la routine di interrupt metta start_ok=1
  while (!start_ok)
  {
    asm ("nop"); // non faccio nulla
  }
  di(); // impedisco altri interrupt
//    verifichiamo che lo stop sia ancora alto
  if (!testbit(PORTB, stop)) goto inizia;       // errore,  reset

  while (testbit(PORTB, stop))  // stop a 0 ?
  {
    asm ("nop");                // no
  }
  // si, stop =0
  H_byte = CCPR1H;  // salviamo il risultato
  L_byte = CCPR1L;
  B2_BCD(); // convertiamo in decimale (BCD packed)
  // resettiamo il timer1
  bitclr (T1CON, TMR1ON);    // fermo   timer1
  CCPR1H = CCPR1L =0; // azzero timer1
  bitclr (PORTA, unita); // spengo tutto

  while (1)   // loop senza fine per mostrare il risultato sul display
  {
    tmp = unit;
    converti();
    manda();
    bitset(PORTA, unita);
    bitclr(PORTA, migliaia);
    __delay_ms(1);

    tmp = dec;
    converti();
    manda();
    bitclr (PORTA, unita);
    bitset(PORTA, decine);
    __delay_ms(1);
```

```
    tmp = cent;
    converti();
    manda();
    bitset(PORTA, centinaia);
    bitclr(PORTA, decine);
    __delay_ms(1);

    tmp = mig;
    converti();
    manda();
    bitclr(PORTA, centinaia);
    bitset (PORTA, migliaia);
    __delay_ms(1);
  }
}

void converti()
 {
// converte una cifra decimale 0-9 nella corrispondente
// configurazione del display 7 segment
// segmento a su RB7, b su RB6, RB3 non usato, g su RB0.
  switch (tmp)
  {
   case 0:
     tmpout = 0x77;  // 0
     return;
   case 1:
     tmpout = 0x12;  // 1
     return;
   case 2:
     tmpout = 0xB5;  // 2
     return;
   case 3:
     tmpout = 0xB3;  // 3
     return;
   case 4:
     tmpout = 0xD2;  //4
     return;
   case 5:
     tmpout = 0xE3;  // 5
     return;
   case 6:
     tmpout = 0xE7;  // 6
     return;
   case 7:
     tmpout = 0x32;  // 7
     return;
   case 8:
     tmpout = 0xF7;  // 8
     return;
   case 9:
```

```
     tmpout = 0xF2;  // 9
     return;
  }
}

void manda()
{
// accendo i 7 segmenti, 6 segmenti sono sulla porta B
//  il segmento A e' su PORTA, 0
  PORTB = tmpout;
  if (testbit(tmpout, 7)) PORTA |= 0x80;   // lo metto a 1
  else  PORTA &= 0x7F; // lo azzero
  return;
}

void interrupt isr()
{
  int dummy;
// routine di interrupt, viene chiamata direttamente dalla cpu quando start=1
  PORTB   = 0;   // spengo il display (per debug)
  if (INTCON & 0x01)      // se bit RBIF ==1 (cioe' e' cambiato start)
  {
   bitset (T1CON,0);    //faccio partire il timer
   dummy = PORTB;    //leggo la porta per annullare il cambiamento del bit
   bitclr (INTCON, 0);         // azzero il flag di interrupt
   start_ok =1;
   return;
  }

//   interrupt spurio
  INTCON = 8;     // non riabilito interrupt, dovro' dare reset
  return;
}

void B2_BCD()
{ // estrae le cifre decimali dal risultato del timer1
  contatore = (int)H_byte<<8 | (int)L_byte;
  mig = contatore / 1000;
  cent = (contatore/100) % 10;
  dec  = (contatore/10) % 10;
  unit =  contatore % 10;
  return;
}

//   Inizializziamo la EEPROM con alcuni dati legati alla versione
//  "V1.1  F.F. - Gennaio 2010 ",0
__EEPROM_DATA('V', '1', '.', '1', ' ', 'F', '.', 'F');
__EEPROM_DATA('5', '/', '1', ' ', '2', '0', '1', '0');
```

Esercizio 35 *Modificare il programma precedente per fargli eseguire automaticamente 100 misure, mostrando ogni risultato per alcuni secondi.*

Esercizio 36 *(solo dopo avere risolto l'esercizio precedente, usare il C) Modificare il programma precedente per fargli mostrare anche la media delle 100 misure, prima assicuratevi che il sistema funzioni in modo affidabile, potrebbe essere necessario rigettare automaticamente misure fuori da un intervallo ragionevole.*

Bibliografia

[1] F.Luccio, L.Pagli. Reti logiche e calcolatore. Boringhieri

Buona trattazione dei circuiti combinatori e dei principi dei sistemi sequenziali, l'analisi di alcuni problemi formali a volte ne appesantisce la lettura.

[2] A. De Gloria. Fondamenti di progettazione elettronica analogica e digitale. Franco Angeli

[3] F. J. Hill, G. R. Peterson. Introduction to Switching Theory and Logical Design. John Wiley and sons.

Trattazione abbastanza completa e precisa dell'algebra di Boole.

[4] P.Horowitz, W.Hill. The art of electronics. Libro enciclopedico, molto chiaro, ben fatto e abbastanza aggiornato.

[5] Cantarano, Pallottino. Elettronica Integrata Vol.2, Etas Kompass.

Ottimo libro per gli aspetti piu' formali, e' diventato piuttosto obsoleto.

[6] Millmann ha scritto molti libri di elettronica analogica e digitale, sempre molto chiari e didattici su cui si sono formate generazioni di progettisti, cercate in biblioteca quello che più vi piace, l'ultima fatica (credo) sia quella scritta insieme a Grabel, tratta tutto tranne i microprocessori. Esistono anche edizioni italiane, ad esempio: Microelettronica.

[7] V.Alessandroni Elettronica digitale e microprocessori. Boringhieri

Libro piuttosto vecchio (il microprocessore in questione è l'8080) ma molto pratico, utile per conoscere la tecnologia TTL e le tecniche di progetto dei sistemi sequenziali.

[8] Jan M. Rabaey, Digital Integrated Circuits A design perspective, Prentice Hall

Libro specialistico sulla progettazione dei circuiti digitali, trattazione molto chiara e di ottimo livello.