

Corso di Reti logiche e sistemi di interfacciamento a microprocessori

LEZIONE N°1: "L'algebra Booleana"

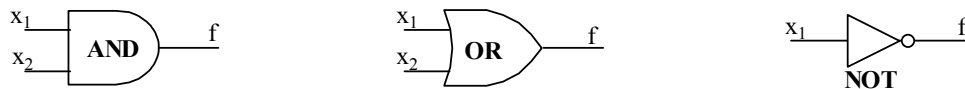
L'algebra che useremo nel nostro corso è dovuta a Boole ed è detta algebra booleana. Essa prende forma già nel XVII secolo da alcuni studi iniziati da Leibnitz e poi conclusi e raccolti da George Boole in un suo libro. Come vedremo tra breve è lecito definire la teoria di Boole come un'algebra poiché essa assegna all'insieme $\{1,0\}$ due leggi binarie di corrispondenza interna.

Le variabili di questa algebra sono binarie e possono assumere solo due valori, l'assegnazione più comune è quella dell'insieme logico $K=\{1,0\}$, tuttavia sono anche usate diverse convenzioni che possono in ogni caso essere riportate o riferite all'insieme K .

Le operazioni più comuni che si possono stabilire tra le variabili booleane sono: AND(\cdot), OR($+$) e NOT($\bar{}$), le operazioni di AND e OR costituiscono le due leggi interne che con l'insieme $K=\{1,0\}$ formano la struttura di algebra, $\langle K, \text{AND}(\cdot), \text{OR}(+) \rangle$.

All'operatore AND si associa il simbolo e l'operazione di moltiplicazione, all'operatore OR è associato il simbolo e l'operazione di addizione, infine, all'operatore NOT è associato il simbolo e l'operazione di negazione.

Ciascuna delle funzioni base AND, OR e NOT si accompagna ad un simbolo circuitale e ad una tabella della verità. La tabella della verità indica, per tutte le porte logiche, le possibili combinazioni di ingressi e il risultato dell'operatore logico.



L'operatore AND corrisponde, nella sintassi della letteratura, alla congiunzione "e", questa porta logica assume infatti valore logico 1 solo quando x_1 e x_2 sono alti. Con il termine valore alto indichiamo il valore logico pari ad 1. ($\{1=\text{High}=\text{Alto}; 0=\text{Low}=\text{Basso}\}$)

Tabella di verità per la porta logica AND:

(True tables for logical Operations AND)

AND		
x_1	x_2	$f = x_1 \text{ AND } x_2$
0	0	0
0	1	0
1	0	0
1	1	1

L'operatore OR corrisponde, nella sintassi della letteratura, alla congiunzione "oppure", questa porta logica assume infatti valore logico 1 quando x_1 o x_2 è alto. Questa operazione, come vedremo più avanti, è duale all'operazione logica di AND.

Tabella di verità per la porta logica OR:
(True tables for logical Operations OR)

OR		
x_1	x_2	$f = x_1 \text{ OR } x_2$
0	0	0
0	1	1
1	0	1
1	1	1

L'operatore logico di NOT corrisponde, nella sintassi della letteratura, alla negazione. L'operatore NOT a differenza degli operatori AND e OR, che funzionano con almeno due variabili fino ad un massimo di n variabili, necessita di una sola variabile.

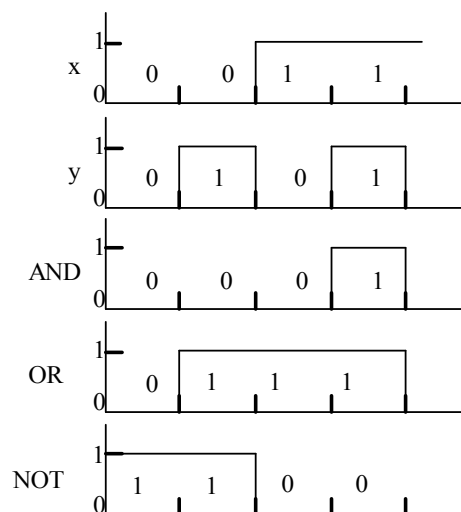
Se in ingresso è posto il valore logico 1 in uscita troveremo la sua negazione logica, cioè zero.

Tabella di verità per la porta logica NOT:
(True tables for logical Operations NOT)

NOT	
x_1	$f = \text{NOT } x_1$
0	1
1	0

Per gli operatori logici di AND e di OR, all'aumentare delle variabili di ingresso corrisponde un aumento, in termini di dimensioni, delle rispettive tabelle di verità che saranno formate sempre da 2^n righe, con n numero di variabili in ingresso. Si dimostra (non sarà oggetto di questo corso) che con le funzioni base dell'algebra booleana è possibile ricostruire qualunque funzione logica.

L'algebra di Boole consente di rappresentare e semplificare relazioni e condizioni nell'interno di un programma. Un'altra interpretazione che possiamo dare all'algebra di Boole ci viene suggerita dai circuiti elettronici. Questi sistemi sono caratterizzati da grandezze fisiche (segnali di tensione o di corrente etc...) che assumono due gamme di livelli logici, uno H (livello alto = High) ed uno L (livello basso = Low) ai quali è spontaneo far corrispondere i valori 0 ed 1 detti stati logici. La corrispondenza può essere fatta secondo la logica positiva (che prevede ad 1 il passaggio di corrente e a 0 l'assenza di corrente, ciò vale anche per la tensione) e secondo la logica negativa (che prevede ad 1 l'assenza di corrente e 0 il passaggio di corrente, ciò vale anche per la tensione). I modi in cui gli ingressi ad una porta logica cambiano possono talvolta essere descritti da alcuni diagrammi detti di transizione, essi forniscono informazioni sulla tempificazione che i diversi segnali logici assumono in ingresso alle porte (Gates) e vengono quindi riportati sulle ordinate di un piano cartesiano in corrispondenza dell'ascissa temporale, ad esempio:



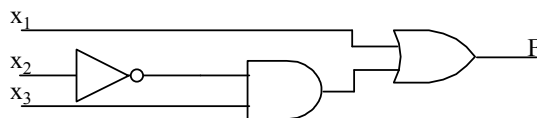
Le funzioni booleane sono quelle funzioni di variabile booleana ottenute componendo le funzioni base (AND, OR e NOT) e che possono assumere soltanto i valori "vero" e "falso", associati ovviamente ai valori logici 1 e 0. Ad esempio:

$$F = X + \bar{Y}Z$$

Ogni funzione logica può essere rappresentata da una tabella di verità, ad ogni possibile valore di ingresso si fa corrispondere in uscita uno dei valori logici. Per la funzione considerata si ha questa tabella di verità:

Tab. di verità della funzione F			
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

L'algebra di Boole favorisce inoltre il progetto di reti combinatorie, a partire da una funzione logica è possibile far corrispondere ad essa un circuito logico. Per la funzione considerata si ha questo circuito:



Come ogni algebra fin qui studiata, anche l'algebra Booleana ha i suoi teoremi. Il metodo più semplice per dimostrare ciascuno di

questi teoremi è quello di segnarne la tabella di verità e di verificarne l'esattezza:

(Basic Identities of Boolean Algebra)

- | | |
|---------------------------|----------------------|
| 1. $X+0=X$ | 2. $X\cdot 1=X$ |
| 3. $X+1=1$ | 4. $X\cdot 0=0$ |
| 5. $X+X=X$ | 6. $X\cdot X=X$ |
| 7. $X+\bar{X}=1$ | 8. $X\cdot\bar{X}=0$ |
| 9. $\overline{\bar{X}}=X$ | |

Altre proprietà derivano invece dalle comuni proprietà dell'aritmetica come quella commutativa, associativa e distributiva, si hanno infatti:

- | | |
|-----------------------|-----------------------|
| 10. $X+Y=Y+X$ | 11. $XY=YX$ |
| 12. $X+(Y+Z)=(X+Y)+Z$ | 13. $X(YZ)=(XY)Z$ |
| 14. $X(Y+Z)=XY+XZ$ | 15. $X+YZ=(X+Y)(X+Z)$ |

Infine, le ultime due proprietà derivano dai teoremi di De Morgan. I teoremi di De Morgan sono molto importanti per ottenere il complemento di una espressione.

- | | |
|--|---|
| 16. $\overline{X+Y}=\bar{X}\cdot\bar{Y}$ | 17. $\overline{X\cdot Y}=\bar{X}+\bar{Y}$ |
|--|---|

Il comportamento dell'operatore logico AND è duale a quello dell'operatore logico OR, vale infatti il principio di dualità.

Il principio di dualità afferma che data una eguaglianza se ne ottiene un'altra sostituendo l'operatore AND con l'operatore OR, 1 con 0 e viceversa. Ad esempio, osservando le prime proprietà notiamo come la 1. sia duale alla 2., così come la 3. è duale alla 4., la 5. alla 6., la 7. alla 8. etc..

Dimostriamo adesso il teorema di De Morgan (questa dimostrazione ci dà l'idea di come dimostrare uno qualunque dei teoremi e delle proprietà viste prima). A partire dalla relazione che afferma il teorema di De Morgan e che per comodità riscriviamo si costruisce una tabella di verità per il primo e per il secondo membro, l'uguaglianza che dobbiamo dimostrare è:

$$16. \overline{X+Y} = \overline{X \cdot Y}$$

Primo membro

X	Y	X+Y	$\overline{X+Y}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Secondo membro

X	Y	\overline{X}	\overline{Y}	$\overline{X \cdot Y}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Le due tabelle sono uguali, quindi il teorema di De Morgan è dimostrato.

Il vantaggio dell'algebra di Boole sta nel fatto di permettere, utilizzando i teoremi visti primi, la semplificazione dei circuiti digitali. Ad esempio sia F una funzione logica che vogliamo semplificare:

$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

Con l'uso delle proprietà viste prima possiamo semplificare la funzione nel seguente modo:

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

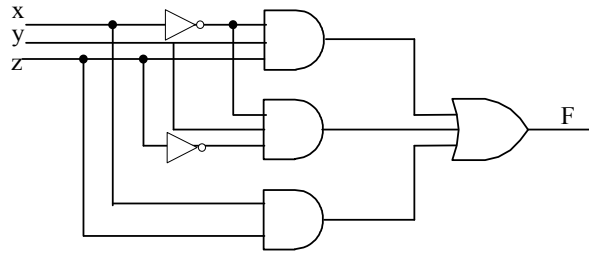
Le due funzioni, quella di partenza e quella ottenuta dalle semplificazioni, sono equivalenti, hanno cioè la stessa tabella di verità, tuttavia la seconda funzione è realizzabile con un circuito logico più semplice poiché richiede un numero inferiore di porte logiche.

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

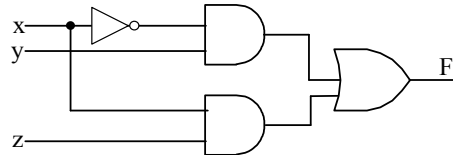
Nella progettazione di dispositivi logici la semplificazione delle funzioni che permettono di ottenere i risultati voluti manipolando i segnali in ingresso riveste un ruolo fondamentale.

Osservando i due circuiti è possibile notare quanto già detto:

Funzione logica non semplificata



Funzione logica semplificata



Il costo di realizzazione per un circuito logico è uno dei parametri che non deve mai essere sottovalutato poiché è ovvio che risulta più conveniente un circuito logico che realizza la medesima funzione con un numero di gates inferiori rispetto ad un altro.

Il complemento di una funzione è una funzione la cui tabella di verità riporta i valori invertiti rispetto alla funzione considerata, in corrispondenza di un valore logico 1 nella tabella di verità della funzione "sorgente" la funzione complemento riporta semplicemente il valore opposto e cioè 0, e viceversa. Il complemento di una funzione si ottiene applicando la seguente formula:

$$\overline{F}(a,b,c,\dots,+,.) = F(\overline{a},\overline{b},\overline{c},\dots,+,.)$$

Il teorema del **consensus** ci permette di semplificare una espressione Booleana di questo genere:

$$XY + \overline{X}Z + YZ = XY + \overline{X}Z$$

Come si può notare viene eliminato il termine YZ poiché risulta essere ridondante. Si noti che Y e Z sono associati a X e ad \overline{X} e appaiono insieme nel termine che è eliminato.

$$\begin{aligned} XY + \overline{X}Z + YZ &= XY + \overline{X}Z = XY + \overline{X}Z + YZ(X + \overline{X}) \\ &= XY + \overline{X}Z + XYZ + \overline{X}YZ \\ &= XY + XYZ + \overline{X}Z + \overline{X}YZ \\ &= XY(1 + Z) + \overline{X}Z(1 + Y) \\ &= XY + \overline{X}Z \end{aligned}$$

Un caso duale a quello visto è il seguente:

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

Le funzioni booleane possono essere scritte in vari modi, alcuni di questi modi sono considerati standard e necessitano delle definizioni di mintermine e maxtermine.

Considerando una riga della tabella di verità si definisce mintermine il prodotto delle variabili booleane relative a tale riga presa in forma diretta o complementata (cioè negata) a seconda che esse assumano valore 1 o 0.

Si definisce maxtermini la somma delle variabili booleane prese in forma diretta o negata a seconda che esse assumano valore 0 o 1. Quindi con n variabili abbiamo 2^n mintermini e maxtermini. Il modo di scrivere le funzioni booleane usando i mintermini o maxtermini è detto forma canonica o standard della/e funzione/i booleana/e. Ad esempio:

Elenco dei mintermini					Elenco dei maxtermini				
X	Y	Z	Prodotto	Simbolo	X	Y	Z	Somma	Simbolo
0	0	0	\overline{XYZ}	m_0	0	0	0	$X+Y+Z$	M_0
0	0	1	$\overline{XY}Z$	m_1	0	0	1	$X+Y+\bar{Z}$	M_1
0	1	0	$\overline{X}YZ$	m_2	0	1	0	$X+\bar{Y}+Z$	M_2
0	1	1	$\overline{X}YZ$	m_3	0	1	1	$X+\bar{Y}+\bar{Z}$	M_3
1	0	0	$X\bar{Y}\bar{Z}$	m_4	1	0	0	$\bar{X}+Y+Z$	M_4
1	0	1	$X\bar{Y}Z$	m_5	1	0	1	$\bar{X}+Y+\bar{Z}$	M_5
1	1	0	$XY\bar{Z}$	m_6	1	1	0	$\bar{X}+\bar{Y}+Z$	M_6
1	1	1	XYZ	m_7	1	1	1	$\bar{X}+\bar{Y}+\bar{Z}$	M_7

Il mintermine si denota con una qualunque lettera minuscola (generalmente m_j) mentre il maxtermine si denota con una qualunque lettera maiuscola (generalmente M_j). Il pedice j del mintermine è dato dall'equivalente decimale del numero binario che si ottiene dando valore 1 alla variabile in forma diretta, 0 per quella complementata. Per il maxtermine invece il pedice j è dato dall'equivalente del numero binario che si ottiene dando 1 alla variabile complementata e 0 alla variabile diretta. E' possibile ottenere la forma analitica di una funzione con due metodi:

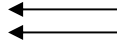
Somma di prodotti

Possiamo ottenere la forma analitica di una funzione a partire dalla tabella di verità nel seguente modo:

1. Si individuano le righe per cui la funzione F ha valore 1;
2. Si scrivono tanti prodotti per quante sono le righe individuate (ricordando che in questo caso le variabili 0 vanno prese dirette mentre le variabili 1 vanno prese negate o complementate);
3. Ogni prodotto è il mintermine relativo alla riga avente valore 1;
4. Si sommano i prodotti;

Ad esempio:

a	b	F
0	0	0
0	1	1
1	0	1
1	1	0



$$F = \bar{a}\bar{b} + \bar{a}b$$

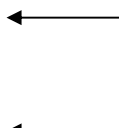
Prodotto di somme

A partire dalla tabella di verità della funzione F si procede in questo modo:

1. Si individuano le righe per cui F ha valore 0;
2. Si scrivono tante somme quante sono le righe individuate (ricordando che in questo caso le variabili 0 vanno prese in modo diretto mentre le variabili 1 vanno prese in modo negato o complementato);
3. Ogni somma è il maxtermine relativo alla riga avente valore 0;
4. Si effettua il prodotto delle somme;

Ad esempio:

a	b	F
0	0	0
0	1	1
1	0	1
1	1	0



$$F = (\bar{a} + \bar{b})(a + b)$$

Sia usando il primo che il secondo metodo si ottengono due funzioni booleane equivalenti (perché dotate della stessa tabella di verità) rappresentative di due circuiti logici diversi (perché usano porte logiche diverse) ma comunque equivalenti. Il vantaggio delle forme standard è quello di permettere la realizzazione delle funzioni con circuiti a due livelli: AND - OR oppure OR - AND.

Una funzione booleana può essere rappresentata, oltre che con la tabella di verità, con le cosiddette mappe di Karnough. Questa mappa è costituita da quadrati o celle. Due lati del quadrato sono contrassegnati dai valori delle variabili e ne caratterizzano la posizione. Le colonne e le righe presentano un ordine ciclico in modo che due celle adiacenti differiscano tra loro solo per il valore di un letterale. Le celle corrispondono ai mintermini di una funzione ad n variabili. Tutta la mappa meno la cella i corrisponde al maxtermine i . Queste mappe risultano abbastanza utili per rappresentare le funzioni in forma standard, se si usano i mintermini si devono considerare le celle contenenti i valori logici 1, se si usano i maxtermini quelle contenenti i valori logici 0. L'uso delle mappe di Karnough sono inoltre molto utili alla semplificazione di una funzione logica, pur avendo dei

limiti teorici. Consideriamo ad esempio una mappa di due sole variabili booleane:

m_0	m_1
m_2	m_3

$X \backslash Y$	0	1
0	$\overline{X}\overline{Y}$	$\overline{X}Y$
1	$X\overline{Y}$	XY

Le celle sono occupate dai rispettivi valori logici previsti dalla funzione in esame, sono possibili diverse situazioni ad esempio le seguenti mappe:

$X \backslash Y$	0	1
0	1	0
1	0	1

$$F = \overline{X}Y + X\overline{Y}$$

$X \backslash Y$	0	1
0	1	1
1	1	0

$$F = \overline{X}\overline{Y} + \overline{X}Y + X\overline{Y}$$

Mappa di Karnaugh per funzioni di tre variabili booleane, notare che: sulle colonne segniamo la sequenza di ingressi 00, 01, 11, 10 anziché la numerazione classica binaria che prevede 00, 01, 10, 11. Questo poiché è necessario cambiare una variabile di ingresso alla volta e mai due o più contemporaneamente, questo modo di procedere ci dà la possibilità di effettuare una semplificazione della funzione logica come vedremo più avanti.

$X \backslash YZ$	00	01	11	10
0	m_0	m_1	m_2	m_3
1	m_4	m_5	m_6	m_7

$X \backslash YZ$	00	01	11	10
0	$\overline{X}\overline{Y}\overline{Z}$	$\overline{X}\overline{Y}Z$	$\overline{X}YZ$	$\overline{X}Y\overline{Z}$
1	$X\overline{Y}\overline{Z}$	$X\overline{Y}Z$	XYZ	$XY\overline{Z}$

La mappa di Karnaugh per tre variabili booleane è lo sviluppo nel piano di un cilindro in cui le celle 0-2 e 4-6 sono allora tra loro adiacenti:

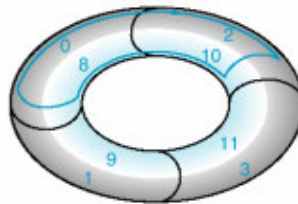
X\YZ	00	01	11	10
0	0	1	3	2
1	4	5	7	6



Mappa di Karnaugh per funzioni di quattro variabili booleane, sulle righe e sulle colonne possiamo adesso disporre due variabili e la numerazione corretta da seguire è 00, 01, 11, 10. La mappa di Karnaugh a quattro variabili è lo sviluppo nel piano di un toroide, quindi i mintermini o le celle 0-2-8-10 (quelle cioè poste negli spigoli della mappa) sono tra loro adiacenti:

ab\cd	00	01	11	10
00	m ₀	m ₁	m ₃	m ₂
01	m ₄	m ₅	m ₇	m ₆
11	m ₁₂	m ₁₃	m ₁₅	m ₁₄
10	m ₈	m ₉	m ₁₁	m ₁₀

ab\cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10



Le K-mappe permettono la semplificazione delle funzioni booleane. Supponiamo di avere la funzione espressa come somma di mintermini:

$$F = \sum(4,5,6,12,13)$$

allora, per ottenere la mappa di Karnaugh data la funzione come somma di mintermini occorre mettere un valore logico 1 nella cella corrispondente al mintermine indicato nella sommatoria:

ab\cd	00	01	11	10
00				
01	1	1		1
11	1	1		
10				

I mintermini 4 e 5 sono adiacenti, risulta che:

$$\overline{a}b\overline{c}d + \overline{a}bcd = \overline{a}b\overline{c}$$

Lo stesso per 12 e 13:

$$abc\overline{d} + abcd = abc$$

Abbiamo trasformato la somma di due prodotti di 4 variabili in un prodotto di 3 variabili, è saltato un letterale. Queste coppie di caselle adiacenti in cui si trova un valore logico 1 costituiscono un cosiddetto *accoppiamento a due*:

ab\cd	00	01	11	10
00				
01		1	1	
11		1	1	
10				

Questi accoppiamenti sono ancora contigui tra loro per cui sommandoli salta un altro letterale; si usa fare allora un cosiddetto *accoppiamento a quattro*:

ab\cd	00	01	11	10
00				
01		1	1	1
11		1	1	
10				

$$\overline{a}b\overline{c} + abc\overline{d} = b\overline{c}$$

$$F = b\overline{c} + \overline{a}bcd$$

E' possibile generare accoppiamenti a 2, 4, 8, 16, 2^i . In un accoppiamento 2^i si perdono i letterali. La ricerca deve partire dagli accoppiamenti più grandi perché solo in questo modo si perde un maggior numero di letterali semplificando notevolmente la funzione. In una tabella a 4 variabili gli accoppiamenti che si possono trovare sono a 2, 4, 8, 16.

L'ordine in cui si dispongono le variabili lungo le colonne e le righe non è importante, è possibile cioè invertire le colonne con le righe mantenendo sempre una numerazione degli ingressi adiacente per letterali, cambiando quest'ordine si otterrà una mappa di Karnough diversa (ruotata in un senso o in altro) la cui minimizzazione è comunque uguale alla stessa funzione booleana mappata diversamente da un secondo individuo.

Gli accoppiamenti devono essere tutti multipli di 2, cioè 2, 4, 8, 16 etc... Conviene, durante la fase di minimizzazione, cominciare dagli accoppiamenti più grandi per poi cercare quelli più piccoli fino a coprire l'intera mappa.

Un prodotto di termini implica una funzione quando per la combinazione delle variabili del termine per cui esso assume valore logico 1, anche la funzione f assume valore logico 1. Si dice Primo implicante (P) della funzione un prodotto di termini che implica la funzione f e tale che eliminando un qualunque letterale il prodotto rimanente non implica più f . Se un mintermine di una funzione è incluso solo in un primo implicante quest'ultimo si dice essenziale.

E' bene osservare che se è vero un solo termine della funzione (mintermine) è allora vera la funzione logica, se invece non è vero un mintermine nulla si può dire sull'intera funzione logica. Dato il concetto di implicanti, a questo punto osserviamo che gli accoppiamenti costituiscono primi implicanti purché non siano completamente interni ad accoppiamenti di ordine superiore. Ad esempio data la seguente mappa di Karnough:

AB\CD	00	01	11	10	
00		1	1		\overline{AD}
01	1	1	1	1	\overline{AB}
11	1			1	
10					$B\overline{D}$

Sono primi implicanti essenziali:

$$\overline{AD} \text{ e } B\overline{D}$$

Infatti la funzione può essere scritta come somma dei primi implicanti essenziali:

$$F = \overline{AD} + B\overline{D}$$

mentre \overline{AB} non è un primo implicante essenziale perché non contiene mintermini coperti esclusivamente da esso.

Vi possono essere delle funzioni non completamente specificate. Funzioni in cui, in corrispondenza a certi valori di ingresso, non si vuole un fissato valore di uscita, ovvero qualunque valore dell'uscita è accettabile. Nella mappa di Karnough questi stati si indicano con d (sta per **don't care** = non importa) oppure X oppure ancora con -. In fase di semplificazione di una funzione ad essi si può assegnare valore 1 o 0 a seconda se permettono di ottenere accoppiamenti più grandi e quindi maggiori semplificazioni.

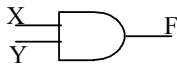
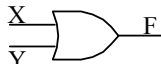
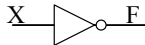
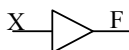
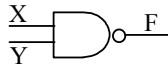
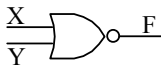
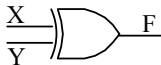
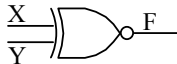
Ad esempio:

ab\cd	00	01	11	10
00	-	1	1	-
01		-	1	
11			1	
10			1	

$$F = \overline{ab} + cd$$

Abbiamo più volte detto come sia possibile esprimere una funzione booleana tramite i gate AND, OR e NOT. Tuttavia ci sono altri gate

di particolare interesse e che da soli permettono di rappresentare la funzioni come somma di prodotti (è questo il caso della porta NAND) oppure come prodotto di somme (è questo il caso della porta NOR).

Nome	Simbolo	Equazione algebrica	Tab. di verità															
AND		$F = XY$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	F	0	0	0	0	1	0	1	0	0	1	1	1
X	Y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = X + Y$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	F	0	0	0	0	1	1	1	0	1	1	1	1
X	Y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{X}$	<table><tr><td>X</td><td>F</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	X	F	0	1	1	0									
X	F																	
0	1																	
1	0																	
Buffer		$F = X$	<table><tr><td>X</td><td>F</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	X	F	0	0	1	1									
X	F																	
0	0																	
1	1																	
NAND		$F = \overline{X \cdot Y}$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	X	Y	F	0	0	1	0	1	1	1	0	1	1	1	0
X	Y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{X + Y}$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	X	Y	F	0	0	1	0	1	0	1	0	0	1	1	0
X	Y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = X\overline{Y} + \overline{X}Y$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	X	Y	F	0	0	0	0	1	1	1	0	1	1	1	0
X	Y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$F = XY + \overline{X\overline{Y}}$	<table><tr><td>X</td><td>Y</td><td>F</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	F	0	0	1	0	1	0	1	0	0	1	1	1
X	Y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Non è possibile utilizzare la mappa di Karnaugh per funzioni a più di quattro variabili booleane, tuttavia, per funzioni logiche di 5 variabili si adotta uno schema come quello in figura:

		e = 0				e = 1			
ab \ cd	cd	00	01	11	10	00	01	11	10
	ab								
	00	1				1			
	01	1				1			
	11	1				1			
	10	1				1			

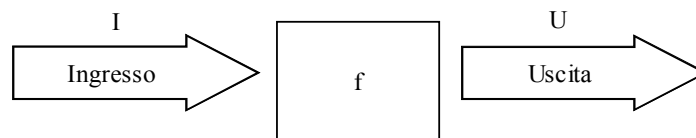
c d

Si riportano cioè due tabelle, in una $e=0$ e nell'altra $e=1$. Per effettuare la semplificazione tra due o più mintermini adiacenti occorre immaginare di sovrapporre le due mappe e di osservare quindi le celle adiacenti, considerando anche i possibili accoppiamenti che si possono realizzare tra i due piani.

LEZIONE N°2: "La progettazione di macchine combinatorie"

Ancor prima di entrare nel dettaglio di come si progetta una rete combinatoria è preferibile dare alcune nozioni di base che riguardano i diversi tipi di macchine esistenti.

Esistono due tipi di macchine, uno di tipo **sequenziale** ed uno di tipo **combinatorio**. Una rete combinatoria è una macchina in cui l'uscita è funzione soltanto dei valori di ingresso e genera lo stesso output se sollecitato dal medesimo input (ad esempio una calcolatrice), al contrario invece, in una macchina sequenziale l'uscita è funzione oltre che dall'ingresso anche dallo stato interno generato da un precedente ingresso. Per le macchine combinatorie, talvolta, si usa assegnare uno schema molto semplice come:



Una macchina combinatoria propone in uscita sempre lo stesso risultato in risposta al medesimo risultato

Dove con I e con U si intendono rispettivamente gli ingressi e le uscite mentre con f si vuole indicare la funzione o il metodo con cui lavora la macchina combinatoria.

Le reti combinatorie sono largamente usate nel settore informatico, basti pensare ai decodificatori, multiplexer, addizionatori, controllori etc...

Il progetto delle reti combinatorie viene schematizzato nei suoi seguenti punti chiave quali:

- Definizione della macchina;
- Codifica degli ingressi e delle uscite;
- Costruzioni della funzione logica;
- Minimizzazione e scelta della forma algebrica;
- Realizzazione del circuito;
- Adattamento del circuito;

Vediamo nel dettaglio ciascuno dei punti chiavi elencati. In realtà i passi sopra elencati sono validi anche per le reti sequenziali, tuttavia andando avanti nel corso il problema sarà affrontato con altri schemi.

Definizione della macchina:

Costituisce il primo passo da compiere per la progettazione di una macchina combinatoria ed è inteso come la fase a più alto livello nel progetto. E' indispensabile, per il proseguimento del progetto, chiarire i compiti o le funzioni che la macchina dovrà svolgere. In questa fase è importante raccogliere quante più informazioni possibili sulla macchina combinatoria che si vuole realizzare poiché, come vedremo più avanti, alcuni comportamenti della macchina non descritti o comunque non contemplati dalle specifiche del problema potrebbero poi rivelarsi utili alla progettazione

semplificandone il costo realizzativo (vedi don't care e semplificazione delle mappe di Karnough). Quando le regole che comandano la macchina sono state focalizzate è opportuno riconoscere e definire con precisione l'insieme degli stati di ingresso e di uscita.

Codifica degli ingressi e delle uscite:

Costituiscono il secondo passo della progettazione, la codifica degli ingressi e delle uscite; occorre in questa fase definire ciascuno degli ingressi e ciascuna delle uscite scegliendo una rappresentazione idonea al problema. Differenti codici portano a macchine completamente differenti. Questa fase della progettazione è l'unica fase che non è automatizzabile, anche se esistono software per calcolatori i quali presi gli ingressi della macchina e le uscite che essa deve fornire si incaricano di mostrare all'utente una delle migliori rappresentazioni per la codifica. Tuttavia in ogni caso solo il progettista ha la facoltà di scegliere una rappresentazione piuttosto che un'altra. Una scelta accurata della codifica permette spesso di semplificare il progetto di una macchina, talvolta una codifica dei dati può addirittura semplificare il problema in maniera clamorosa, altre volte, invece, una codifica degli ingressi e delle uscite può rendere il problema della progettazione ancora più difficile. L'esperienza, e la conoscenza dei passi successivi alla progettazione, può aiutare il progettista a compiere una scelta adeguata o comunque a trovare il giusto compromesso tra difficoltà e realizzazione del progetto. A partire dal prossimo passo, fino alla fine, è possibile rendere la progettazione di una macchina combinatoria automatizzabile.

Costruzione della funzione logica:

Questa fase è considerata da molti la prima e vera fase di progetto della macchina. Nonostante ciò questa fase richiede che siano stati risolti i problemi legati alle specifiche della macchina e della codifica degli ingressi e delle uscite.

Quindi, definita la macchina e risolti i problemi sulla codifica dei dati è possibile effettuare la costruzione della funzione logica. Noto il rapporto I/O e la codifica è possibile arrivare a definire le operazioni logiche da realizzare. In altre parole è possibile individuare la "tabella di verità" della macchina.

Le variabili di ingresso vengono disposte in maniera ordinata secondo delle colonne mentre sulle righe si alternano i possibili valori in ingresso in maniera tale da tracciare per ogni riga (e quindi per ogni ingresso) la corretta uscita.

Minimizzazione e scelta della forma algebrica:

A questo punto il progetto della macchina è già in parte realizzato (almeno su carta); occorre semplificare, ove è possibile, la funzione compiendo una minimizzazione e scegliendo una delle forme algebriche per la rappresentazione della funzione. Utilizzando l'algebra di Boole è possibile individuare una forma algebrica minima della funzione logica. Oltre ai teoremi per la semplificazione della funzione logica occorre in questa fase della progettazione tracciare essenzialmente la mappa di Karnough della

funzione logica ed effettuare quindi la minimizzazione secondo la tecnica dei raggruppamenti 2-4-8-16 e l'uso degli implicanti.

Realizzazione del circuito:

La realizzazione del circuito scaturisce dalla forma algebrica ricavata al passo precedente e costituisce un processo puramente meccanico. Una semplificazione accurata della funzione consiste nell'usare un numero di gates sempre più piccolo e facendo abbassare il costo di realizzazione del circuito. Il circuito sarà quindi provato al simulatore per verificarne l'efficienza e la validità logica. Alcuni problemi dettati dai limiti elettronici impongono alcune condizioni sul modo di realizzare la rete come i problemi dovuti al carico elettrico. Questo ed altri problemi come il numero di porte pilotate in uscita che si hanno a disposizione, oppure il numero di ingressi accettabili e quindi gestibili da una porta logica rientrano nell'ultima fase di progettazione detta di *adattamento del circuito*.

Per la conoscenza delle reti fondamentali, e l'utilizzo che ne verrà fatto in fase avanzata del corso, vedremo in seguito la progettazione di alcune macchine largamente usate, alla quale cercheremo di applicare lo schema appena visto.

La rete di parità

La rete di parità è una macchina combinatoria abbastanza diffusa che serve a stabilire se una sequenza di bit in ingresso è pari oppure dispari. L'uso di questi circuiti è largamente diffuso nella gestione delle memorie r.a.m. per calcolatori. La memoria può essere allocata secondo due schemi diversi: little-endian e big-endian. Nello schema little-endian il bit più significativo si trova collocato più a destra rispetto alle altre celle di memoria, viceversa per lo schema big-endian. Siccome gli indirizzi di memoria vengono trattati con sequenze di bit pari o dispari la rete di parità è usata in fase di controllo o generazione per gli indirizzi pari o dispari. In fase di generazione di un indirizzo, se $x_0, x_1, x_2, \dots, x_7$ sono gli 8 bit di un byte un ulteriore bit x_8 viene aggiunto affinché la stringa assuma una parità o una disparità in base ai bit fin lì assunti. Analogamente, in fase di controllo se $x_0, x_1, x_2, \dots, x_7$ sono gli 8 bit di un byte il successivo bit x_8 indica la parità o la disparità della stringa.

La rete di parità è dotata generalmente di n ingressi ed una sola uscita. L'uscita è dunque un segnale binario, si sceglie infatti di assegnare il bit 1 se la sequenza è pari, oppure 0 se la sequenza di bit in ingresso è dispari (rete di parità in senso stretto).

Esistono sia reti di parità, per le quali $y=1$ indica la parità dei bit in ingresso, e sia reti di disparità in cui l'uscita $y=1$ indica che i bit in ingresso sono dispari. Essendoci n bit di ingresso è allora possibile avere 2^n possibili combinazioni.

Nel caso della progettazione di una macchina di parità la codifica non gioca un ruolo importante, essa è, in questo caso, già scelta poiché sia gli ingressi e l'uscita sono dei bit. Progettiamo, ad esempio, una rete di parità avente 3 bit in ingresso ed un bit in uscita:

Ingresso

b_1, b_2 e b_3 sono i bit ingresso,
3 bit in ingresso generano $2^3=8$
possibili ingressi

Uscita

y è il bit di uscita
 $y=1$ (pari) oppure $y=0$ (dispari)

Costruiamo la tabella di verità per la nostra rete:

	b_1	b_2	b_3	y
1	0	0	0	1
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	0

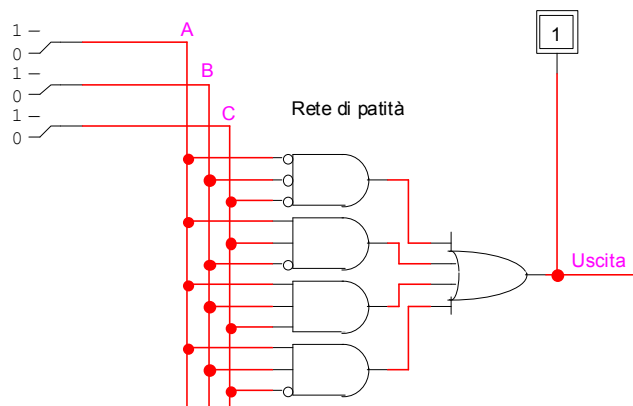
Quindi tracciamo la mappa di Karnaugh:

$b_3 \backslash b_1 b_2$	00	01	11	10
0	1	0	1	0
1	0	1	0	1

La mappa di Karnaugh per una qualunque rete di parità o di disparità si presenterà sempre come una scacchiera poiché gli elementi pari o dispari si trovano alternati tra di loro. Questa disposizione della tabella di verità non ci permette alcuna minimizzazione poiché non è possibile formare nessun accoppiamento di quelli 8-4-2 visti in precedenza. Dalla tabella di verità si estrae questa funzione logica:

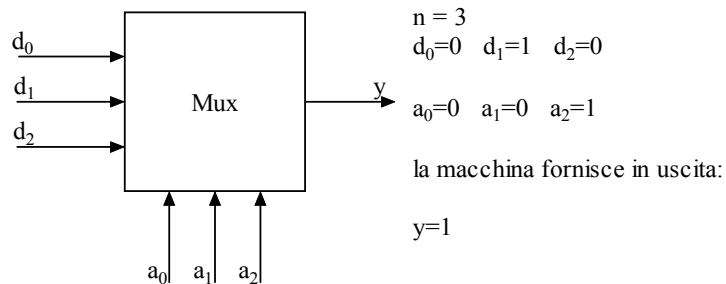
$$y = \underline{b_1} \underline{b_2} \underline{b_3} + \underline{b_1} b_2 b_3 + b_1 \underline{b_2} b_3 + b_1 b_2 \underline{b_3}$$

La rete di parità a 3 bit di ingresso avrà il seguente schema:



Multiplexer e multiplexer indirizzabile

Il multiplexer binario è una macchina combinatoria abbastanza semplice che è però caratterizzato da un elevato numero di ingressi. Il primo multiplexer che vediamo è quello semplice poiché esso prevede n ingressi dati ed n ingressi indirizzo. Il numero di ingressi dati e quello per gli indirizzi sono uguali. Gli ingressi sono suddivisi in ingressi dati ed ingressi indirizzo solo per avere una maggiore comprensione sul comportamento della macchina, nella realtà il multiplexer binario ha $2n$ ingressi. (il fatto che alcuni di questi $2n$ indirizzi siano riservati ai dati ed altri agli indirizzi è una nostra scelta o modo di dire che ci agevola nello studio del dispositivo). Il multiplexer binario è dotato di una sola uscita, il bit y . L'uscita è il valore di uno dei bit di ingresso dati selezionato attraverso il bit ingresso indirizzo. Una cosa importante da notare è il fatto di adottare una codifica molto ridondante, infatti dei $2n$ ingressi solo n sono utili. Si può immaginare di numerare ciascuno degli ingressi dati e indirizzi. In base al numero binario che si compone sulla linea indirizzi il dispositivo pone in uscita, sul bit y , il valore logico della linea dati corrispondente. Ad esempio, per $n=3$:



All'ingresso $d_0=0$ $d_1=1$ e $d_2=0$, che corrisponde (secondo le specifiche della macchina descritte prima) a prelevare il bit contenuto nel bit 3 (cioè a_2) dell'ingresso indirizzo, la macchina risponde con una uscita $y=1$, essendo $a_2=1$. Per ovviare a questo problema, dovuto ad una codifica ridondante, si è solito procedere nella progettazione di un multiplexer indirizzabile dotato di queste specifiche:

- n ingressi dati;
- $\log_2 n$ ingressi indirizzo;
- Una sola uscita;

Gli ingressi indirizzo sono $\log_2 n$.

Un multiplexer indirizzabile è una macchina che inoltra sull'unica uscita y quello fra gli n bit-dati in ingresso il cui indirizzo è quindi individuato da $k=\log_2 n$ bit-indirizzo.

A titolo di esempio progettiamo un multiplexer a quattro ingressi, pertanto essendo $n=4$ il mux avrà 4 ingressi dati e $\log_2 4=2$ ingressi indirizzo che denotiamo come:

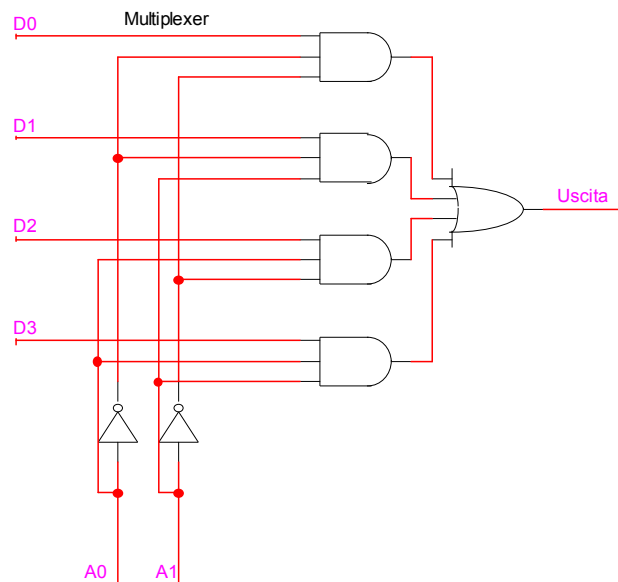
- d_0 , d_1 , d_2 e d_3 sono gli ingressi dati;
- a_1 , a_2 ;
- z è l'uscita;

1 d ₀	2 d ₁	3 d ₂	4 d ₃	a ₁	a ₂	z
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	0
0	0	0	1	1	1	1
continua...						

La tabella della verità per questa macchina è molto lunga, è comunque utile, per capire ancora meglio il comportamento della macchina, osservare i primi ingressi. Osserviamo la prima riga, in ingresso ci sono tutti zero, gli ingressi indirizzo a₁ ed a₂ sono posti entrambi a zero, essi indicano di prendere il bit contenuto nell'ingresso dati d₀, la macchina riporta in uscita z=0. L'ultima riga della tabella è ancora più significativa. I bit di ingresso indirizzo sono a₁=1 ed a₂=1, stanno cioè indicando il contenuto del bit 4 dell'ingresso dati, che è d₃. Il contenuto di d₃, cioè il bit 1 è riportato in uscita per cui z=1:

a ₁	a ₂	z
0	1	d ₁
0	0	d ₂
1	1	d ₃
1	0	d ₄

Lo schema logico è il seguente:

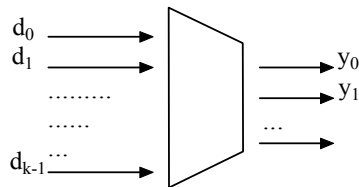


Decodificatori

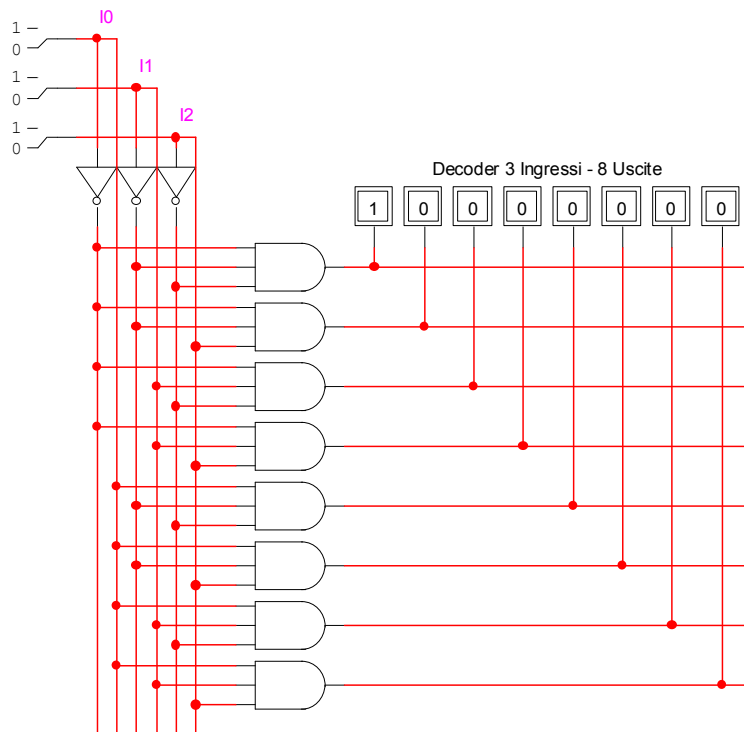
Gran parte dell'informazione nei calcolatori è manipolata in forma codificata. In un'istruzione, si può usare un campo di n bit per selezionare 1 tra 2^n possibili azioni da intraprendere. Per eseguire l'azione desiderata, l'istruzione codificata deve essere prima decodificata. Un circuito in grado di accettare un ingresso a n bit e di generare il corrispondente segnale di uscita su una delle 2^n possibili uscite si chiama decodificatore. Un decodificare completo è una macchina con:

- k ingressi;
- $M=2^k$ uscite;

tale che al più una delle uscite è alta. Il decodificatore completo talvolta detto anche demultiplexer (de-mux) è generalmente schematizzato in questo modo:



Un esempio di decoder con $K=3$, dove K è il numero di ingressi:



Full adder

La macchina è anch'essa di tipo combinatorio ed effettua la somma dei due bit ingresso e del bit di riporto, in uscita la macchina propone con un bit la somma $s=a+b$ e con un altro bit l'eventuale riporto. Riepilogando, la macchina ha 3 bit di ingresso e 2 bit di uscita:

Ingresso

a, b ed r_{in} sono i bit di ingresso

Uscita

s ed r_{out} sono i bit di uscita

$$s = a+b+r_{out}$$

Trattandosi di 3 bit di ingresso, la macchina potrà avere $2^3=8$ possibili combinazioni di ingresso. La codifica in questo caso non consiste alcun problema poiché si tratta di bit. Inoltre, avendo due uscite, la macchina si comporrà di due mappe di Karnaugh, ciascuna delle quali farà riferimento rispettivamente ad un bit di uscita e quindi al bit s e al bit r_{out} .

Tabella della verità:

a	b	r_{in}	s	r_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mappe di Karnaugh.

Per s:

$a \backslash b r_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Per r_{out} :

$a \backslash b r_{in}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Minimizzazione: non è possibile minimizzare la prima mappa essendo disposta secondo una scacchiera. La seconda mappa, quella riferita al bit r_{out} , è invece minimizzabile secondo tre accoppiamenti di 2 variabili booleane. Le funzioni logiche che realizzano l'uscita sono:

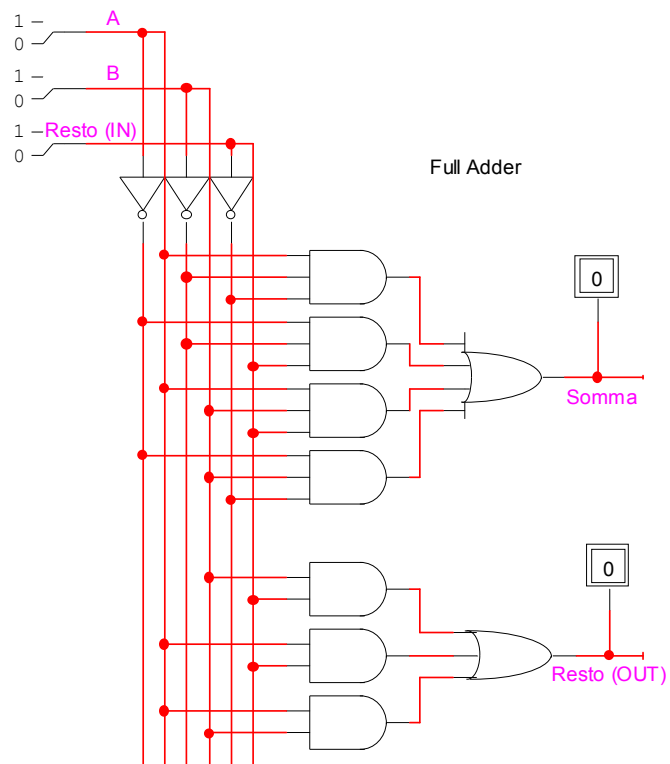
Per s:

$$f(s) = \underline{a} \underline{b} r_{in} + \underline{a} b r_{in} + a \underline{b} r_{in} + a b \underline{r}_{in}$$

Per r_{out} :

$$f(r_{out}) = a r_{in} + b r_{in} + a b$$

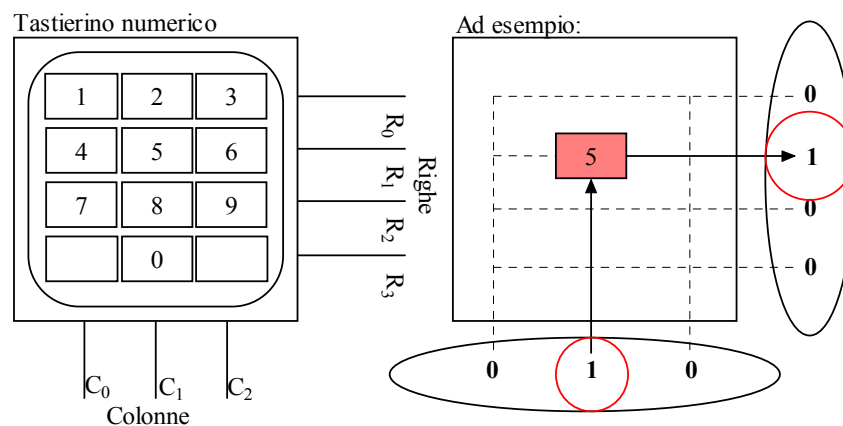
Il circuito logico del full adder è:



Lezione N°3: "Il tastierino numerico"

Progettare una funzione logica che realizzi come uscita 4 bit associati ai numeri che vanno da 0 a 9 (per un totale di 10 simboli, con 4 bit si hanno $2^4=16$ possibili rappresentazioni), dati 7 bit di ingresso, di cui 4 bit riferiti alle righe della tastiera e 3 bit riferiti alle colonne della tastiera. Il funzionamento è estremamente semplice, la selezione di un numero sulla tastiera è identificato da due stringe di bit, una identifica la colonna di appartenenza del tasto selezionato e l'altra invece identifica la riga. I tasti collocati a destra ed a sinistra dello 0 (solitamente si tratta dei simboli # e *) non sono specificati, il loro utilizzo non è contemplato in questo progetto, inoltre il comportamento della macchina non è specificato qualora due o più tasti siano pigiati contemporaneamente.

Ad esempio:



I dati di ingresso sono codificati in binario, l'uscita è invece codificata secondo le specifiche 8-4-2-1 che è una codifica (binaria) basata su 4 bit (ad esempio l'uscita 1000=8, 0001=1, 0101=5 etc...). L'analisi della tabella è estremamente complessa poiché essa si compone di 128 righe ed 11 colonne. Tuttavia ci sono molti ingressi non contemplati che corrispondono a tutti gli ingressi dotati di due o più valori logici alti per ciascuna riga e per ciascuna colonna (cioè gli ingressi equivalenti ai casi in cui vengono pigiati due o più tasti).

Dovendo progettare una macchina con 7 ingressi (3 bit per le colonne e 4 bit per le righe) e 4 uscite (i bit necessari alla codifica 8-4-2-1) non è possibile avvalersi delle mappe di Karnough, pertanto, l'unico modo per minimizzare le funzioni logiche è quello di semplificare quest'ultime mediante l'uso dei teoremi già visti. La tabella di verità per questo genere di funzioni è più facile se ordinata secondo le uscite, ad esempio:

C ₀	C ₁	C ₂	r ₀	r ₁	r ₂	r ₃	Tasto premuto	8 y ₀	4 y ₁	2 y ₂	1 y ₃
1	0	0	1	0	0	0	1	0	0	0	1
0	1	0	1	0	0	0	2	0	0	1	0
0	0	1	1	0	0	0	3	0	0	1	1
1	0	0	0	1	0	0	4	0	1	0	0
0	1	0	0	1	0	0	5	0	1	0	1
0	0	1	0	1	0	0	6	0	1	1	0
1	0	0	0	0	1	0	7	0	1	1	1
0	1	0	0	0	1	0	8	1	0	0	0
0	0	1	0	0	1	0	9	1	0	0	1
0	1	0	0	0	0	1	0	0	0	0	0

Per ogni uscita occorre realizzare una funzione logica, quindi, per y₀:

$$y_0 = \underline{C}_0 C_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 \underline{C}_1 C_2 r_0 r_1 r_2 r_3$$

Per y₁:

$$y_1 = C_0 \underline{C}_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 C_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 \underline{C}_1 C_2 r_0 r_1 r_2 r_3 + C_0 \underline{C}_1 \underline{C}_2 r_0 r_1 r_2 r_3$$

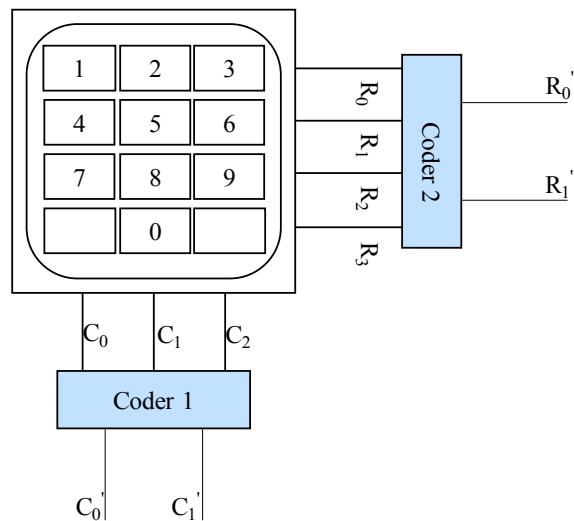
Per y₂:

$$y_2 = \underline{C}_0 C_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 \underline{C}_1 C_2 r_0 r_1 r_2 r_3$$

Per Y₃:

$$y_3 = C_0 \underline{C}_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 \underline{C}_1 C_2 r_0 r_1 r_2 r_3 + \underline{C}_0 C_1 \underline{C}_2 r_0 r_1 r_2 r_3 + C_0 \underline{C}_1 \underline{C}_2 r_0 r_1 r_2 r_3 + \underline{C}_0 \underline{C}_1 C_2 r_0 r_1 r_2 r_3$$

Alcune considerazioni: le uscite C₀-C₂ e R₀-R₃ del tastierino telefonico sono due uscite codificate. Entrambe possono essere ulteriormente codificate, infatti: C₀-C₁-C₂ possono essere codificati con due bit (in tal caso delle 4 possibili rappresentazioni una sola di queste verrebbe ad essere inutile); R₀-R₁-R₂-R₃ possono essere codificati con due bit utilizzando in questo caso tutte le possibili rappresentazioni. Il progetto di partenza subisce allora una leggera modifica:



In questo caso abbiamo due macchine intermedie che codificano le uscite. La macchina risultante è una macchina con solo 4 uscite anziché 7. Per il coder 1:

- 3 bit di ingresso decodificati;
- 2 bit di uscita decodificati;

il gran numero di don't care per il coder 1 semplificherà la minimizzazione:

C_0	C_1	C_2	C_0'	C_1'
0	0	0	-	-
0	0	1	0	0
0	1	0	0	1
0	1	1	-	-
1	0	0	1	0
1	0	1	-	-
1	1	0	-	-
1	1	1	-	-

Vediamone la minimizzazione:

$C_2 \backslash C_0 C_1$	00	01	11	10
0	-	0	-	1
1	0	-	-	-

$C_2 \backslash C_0 C_1$	00	01	11	10
0	-	1	-	0
1	0	-	-	-

Per il coder 2:

- 4 bit di ingresso decodificati;
- 2 bit di uscita decodificati;

La tabella della verità è:

Per $R_0=0$:

R_1	R_2	R_3	R_0'	R_1'
0	0	0	-	-
0	0	1	0	0
0	1	0	0	1
0	1	1	-	-
1	0	0	1	0
1	0	1	-	-
1	1	0	-	-
1	1	1	-	-

Per $R_0=1$:

R_1	R_2	R_3	R_0'	R_1'
0	0	0	1	1
0	0	1	-	-
0	1	0	-	-
0	1	1	-	-
1	0	0	-	-
1	0	1	-	-
1	1	0	-	-
1	1	1	-	-

Minimizzazione:

$R_2R_3 \backslash R_0R_1$	00	01	11	10
00	-	1	-	1
01	0	-	-	-
11	-	-	-	-
10	0	-	-	-

$R_2R_3 \backslash R_0R_1$	00	01	11	10
00	-	0	-	1
01	0	-	-	-
11	-	-	-	-
10	1	-	-	-

Lezione N°4: Esercizi fatti in classe

Esercizio N°1:

Progettare una macchina combinatoria che ha in ingresso 4 bit e in uscita 2 bit detti b_0 e b_1 , il bit b_0 è alto se il numero di zeri è pari, il bit b_1 è alto se il numero di zeri è maggiore di 1.

Ingressi

Uscite

a_0, a_1, a_2 ed a_3 sono i bit di ingresso, con 4 bit si hanno $2^4=16$ possibili combinazioni.

b_0 e b_1 sono i due bit di uscita, il bit b_0 è alto se il numero di zeri è pari, il bit b_1 è alto se il numero di zeri è maggiore di 1.

La tabella della verità è:

a_0	a_1	a_2	a_3	b_0	b_1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	0

Le mappe di Karnaugh sono:

Per b_0 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

Per b_1 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	1	1	1	1
01	1	1	0	1
11	1	0	0	0
10	1	1	0	1

Le funzioni logiche sono, per b_0 :

$$f(b_0) = \underline{a_0} \underline{a_1} \underline{a_2} \underline{a_3} + \underline{a_0} \underline{a_1} a_2 a_3 + a_0 a_1 a_2 a_3 + \underline{a_0} a_1 \underline{a_2} a_3 + \underline{a_0} a_1 a_2 \underline{a_3} + a_0 a_1 \underline{a_2} \underline{a_3} + a_0 \underline{a_1} \underline{a_2} a_3 + a_0 \underline{a_1} a_2 \underline{a_3}$$

Per b_1 :

$$f(b_1) = \underline{a_0} \underline{a_2} + \underline{a_2} \underline{a_3} + a_0 \underline{a_1} \underline{a_2} + \underline{a_0} \underline{a_1} a_2 + a_2 \underline{a_3} \underline{a_0} + \underline{a_3} \underline{a_1}$$

Esercizio N°2:

Progettare una rete combinatoria che riceve in ingresso un numero codificato da 0 a 9 su 4 bit e fornisce come uscita la somma $n+3$ codificata sempre in 4 bit.

Ingressi

Uscite

I bit a_0, a_1, a_2 ed a_3 sono bit di I bit y_0, y_1, y_2 ed y_3 sono bit ingresso, con 4 bit è possibile di uscita.

$2^4=16$ diversi ingressi.

La prima cosa da osservare è il fatto che si sta utilizzando una codifica ridondante, infatti, in ingresso con 4 bit è possibile rappresentare $2^4=16$ segni mentre se ne usano solo 10, quindi 6 rappresentazioni sono ignorate e costituiscono casi di don't care. Anche per l'uscita il codice è ridondante. Il massimo valore che può capitare all'uscita della macchina è $12=9+3$, in questo caso dei 12 segni utili alla macchina non se ne utilizzano 4. La tabella della verità è:

a_0	a_1	a_2	a_3	$n+3$	y_0	y_1	y_2	y_3
0	0	0	0	3	0	0	1	1
0	0	0	1	4	0	1	0	0
0	0	1	0	5	0	1	0	1
0	0	1	1	6	0	1	1	0
0	1	0	0	7	0	1	1	1
0	1	0	1	8	1	0	0	0
0	1	1	0	9	1	0	0	1
0	1	1	1	10	1	0	1	0
1	0	0	0	11	1	0	1	1
1	0	0	1	12	1	1	0	0
1	0	1	0	-	-	-	-	-
1	0	1	1	-	-	-	-	-
1	1	0	0	-	-	-	-	-
1	1	0	1	-	-	-	-	-
1	1	1	0	-	-	-	-	-
1	1	1	1	-	-	-	-	-

Per ogni bit di uscita ci sarà una mappa di Karnough:

Per y_0 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	d	d	d	d
10	1	1	d	d

La funzione logica è:

$$f(y_0) = a_0 + a_1a_3 + a_1a_2a_3$$

Per y_1 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	d	d	d	d
10	0	1	d	d

La funzione logica è:

$$f(y_1) = \underline{a}_1a_2 + a_1\underline{a}_2\underline{a}_3 + \underline{a}_1a_3$$

Per y_2 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	d	d	d	d
10	1	0	d	d

La funzione logica è:

$$f(y_2) = \underline{a}_2a_3 + a_2a_3$$

Per y_3 :

$a_0a_1 \backslash a_2a_3$	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	d	d	d	d
10	1	0	d	d

La funzione logica è:

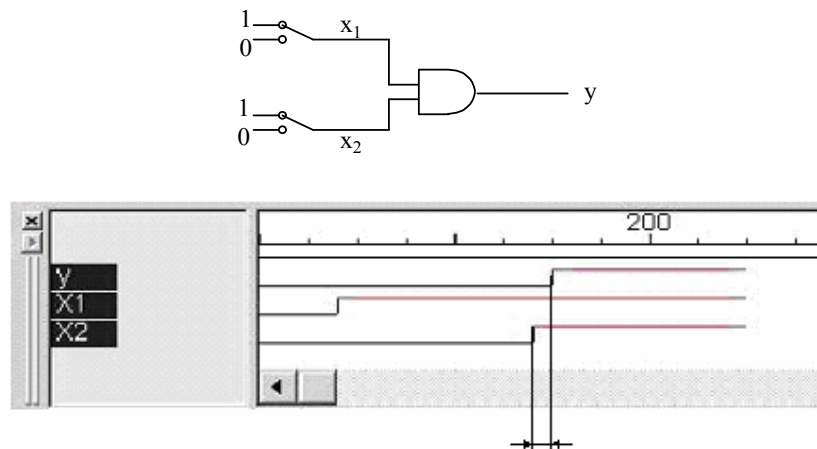
$$f(y_3) = \underline{a}_3$$

Lezione N°5: "La tempificazione"

Finora abbiamo effettuato il progetto delle macchine combinatorie trascurando gli effetti temporali sugli ingressi I della macchina. In realtà, durante la fase di progettazione, bisogna tener conto non solo dell'aspetto logico della macchina combinatoria ma è opportuno studiarne anche il comportamento seguendo l'evoluzione nel tempo che le variabili d'ingresso subiscono.

Le macchine ideali reagiscono istantaneamente ad un segnale, esse rappresentano tuttavia un modello ben lontano dalla realtà, infatti, una sostanziale differenza con le macchine reali è il tempo di risposta che esse mostrano nei confronti di un ingresso. Ogni sistema, anche fisico, impiega un certo tempo per rispondere alla sollecitazione che gli viene mossa, è questo il concetto da tenere in considerazione per realizzare macchine più vicine alla realtà.

Pertanto, uno dei primi concetti che dobbiamo necessariamente introdurre è il tempo di risposta. Il tempo di risposta di una macchina è il ritardo $d=t_f-t_i$ con il quale una variazione sull'ingresso è seguita da una variazione sull'uscita. Ad esempio consideriamo una semplice rete logica:



La rete logica è costituita dalla sola porta AND, la cui uscita è appunto x_1 AND x_2 . Nel diagramma che immediatamente segue sono invece riportati i valori delle singole variabili di ingresso e dell'uscita y . Nel tratto iniziale del diagramma temporale, quello cioè in cui entrambi le variabili di ingresso sono ancora 0 non vi è alcun problema, visto che gli ingressi non cambiano. Osservando invece l'ultima parte del diagramma, quello in cui vengono a trovarsi entrambe le variabili x_1 ed x_2 ad 1, notiamo che la risposta y in uscita non assume subito il valore logico (alto) che gli compete, ma in corrispondenza del tratto in cui l'ultima variabile d'ingresso, e cioè x_2 , raggiunge anch'essa un valore logico alto si nota che l'uscita y attende un certo tempo prima di portarsi anch'essa sul valore logico alto, ecco quindi il manifestarsi di un ritardo. Nell'intervallo di tempo che va dall'istante in cui x_2 si alza all'istante in cui l'uscita assume il giusto valore logico la macchina si comporta in modo errato e può causare problemi dovuti a funzionamenti impropri oltre a generare un uscita non corretta. Con lo scopo di modellare questo

effetto di ritardo si introduce un dispositivo detto appunto dispositivo di ritardo iniziale. Si chiama ritardo inerziale E un elemento $I(t)$ e la cui uscita $U(t)$ rispettino le seguenti condizioni:

Se E è il ritardo ed $I(t)$ l'ingresso in funzione del tempo, allora:

$$I(t-\varepsilon) \neq I(t) \text{ per } \varepsilon > 0$$

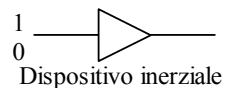
ed:

$$I(\tau) = I(t) \text{ per } t \leq \tau < t + E$$

pertanto:

$$U(\tau) = I(t - \varepsilon) \text{ e } U(t + E) = I(t)$$

In altre parole il dispositivo di inerzia non fa nulla!!! Esso si limita semplicemente a riproporre l'ingresso ricevuto dopo un certo tempo t. Quest'oggetto è tipico dei simulatori logici che dovendo modellare e simulare una rete logica necessitano di questi e di altri strumenti per avvicinare il progetto quanto più possibile alla realtà. Il suo simbolo è il seguente:



Esso è simile all'operatore di not, infatti tutti gli operatori unari si rappresentano in questo modo. Il dispositivo, seguendo la sua definizione, se sollecitato ad un ingresso all'istante t rimane nello stato precedente per un tempo E, che appunto il ritardo inerziale.

La presenza di ritardi nei dispositivi utilizzati in alcuni casi può avere l'effetto di modificare il comportamento delle uscite. Ecco allora l'altro concetto fondamentale da ricordare. Si chiamano alee quei fenomeni per i quali le uscite, anche se solo per brevi intervalli di tempo, assumono dei valori imprevisti. In tal caso il comportamento della rete risulterebbe aleatorio in quanto legato alle caratteristiche incontrollabili dei ritardi. Una delle prime condizioni di funzionamento da rispettare affinché una macchina combinatoria funzioni a dovere è la seguente:

$$f_{\max} = \frac{1}{2R}$$

La variazione di frequenza con la quale le variabili si alternano o comunque cambiano agli ingressi della macchina può avvenire al massimo con periodo $2R$, con R ritardo della macchina. Superato il valore di f_{\max} il comportamento della macchina non è definito e risulta essere aleatorio. Tuttavia, pur rispettando questa ipotesi è possibile comunque il verificarsi di possibili

fenomeni aleatori. Una classificazione delle alee prevede in un primo livello di generalizzazione le cosiddette alee transitorie ed alee di regime.

Un alea è di tipo transitorio se le uscite della rete assumono valori diversi da quelli progettati soltanto nel transitorio conseguente alle variazioni degli ingressi.

Un alea di regime si verifica quando l'uscita assume un valore diverso da quello progettato anche dopo il transitorio.

Un alea transitoria è preferibile ad una di regime poiché se attendiamo un tempo sufficientemente grande siamo sicuri che la rete fornisca in uscita il giusto risultato, mentre il risultato di un alea di regime non è, per definizione, mai quello della rete di progettazione. Tipicamente le alee transitorie vengono anche dette alee combinatorie poiché riguardano esclusivamente le macchine di tipo combinatorio. Invece, le alee di regime sono anche dette alee sequenziale poiché investono principalmente le macchine di tipo sequenziale.

Alee tipiche	
Alee transitorie (Macc. Comb.) <ul style="list-style-type: none"> • Alea multipla variazione simultanea di due o più variabili di ingresso • Alea per impulsi concom. presenza di due o più impulsi • Alea statica variazione temporanea dell'uscita che dovrebbe rimanere costante • Alea dinamica oscillazione temporanea dell'uscita 	Alee di regime (Macc. Seq.) <ul style="list-style-type: none"> • Alee essenziali Dovute alle caratteristiche della rete • Alee per corse critiche A causa della codifica le variazioni delle uscite dipendono dall'ordine degli ingressi • Alee per frequenza elevata Gli ingressi variano troppo rapidamente

Alea multipla

Per introdurre il concetto di alea multipla è utile prima dare spiegazioni sull'ingresso di dati di tipo adiacenti. Due stati di ingresso si dicono adiacenti se codificati in modo che essi differiscano solo in una sola variabile. Ad esempio:

```

      |
      | i1:000010
      |           Valori adiacenti
t  ↓  i2:000000

```

infatti i due ingressi differiscono per una sola variabile che è appunto quella segnata in grassetto. Mentre:

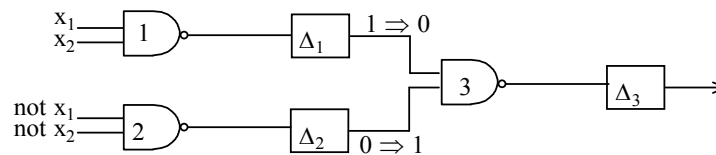
```

      |
      | i1:000010
      |           Valori non adiacenti
t  ↓  i2:010000

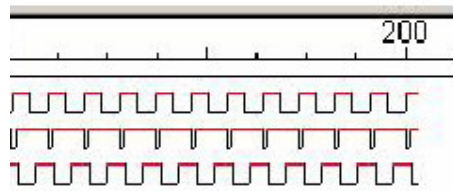
```

Infatti i due ingressi presi come esempio differiscono per due variabili. Si ha un'alea multipla se due ingressi consecutivi nel tempo, i_1 e i_2 , non sono tra di loro adiacenti. Ad esempio, consideriamo la rete che realizza questa funzione:

$$y = x_1 x_2 + \bar{x}_1 \bar{x}_2$$



Avete il seguente diagramma temporale:



Osservando la funzione logica, si nota che il valore in uscita della macchina combinatoria dovrebbe essere fissa ad 1, tuttavia già nel diagramma temporale notiamo una leggera oscillazione che ne modifica, se pur in brevi istanti di tempo, il giusto valore.

Ciò è dovuto ad una parte della macchina che risulta essere più lenta rispetto all'altra per la presenza non solo della porta logica AND ma anche di una NOT, ogni porta logica è infatti caratterizzata da un tempo di risposta o ritardo che nel caso analizzato è la somma di due ritardi.

Se tutti e due gli ingressi alla porta n°3 assumono il valore 1 per un tempo maggiore del suo ritardo inerziale, in corrispondenza l'uscita y assumerà valore 0.

Il problema sembra essere di natura tecnologico, la realizzazione di porte logiche più rapide in termini di risposta agli ingressi potrebbe quindi ridurre gli istanti di tempo in cui l'uscita y anziché rimanere fissa ad 1 oscilla per brevi istanti di tempo.

Tuttavia, anziché scaricare il problema sul processo tecnologico, investendo tempo e denaro nella costruzione di dispositivi logici ad alta efficienza è possibile correggere il problema agendo sulla codifica degli ingressi e facendo in modo che questi risultino tra loro sempre adiacenti se questi devono contribuire all'uscita della macchina. Quindi, per eliminare le alee multiple è opportuno eliminare la possibilità di ingressi non adiacenti agendo quindi sulla codifica dell'input.

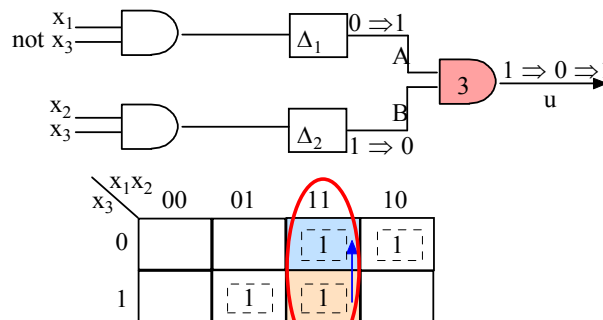
Alee per impulsi concomitanti

L'alea ad impulsi concomitanti è un caso particolare di alea multipla, essa si manifesta quando due impulsi avvengono contemporaneamente, in tal caso i ritardi possono cambiare il comportamento della funzione. Gli impulsi si hanno per variabili d'ingresso che cambiano in tempi molto rapidi. Supponiamo ad esempio il caso in cui una rete realizzi la funzione logica di AND. All'ingresso delle variabili $x_1 = 1$ ed $x_2 = 1$ la funzione logica $y = x_1 \text{ and } x_2$ riceve contemporaneamente due valori alti, ma per un certo intervallo di tempo essa non presenta il giusto risultato. Per fare in modo che ciò non avvenga si è soliti mettere un dispositivo di inerzia su una delle variabili d'ingresso in maniera tale da provocarne un rallentamento. Poiché tutte le funzioni sono derivate dalle funzioni somma e prodotto, al fine di evitare i fenomeni aleatotici è necessario che due impulsi in ingresso alla medesima macchina non siano mai in concomitanza se insieme debbono contribuire a determinare l'uscita.

Alea statica

Questo tipo di alea avviene se avendo due ingressi i_1 ed i_2 adiacenti con uscite uguali $f(i_1)=f(i_2)$ l'uscita assume nel transitorio il valore $\underline{f(i_1)=f(i_2)}$, ad esempio:

$$y^* = x_0 \bar{x}_2 + x_1 x_2$$



Implicante ridondante da aggiungere

Gli ingressi pur essendo tra loro adiacenti causano un oscillazione temporanea dell'uscita, ciò risulta essere causato dalla minimizzazione che segue dalla mappa di Karnough e che tende ad escludere nella funzione logica un primo implicante perché già contenuto in due implicanti essenziali.

Infatti sollecitando la macchina con gli ingressi $x_0x_1x_2=111$ e successivamente con $x_0x_1x_2=110$ (e/o viceversa) notiamo che:

Per $x_0x_1x_2=111$, $\Delta_1=0$ e $\Delta_2=1$;

Per $x_0x_1x_2=110$, $\Delta_1=1$ e $\Delta_2=0$;

L'uscita y quindi, per fornire il giusto valore in uscita deve necessariamente attendere un certo istante di tempo in quanto l'oscillazione di $\Delta_1=0 \rightarrow 1$ e $\Delta_2=1 \rightarrow 0$ avviene simultaneamente, in questo istante di tempo si vengono a trovare entrambe pari a 0.

Per eliminare le alee statiche da una rete è necessario e sufficiente aggiungere alla funzione logica che la realizza gli eventuali implicanti ridondanti che contengono le transizioni fra gli ingressi adiacenti per i quali l'uscita assume il valore logico 1, infatti in questo modo la funzione logica comprenderà un altro addendo:

$$y^{**} = x_0x_1 + x_0\bar{x}_2 + x_1x_2$$

Che non risente di alcuna variazione quando poniamo in ingresso $x_0x_1x_2=111$ e $x_0x_1x_2=110$.

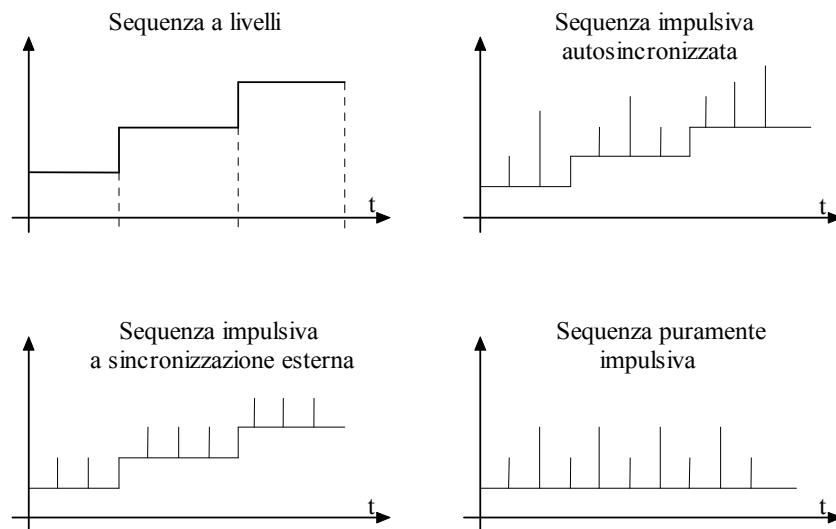
Alea dinamica

Si ha una alea dinamica se avendo due ingressi i_1 ed i_2 adiacenti con uscite $f(i_1)=\alpha$ $f(i_2)=\beta$, la sequenza di uscita è del tipo $\alpha...\beta\alpha\beta...$. L'alea dinamica si verifica soltanto in reti a più di due livelli e può essere vista come il fenomeno risultante da una o più alee statiche contenute nelle sottoreti a due livelli. Le alee dinamiche si eliminano eliminando le alee statiche nelle sottoreti.

Avendo considerato il tempo come fattore determinante, non è più possibile considerare l'ingresso semplicemente come un singolo fattore fornito in ingresso alla macchina.

Inizieremo allora a pensare alle macchine e a come gli ingressi si presentano con il passare del tempo, per fare ciò è opportuno rivedere quali tipologie di ingressi sono possibili e considerare queste come sequenze variabili nel tempo.

Gli ingressi delle macchine sono delle sequenze di valori e di sequenze ne esistono di diverse tipologie, infatti:



Gli ingressi delle macchine da noi progettate fin ora sono delle sequenze di valori in ingresso che evolvono nel tempo. I problemi che gli ingressi introducono alla progettazione della macchine sono stati appena descritti (le alee). I diversi tipi di sequenze che una macchina può ricevere in ingresso sono descritti graficamente in questa pagina.

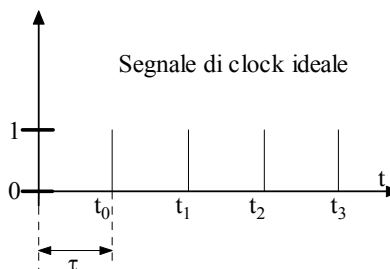
La sequenza a livelli è simile alla funzione a gradino, l'ingresso della macchina rimane costante per un determinato periodo di tempo, in ogni istante di tempo c'è sempre un valore logico in ingresso.

La sequenza puramente impulsiva prevede un ingresso impulsivo caratterizzato da un valore alto raggiunto in una frazione di tempo molto piccolo, questa sequenza di ingresso può essere usata come sincronizzatore, il riferimento in questo caso è ad uno spazio discreto di istanti di tempo in cui agisce appunto l'impulso.

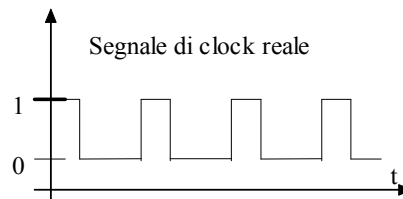
La sequenza a sincronizzazione esterna, ottenuta miscelando una sequenza a livelli e una impulsiva consente ad una macchina di considerare i valori in ingresso solo quando l'impulso indica a quest'ultima di leggerne il valore, si nota infatti che l'impulso è unico su tutta la sequenza a livelli.

Nel caso di una sequenza autosincronizzata è possibile avere su una sequenza a livelli più di un impulso, ciascuno dei quali avrà uno specifico significato per la macchina.

Il clock è una particolare sequenza utile in fase di simulazione e largamente usata dalle macchine di tipo sequenziale che nella prossima lezione vedremo, esso in particolare scandisce gli ingressi secondo una sequenza puramente impulsiva in cui gli impulsi hanno tutti la medesima altezza, gli impulsi si ripetono inoltre con la stessa frequenza, risulteranno pertanto spaziatosi sempre alla stessa distanza. Il clock è pertanto una particolare variabile binaria impulsiva che identifica, in uno spazio discreto dei tempi, una particolare sequenza. Un diagramma temporale di un segnale di clock è il seguente:

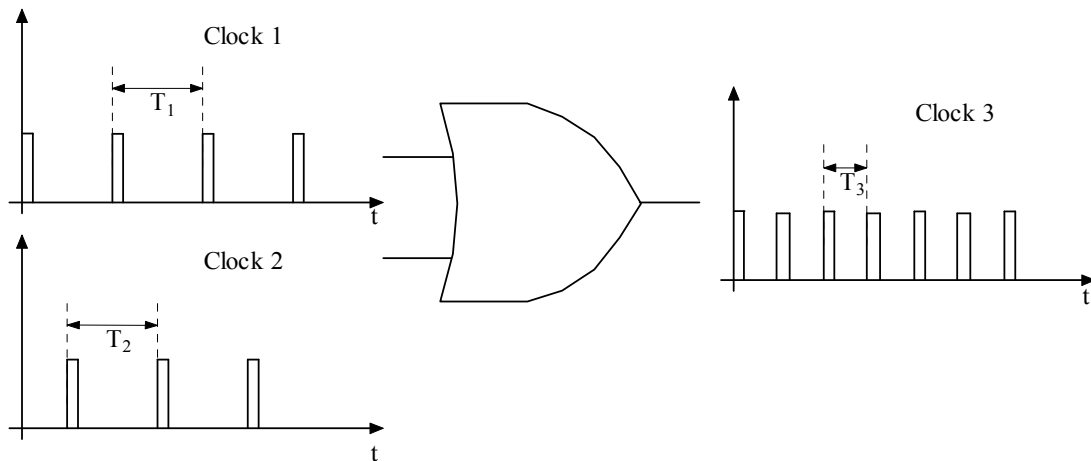


Tuttavia in realtà il clock è realizzato con un oscillatore, esso continua a mantenere i diversi livelli logici ma questi ultimi risultano essere di durata Δ non nulla, pertanto un diagramma temporale di un segnale di clock reale è il seguente:

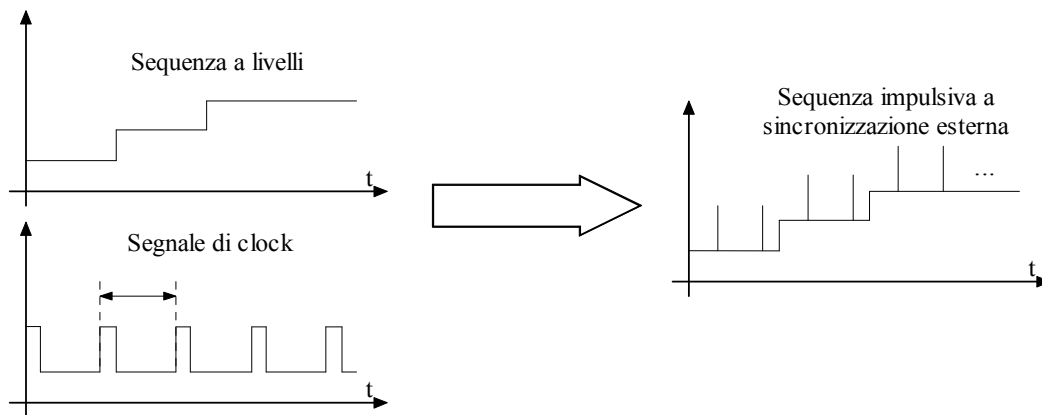


Nel simulatore che si userà (Logic Works) è possibile impostare per ogni dispositivo logico ed anche per il clock un tempo di ritardo.

Molto spesso è utile avere per una macchina diversi valori di clock (almeno più di uno) sfasati tra loro che insieme definiscono una sequenza d'ingresso. E' possibile ottenere clock ad n fasi, miscelando n clock e ritardandoli tra loro, ad esempio:



E' inoltre possibile realizzare una sequenza a sincronizzazione esterna miscelando un segnale di clock con una sequenza a livello:



Lezione N°6: "Macchine e reti sequenziali"

Una macchina sequenziale a stati finiti è quella particolare macchina in cui le uscite dipendono non solo dagli ingressi che gli vengono mossi ma anche dallo stato in cui attualmente si trova la macchina. Pertanto, lo stesso ingresso potrebbe provocare uscite differenti se la macchina si trova in uno stato diverso dal precedente. (Per le macchine combinatorie invece lo stesso ingresso produce sempre la stessa uscita, come ad esempio fa una calcolatrice)

Ogni ingresso alla macchina genera un cambiamento di stato della stessa oppure consente alla macchina di rimanere nello stato attuale.

Proprio per questo motivo si è soliti dire che le macchine sequenziali, a differenza di quelle combinatorie, sono dotate di memoria, dovendo infatti memorizzare lo stato in cui esse si trovano. E' possibile dare due definizioni matematiche di macchine sequenziali, esse sono dovute a Moore ed a Mealy, abbiamo pertanto un modello di Moore di macchina sequenziale ed un modello di Mealy.

Secondo il **modello di Mealy** la macchina sequenziale (a stati finiti) è una quintupla ordinata di enti $M(Q, I, U, t, \omega)$ dove:

- Q è l'insieme di stati finiti interni alla macchina, ogni singolo stato è denotato con la lettera q_i , il pedice lo contraddistingue da altri stati (ad esempio $q_1, q_2, q_3, \dots, q_k$);
- I è l'insieme, anch'esso finito, di stati di ingresso per la macchina;
- U è l'insieme, ancora finito, di tutte le possibili uscite della macchina;
- t è una funzione logica tale che per $D \in Q \times I \rightarrow Q$, è cioè quella funzione che in base all'ingresso ricevuto e allo stato attuale della macchina genera un nuovo stato in cui la macchina si porterà se soggetta a quell'ingresso;
- ω è la funzione tale che per $D \in Q \times I \rightarrow U$, è cioè quella funzione che in base all'ingresso ricevuto ed allo stato attuale della macchina genera l'uscita logica della macchina sequenziale.

Il **modello di Moore** fornisce una descrizione di macchina sequenziale leggermente differente modificando la funzione logica ω che risulta essere così definita:

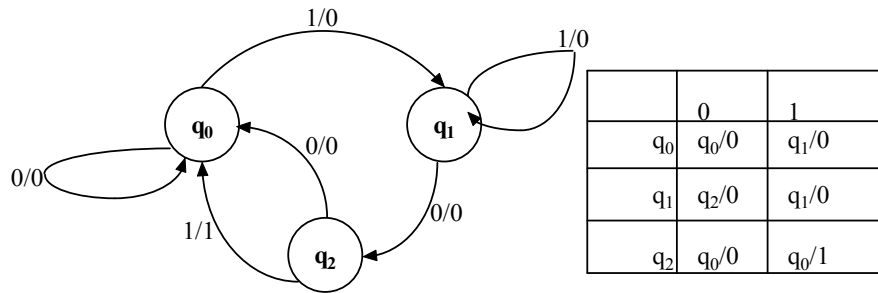
- ω è la funzione logica tale che per $D \in Q \rightarrow U$, è cioè la funzione che genera l'uscita considerando il solo stato interno della macchina;

Data una macchina di Mealy è possibile realizzarne una identica secondo Moore e viceversa. E' bene precisare che gli automi che ci apprestiamo a realizzare non consentono la costruzione di tutte le possibili macchine, esse infatti risulteranno limitate negli stati.

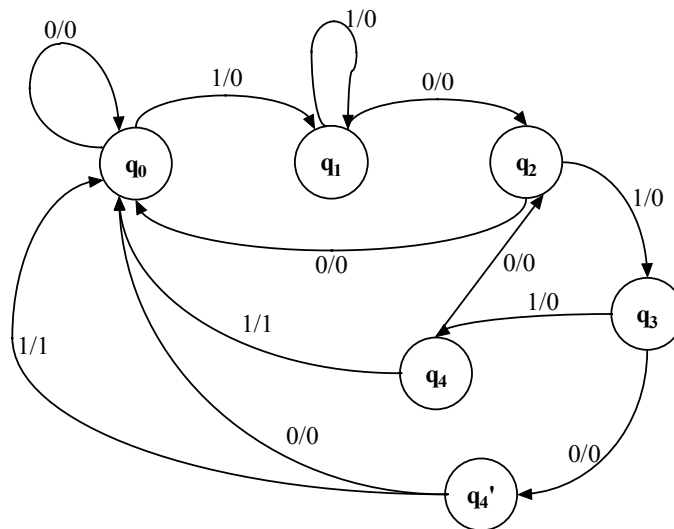
La descrizione della macchina sequenziale oltre che dalle tabelle delle funzioni t ed ω può avvenire anche secondo un grafo detto diagramma degli stati. Le tabelle di t ed ω generano il prossimo

stato della macchina ottenuto incrociando i diversi valori dello stato attuale e del possibile ingresso disposti sulle righe e sulle colonne della tabella.

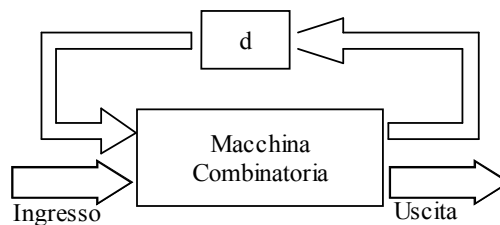
Il grafo della macchina assume un diverso significato a seconda che si tratti del modello di Moore oppure del modello di Mealy. Secondo Mealy l'uscita è associata all'arco che collega due stati mentre secondo Moore l'uscita è associata al nodo o allo stato in cui si trova la macchina. Vediamo alcuni esempi, si vuole realizzare un automa che riconosca la sequenza binaria 101, tracciamo quindi il suo grafo e la sua tabella di transizione per gli stati:



Ancora un esempio: tracciare il grafo di una rete sequenziale in grado di riconoscere la sequenza binaria 101-1:



Una macchina sequenziale può essere realizzata con una macchina combinatoria e un elemento di ritardo collegati nel seguente modo:



L'elemento di ritardo è essenziale poiché il comportamento temporale della macchina suggerisce per la macchina sequenziale un funzionamento secondo uno spazio temporale in base alla quale vengono considerate le funzioni ingresso $i(t)$ e di uscita $u(t)$. Il modello di macchina combinatoria per queste funzioni che abbiamo visto nelle precedenti lezioni realizza le funzioni $i(t)$ ed $u(t)$ e siccome entrambe determinano il nuovo stato per la macchina sequenziale a partire dallo stato attuale esse necessitano, per il corretto funzionamento, di un elemento di ritardo che ha quindi il compito di riproporre in ingresso sia ad $i(t)$ che ad $u(t)$ lo stato attuale della macchina. Uno stato q di un automa si dice stabile all'ingresso i se si verifica che:

$$t(q, i) = q$$

Ovvero, uno stato è stabile rispetto ad un certo ingresso se l'automa in tali stati, sottoposto per un certo tempo all'ingresso indicato, permane nel suo stato indefinitamente.

Gli ingressi in una macchina sequenziale possono essere a livello (in tal caso il riferimento è allo spazio continuo dei tempi) oppure ad impulsi (in tal caso le sequenze di ingresso significative definiscono uno spazio discreto dei tempi); come vedremo più avanti ciò richiede, per il corretto funzionamento della macchina, particolari caratteristiche delle funzioni t ed ω e quindi delle tabelle particolari dette di transizione.

Come fatto per le macchine combinatorie, la minimizzazione in una macchina sequenziale, cioè la ricerca di una macchina M' che contenga o includa una macchina M dotata di un numero minore di stati, assume per le macchine sequenziali la medesima importanza che per quelle combinatorie assume la minimizzazione delle funzioni booleane.

Importanti definizioni utili alla classificazioni di macchine sequenziali sono le definizioni di macchina asincrona e sincrona. Una macchina si dice **asincrona** se per ciascun ingresso i e per ciascuno stato q_i , la sequenza:

$$t(q_{i1}, i) = q_{i2} \quad , \quad t(q_{i2}, i) = q_{i3}$$

termina in uno stato stabile:

$$t(q_{in}, i) = q_{in}$$

Una macchina asincrona è identificabile già dal suo grafo poiché quest'ultima è dotata, per ogni suo stato, di auto-anelli i quali sono significativi del fatto che gli stati della macchina sono stabili nei confronti di certi ingressi. Una macchina si dice invece **sincrona** se non è asincrona.

Una macchina con una sequenza di ingressi a livelli funziona solo se è asincrona, infatti essendo la sequenza di ingressi a livelli quest'ultima fornisce ad ogni istante di tempo un valore significativo per la macchina che sarà assunto come ingresso, pertanto, se la macchina è asincrona (dotata cioè di ogni stato

stabile) l'ingresso ad essa proposto provoca diverse transizioni fino a giungere in uno stato stabile dopo un periodo di evoluzione della macchina noto come periodo transitorio.

Invece, per una macchina sincrona esistono degli stati non stabili rispetto a certi ingressi e quindi una qualsiasi sequenza a livelli potrebbe generare delle transizioni infinite fra gli stati poiché i valori della sequenza a livello potrebbero non terminare in uno stato stabile e proseguire all'infinito. Nel progetto di macchine sincrone l'automa va costruito con attenzione, in modo tale da arrivare sempre ad uno stato stabile qualora alla macchina venga proposto un determinato ingresso.

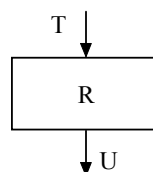
Una macchina con ingresso a livelli è una macchina sequenziale di tipo asincrono. Per una macchina sincrona gli ingressi invece non possono essere a livelli poiché l'ingresso dovrebbe avere una durata esattamente quanto il tempo di ritardo della macchina, condizione irrealizzabile a causa dei ritardi inerziali e parassiti dei circuiti elettronici, l'ingresso può allora essere di tipo impulsivo e di durata sufficiente a cambiare lo stato della macchina.

Nella definizione di macchina sequenziale è emerso il problema di come memorizzare lo stato che la macchina occupa all'interno del suo grafo, a tale proposito definiamo allora i registri.

Il **registro** è una macchina sequenziale con il compito di memorizzare un valore di ingresso per un tempo sufficientemente lungo, finché non venga esplicitamente indicato che un valore diverso deve sostituirlo, esso si compone di:

- K valori di uscita (valori memorizzati);
- K ingressi attivi (valori da memorizzare);
- M ingressi neutri (non cambiano il valore memorizzato);

Il registro è una macchina sequenziale poiché la sua uscita è in generale funzione dello stato precedente, oltre ad essere un elemento di memoria è fondamentale nella realizzazione di macchine sequenziali ed in particolare di quelle sincrone, un registro viene tipicamente rappresentato in questo modo:



T sono gli ingressi attivi che determinano l'uscita U. Un particolare registro è il flip-flop capace di memorizzare un dato binario (e quindi i valori logici 1 oppure 0). Essendo il flip-flop adatto a memorizzare un bit b, la sua uscita ha solo due possibili stati stabili che sono detti rispettivamente:

- 1) Stato di SET, o alto oppure stato "1", corrispondente a b=1;

- 2) Stato di RESET, o basso oppure stato "0", corrispondente a $b=0$;

Affinché sia possibile fornire il dato all'esterno ed il suo complemento, utile in diverse applicazioni, gli stati sono codificati usando variabili di tipo booleane:

$b=1 \quad D=1 \quad \text{not } F=0$
 $b=0 \quad F=0 \quad \text{not } F=1$

In base a quanto detto per i registri anche il flip-flop, che è un particolare registro, deve possedere distinti stati di ingresso ed in particolare almeno uno deve essere neutro poiché questo sarà destinato al mantenimento di un bit, altri stati attivi consentono poi al flip-flop di posizionarsi in uno dei due stati. Gli stati attivi sono:

- 1) ingresso di SET per la memorizzazione di $b=1$ ($F=1$ ed $\text{not } F=0$);
- 2) ingresso di RESET per la memorizzazione di $b=0$ ($F=0$ ed $\text{not } F=1$);

Oltre a questi due stati alcuni flip-flop sono dotati di un ingresso particolare quale è l'ingresso di commutazione: se in ingresso è impostato questo stato il flip-flop commuta lo stato di uscita su $b=1$ se prima esso valeva $b=0$.

I flip-flop possono essere macchine asincrone o sincrone, ad ingressi impulsivi oppure non, ed inoltre diversi flip-flop si ottengono in funzione delle diverse variabili binarie usate di ingresso. Esistono due tipi di flip-flop a memorizzazione: il flip-flop di tipo RS i cui ingressi sono i segnali R ed S ed il flip-flop di tipo D. Ecco una classificazione dei flip-flop:

Asincroni	Funzionalità
Sincroni con tempificazioni:	- RS
- Latch	- D
- Edge Trigger	- JK
- Master/Slave	- T

La tempificazione caratterizza il modo in cui il segnale di uscita cambia al variare del segnale di ingresso mentre le funzionalità distinguono i flip-flop in funzione del tipo di ingresso che accetta.

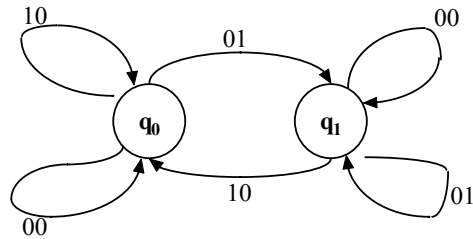
Flip-flop RS fondamentale

Il flip-flop fondamentale è una macchina asincrona (come si può notare dagli auto-anelli presenti nel suo grafo) con ingresso di tipo impulsivo, essa è caratterizzata da:

- Due ingressi R ed S;
- Una uscita (corrispondente al bit memorizzato);

entrambi i bit R ed S non possono mai essere 1, deve essere cioè mantenuto il vincolo $RS=0$. Il segnale R alto, cioè $R=1$, significa che il flip-flop memorizza il bit 0 (condizione di RESET); il

segnale S alto, cioè $S=1$, significa che il flip-flop memorizza il bit 1 (condizione di SET), il grafo del flip-flop RS fondamentale è:



La tabella di transizione che descrive l'evoluzione degli stati per la macchina è la seguente:

STATO\RS	00	01	11	10
q_0	q_0	q_0	-	q_0
q_1	q_1	q_1	-	q_0

Per la codifica degli stati basta un solo bit, infatti:

- Stato 0 $S_0=0$;
- Stato 1 $S_1=1$;

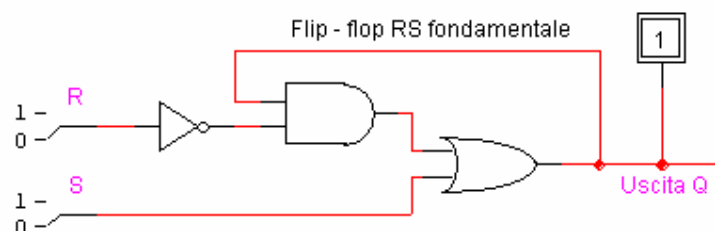
La codifica delle uscite è associata allo stato, gli ingressi sono già codificati per cui la mappa di Karnaugh è:

F\RS	00	01	11	10
0	0	1	-	0
0	1	1	-	0

La funzione logica che si ottiene, detta equazione di stato del flip-flop, è la seguente:

$$F' = S + F\bar{R}$$

Come già visto per le macchine combinatorie, ad ogni funzione logica è possibile associare un circuito logico equivalente alla funzione logica, per il flip-flop RS tale circuito è mostrato nella pagina che segue.



Lezione N°7: "Il flip-flop di tipo D (Latch)"

Il registro è una macchina sequenziale poiché la sua uscita è anche funzione dello stato precedente e coincide con il valore già contenuto nel registro se essa non deve cambiare. Oltre alle sue capacità di memorizzare un dato, il registro ci consente di realizzare le macchine sincrone.

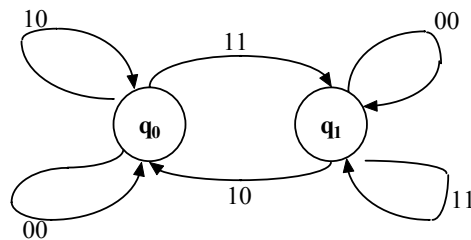
Il registro e/o flip-flop latch è un modello valido per ingressi impulsivi. Il flip-flop di tipo D latch detto anche flip-flop abilitato è ottenuto da quello RS fondamentale con l'aggiunta di una variabile di abilitazione A, analizziamone il progetto.

Il flip-flop D

E' una macchina con:

- 2 ingressi, a e D;
- 1 uscita;

Quando il valore del bit a, detto bit di abilitazione, è alto cioè $a=1$ allora il flip-flop memorizza i valori dell'ingresso al bit D, l'uscita è uguale al valore memorizzato. Il grafo degli stati è:



Il flip-flop D latch è una macchina asincrona perché dato un valore in ingresso la macchina si blocca sicuramente in uno dei suoi stati (osservare il grafo). La tabella di transizione per gli stati è la seguente:

stato\ aD	00	01	11	10
S ₀	S ₀	S ₀	S ₁	S ₀
S ₁	S ₁	S ₁	S ₁	S ₀

Per codificare gli stati basta un solo bit (F), poiché si deve codificare S₀ ed S₁, pertanto:

- S₀=0;
- S₁=1;

Gli ingressi al flip-flop sono già codificati trattandosi infatti di bit. Per la codifica delle uscite si decide di utilizzare lo stato del flip-flop.

La tabella della verità che si ottiene codificando gli stati del flip-flop è:

F	a	D	F'
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Dalla quale si ricava questa mappa di Karnough:

F\ aD	00	01	11	10
0	0	0	1	0
1	1	1	1	0

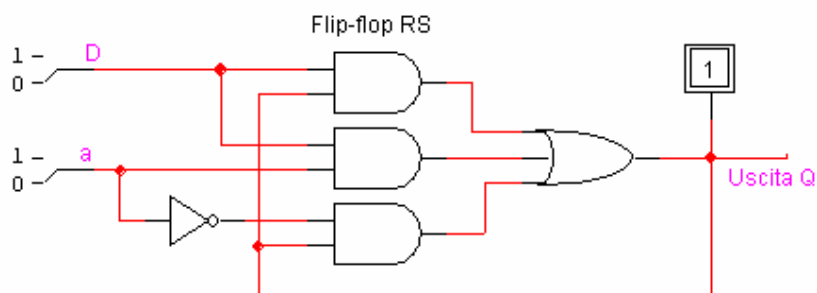
La variazione di due ingressi non adiacenti avente tra l'altro lo stesso valore nella funzione di uscita genera un alea statica, affinché si possa risolvere il problema decidiamo di introdurre nella funzione logica che scriveremo tutti gli implicant ridondanti. (ciò equivale a considerare l'implicante ridondante che in figura è tratteggiato in blu). La funzione logica:

$$F' = \underline{a}F + aD$$

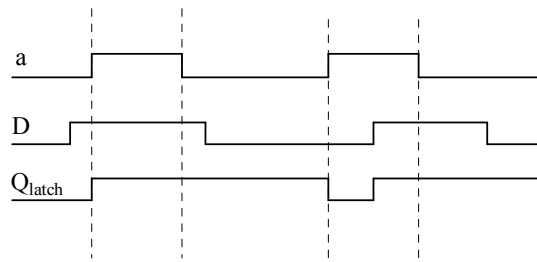
è viziata da alea statica, mentre la funzione logica:

$$F' = \underline{a}F + aD + DF$$

non è viziata da alea statica poiché abbiamo aggiunto l'implicante ridondante escluso dalla minimizzazione, il circuito logico è:

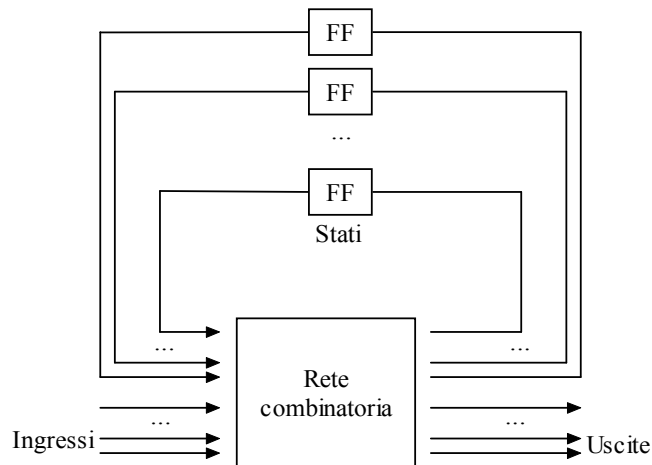


Nel disegno delle tempificazioni si può osservare come i dispositivi di tipo D latch ripropongano l'uscita fintanto che il bit di abilitazione è alto:

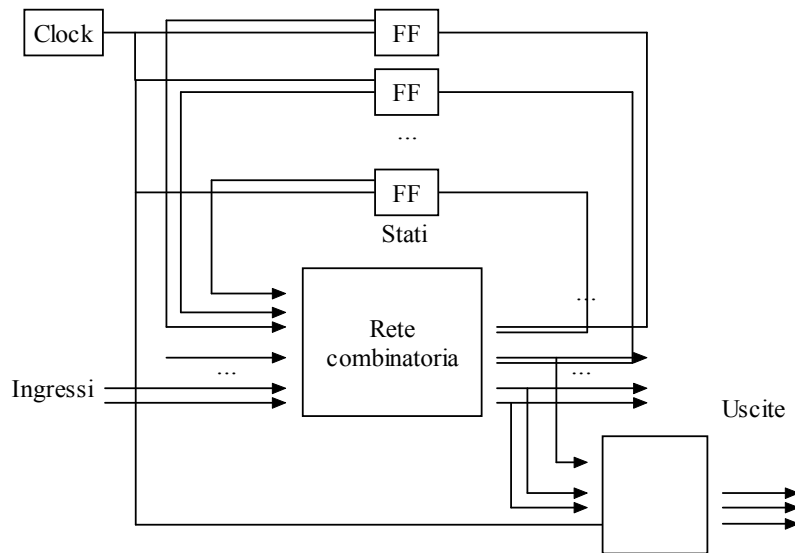


Analizziamo ora le relazioni sequenze di ingresso/tipo di macchina sequenziale che possiamo realizzare.

Per una rete autosincronizzata si hanno ingressi impulsivi che possono contenere anche più di un valore (ad esempio x_0 vale 1, x_1 vale 1, etc...) Volendo usare un flip-flop per una rete di questo tipo osservo che non ho un segnale che mi indica quando memorizzare un dato, a questo tipo di rete si associa l'utilizzo del flip-flop RS fondamentale che è esso stesso una macchina asincrona. Uno schema tipo per reti autosincronizzate è:



In una rete a sincronizzazione esterna (ad esempio un clock scandisce a precisi istanti di tempo i valori di ingresso) gli ingressi sono a livello, pertanto c'è un segnale che sincronizza la macchina e che abilita i flip-flop alla lettura del dato. In questo caso si adotta allora il flip-flop latch, si deve però fare attenzione alla durata del segnale che deve essere tale da permettere al flip-flop di percepire l'abilitazione vincendo l'inerzia iniziale della macchina. Un schema tipico per reti a sincronizzazione esterna è:



L'analisi di progetto per una macchina sequenziale si compone delle seguenti fasi:

- Analisi del problema e definizione della macchina;
- Scelta della macchina;
- Scelta del flip-flop;
- Progetto della rete;
- Realizzazione del dispositivo;

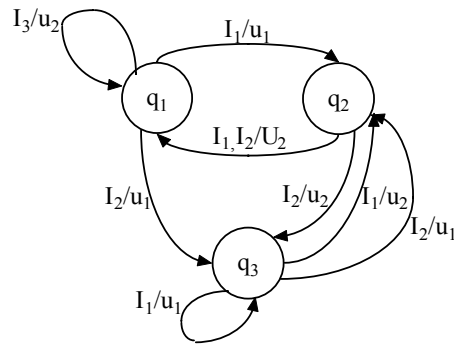
La definizione della macchina deve tener conto dell'insieme dei valori in input e output, indagare su eventuali segnali di abilitazione alle macchine, è opportuno specificare in questa fase i vincoli sul tempo e gli effetti che questi producono sulla reazione della macchina.

La scelta della macchina consiste nell'individuare una collocazione per la macchina sia essa asincrona o sincrona. Questo tipo di scelta è dettato dal flusso di dati in ingresso alla macchina che possono essere a livelli, a sincronizzazione esterna o autosincronizzata. Etc..

Noto il tipo di macchina si procede individuando la migliore tempificazione per il flip flop da scegliere, in alcuni progetti occorre realizzare particolari operazioni quali ad esempio il reset per la macchina, in tal caso è opportuno tenere presente che alcuni flip flop sono dotati di un bit di reset che sarà sicuramente utile.

Noti gli ingressi (stato + ingressi della macchina sequenziale) e note le uscite (stato + uscite della macchina sequenziale) è allora definito il rapporto I/O dell'automa, è possibile quindi ricavare la rete combinatoria che realizza lo spostamento tra i vari stati mentre i flip flop memorizzano lo stato attuale.

Osservando il seguente grafo:



è possibile notare alcune cose: la macchina che ha generato questo grafo è di tipo sincrono; Gli stati q_1 e q_3 pur essendo stabili rispetto agli ingressi I_3 (per q_1) ed I_1 (per q_3) non sono sufficienti a garantire l'asincronicità della macchina poiché lo stato q_2 è instabile a qualunque ingresso, ciò basta per stabilire che la macchina è sincrona.

Per una macchina sincrona gli ingressi devono essere di tipo impulsivo, un segnale dirà alla macchina quando leggere l'ingresso. Macchine sincrone con ingressi a livelli non esistono.

Sequenze di ingressi impulsivi

La sequenza in ingresso a una macchina sequenziale si dice essere impulsiva se è possibile suddividere i valori in ingresso secondo:

- valori impulsivi (i_1, i_2, \dots, i_k);
- basi dei valori impulsivi ($i_{01}, i_{02}, \dots, i_{0k}$);

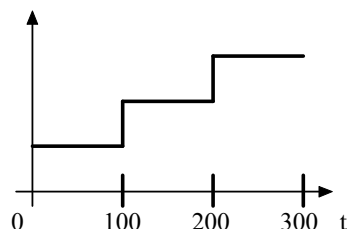
e per ciascuno ingresso a livello (basi dei valori impulsivi) gli stati della macchina sono tutti stabili. Per le macchine sincrone l'impulso deve avere una durata sufficiente a garantire l'acquisizione del nuovo stato ma non deve assolutamente durare più a lungo poiché gli ingressi possono transitare nei diversi stati della macchina alterandone il funzionamento.

Per le macchine asincrone gli ingressi impulsivi permettono di aggiungere alla macchina altre proprietà visto che di per se la macchina funziona anche senza questo tipo di ingresso.

Il modello degli ingressi indipendenti costituisce l'unico modo concreto di concepire il funzionamento delle macchine sincrone.

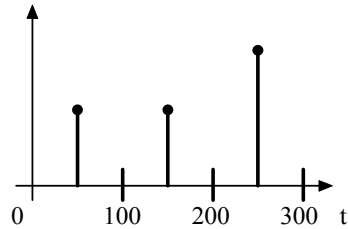
Sequenze a livelli

Gli impulsi alla macchina sequenziale si dicono essere a livelli, cioè distribuiti su di uno spazio continuo del tempo, se in ogni istante di tempo esiste un valore significativo per la macchina:



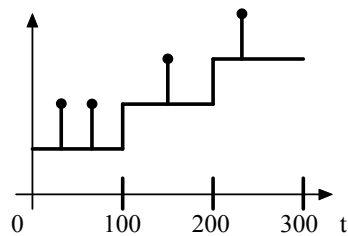
Sequenze impulsive

Per un flusso di dati in ingresso ad una macchina sequenziale, le sequenze impulsive hanno come base una sequenza di livello:



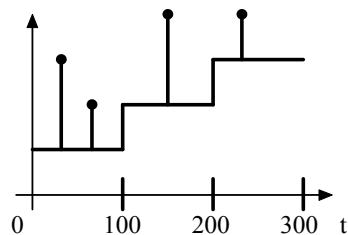
A sincronizzazione esterna

Su ogni base della sequenza a livello c'è sempre un unico valore impulsivo che si ripete nel tempo, tale sequenza è detta a sincronizzazione esterna poiché è una variabile esterna a comandare l'impulso e a scandirne i tempi:



Sequenze impulsive autosincronizzata

Sulla base della sequenza a livelli insistono diversi valori impulsivi:



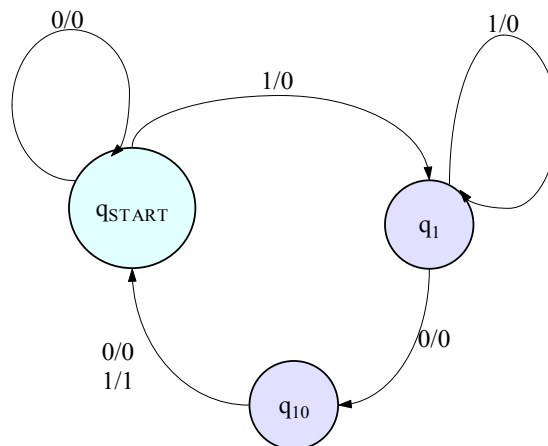
Esempi di progettazione di macchine sequenziali:

Riconoscitore di sequenza 101 (Ver. senza sovrapposizioni)

Progettare una macchina sequenziale a sincronizzazione esterna capace di riconoscere la sequenza di bit 101, non considerare le diverse sovrapposizioni tra i bit in ingresso alla macchina.

L'insieme di ingresso e quello di uscita sono già definiti poiché forniti dalle specifiche di progetto, inoltre per tali insiemi non è necessario effettuare nessuna codifica poiché la macchina da realizzare accetta in ingresso sequenze di bit e fornisce in uscita un bit, alto se viene riconosciuta la sequenza 101.

Procediamo nella costruzione dell'automa:



Le etichette associate a ciascuno stato sono significative dei bit riconosciuti fino a quell'ingresso. La macchina da realizzare è di tipo sincrono a causa di uno stato instabile quale è q_{10} . Questo automa genera dunque questa tabella di transizione:

Stati	0	1
q_{START}	$q_{START}/0$	$q_1/0$
q_1	$q_{10}/0$	$q_1/0$
q_{10}	$q_{START}/0$	$q_{START}/1$

L'automa si compone inoltre di 3 stati, q_{START} , q_1 e q_{10} i quali necessitano per una codifica di 2 bit. Infatti, con due bit è possibile rappresentare 4 stati in forma codificata e nel nostro caso una rappresentazione sarà inutilizzata, queste le codifiche scelte:

$q_{START} = 00$
 $q_1 = 01$
 $q_{10} = 10$

La scelta del flip-flop da utilizzare è caduta sul flip-flop D latch, la tabella di transizione degli stati è quindi codificata rispettando le funzionalità di tale flip-flop:

Stati		Ingresso	Nuovo stato		Uscita
A	B		A'	B'	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	-	-	-
1	1	1	-	-	-

Mappe di Karnough

Funzione logica A' :

AB		00	01	11	10
I					
0			1	-	
1				-	

$$A' = B\bar{I}$$

Funzione logica B' :

AB		00	01	11	10
I					
0				-	
1		1	1	-	

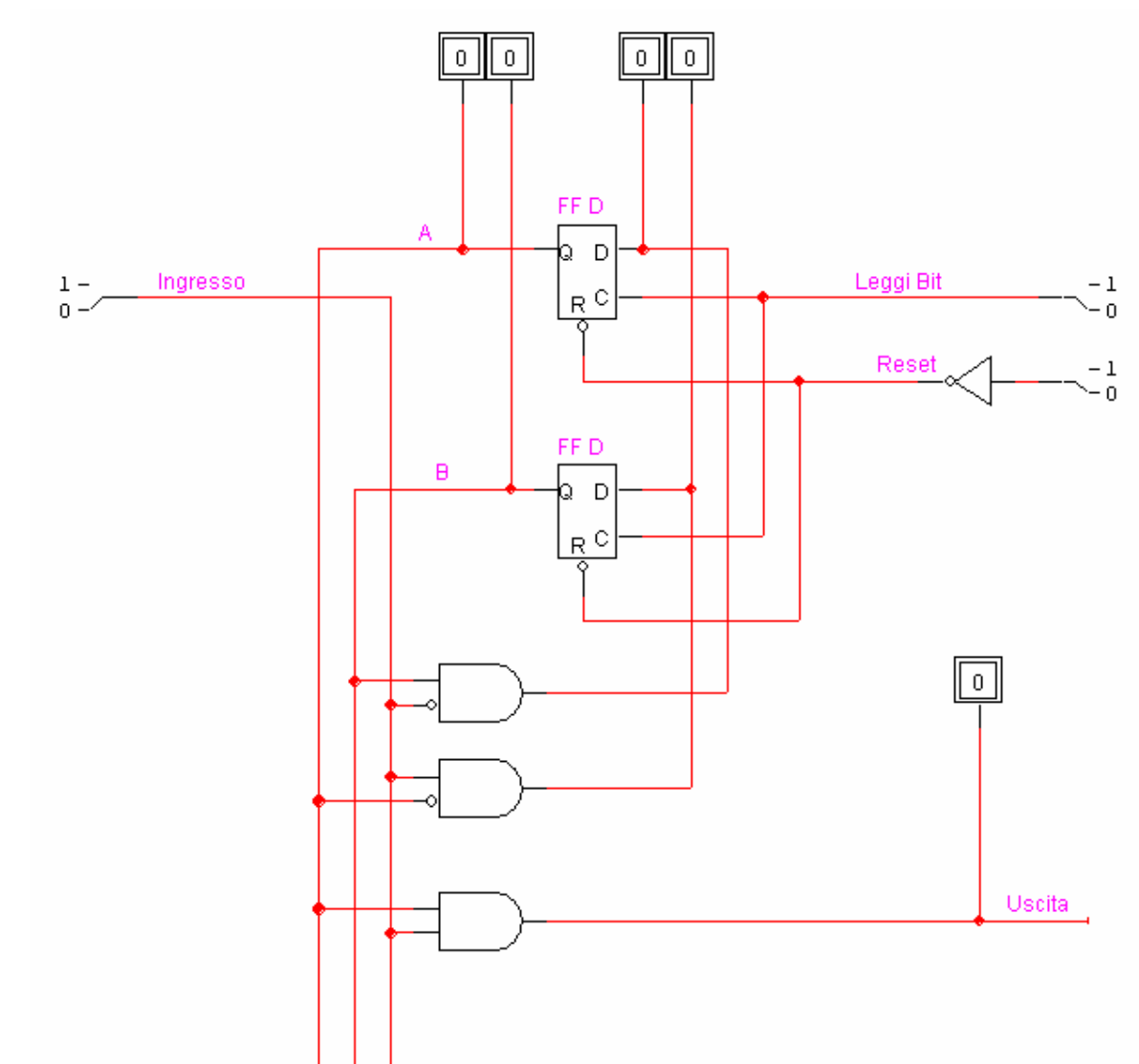
$$B' = \bar{A}I + BI$$

Funzione logica U:
(Uscita)

AB		00	01	11	10
I					
0				-	
1				-	1

$$U = AI$$

Non rimane che tracciare il circuito della macchina realizzata e qui riportato:



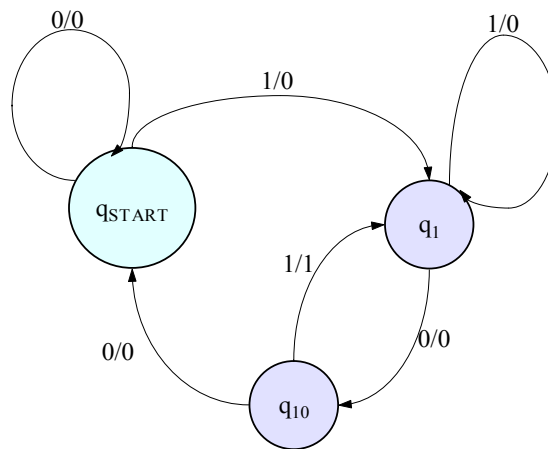
File: Riconoscitore di seq.101 Ver.A.cct

Riconoscitore di sequenza 101 (Ver. con sovrapposizioni)

Progettare una macchina sequenziale a sincronizzazione esterna capace di riconoscere la sequenza di bit 101, non considerare le diverse sovrapposizioni tra i bit in ingresso alla macchina.

Le fasi di progettazione sono simili a quelle viste per la Ver.A di tale macchina fatta in precedenza, una piccola differenza sta appunto nell'automa e ciò modifica leggermente il circuito finale. L'insieme di ingresso e quello di uscita sono già definiti poiché forniti dalle specifiche di progetto, inoltre per tali insiemi non è necessario effettuare nessuna codifica poiché la macchina da realizzare accetta in ingresso sequenze di bit e fornisce in uscita un bit, alto se viene riconosciuta la sequenza 101.

Procediamo nella costruzione dell'automa:



Le etichette associate a ciascuno stato sono significative dei bit riconosciuti fino a quell'ingresso. La macchina da realizzare è di tipo sincrono a causa di uno stato instabile quale è q_{10} . Questo automa genera dunque questa tabella di transizione:

Stati	0	1
q_{START}	$q_{START}/0$	$q_1/0$
q_1	$q_{10}/0$	$q_1/0$
q_{10}	$q_{START}/0$	$q_1/1$

L'automa si compone inoltre di 3 stati, q_{START} , q_1 e q_{10} i quali necessitano per una codifica di 2 bit. Infatti, con due bit è possibile rappresentare 4 stati in forma codificata e nel nostro caso una rappresentazione sarà inutilizzata, queste le codifiche scelte:

$q_{START} = 00$
 $q_1 = 01$
 $q_{10} = 10$

La scelta del flip-flop da utilizzare è caduta sul flip-flop D latch, la tabella di transizione degli stati è quindi codificata rispettando le funzionalità di tale flip-flop:

Stati		Ingresso	Nuovo stato		Uscita
A	B		A'	B'	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	-	-	-
1	1	1	-	-	-

Mappe di Karnough

Funzione logica A' :

AB					
I		00	01	11	10
0			1	-	
1				-	

$$A' = B\bar{I}$$

Funzione logica B' :

AB					
I		00	01	11	10
0				-	
1		1	1	-	1

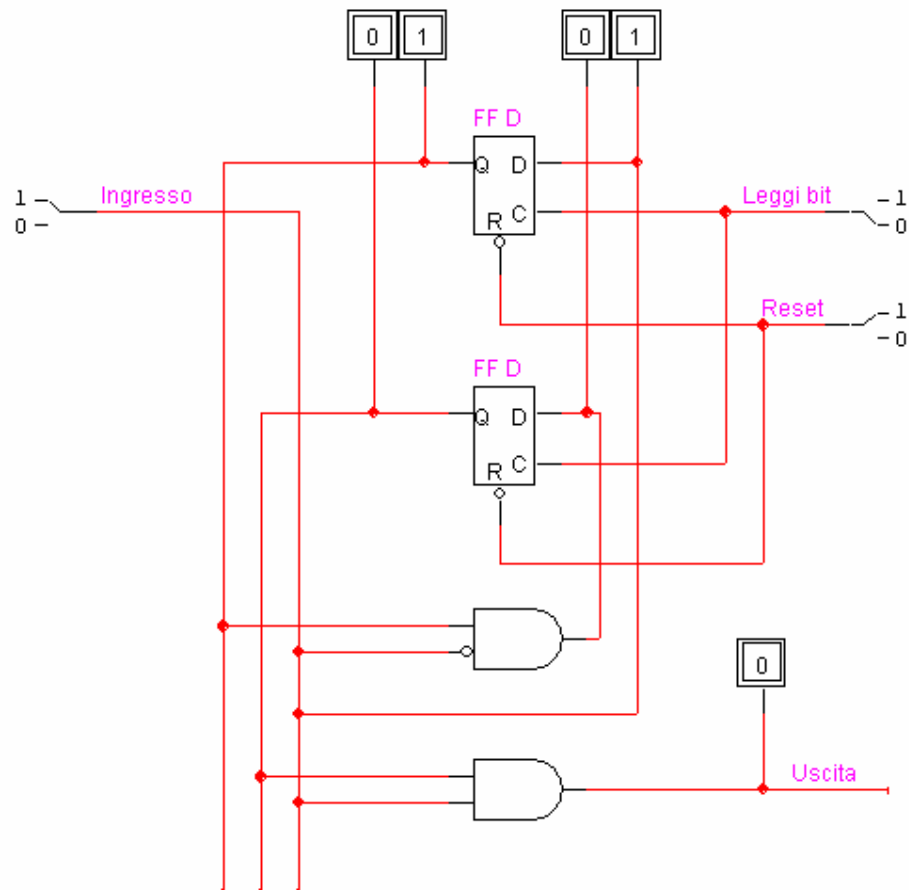
$$B' = I$$

Funzione logica U:
(Uscita)

AB					
I		00	01	11	10
0				-	
1				-	1

$$U = AI$$

Non rimane che tracciare il circuito della macchina realizzata:



File: Riconoscitore di seq.101 Ver.B.cct

Il riconoscitore di sequenze 8421

Le sequenze di bit di tipo 8421 sono sequenze di 4 bit comprese in un intervallo [0,9]. La sequenza 8421 indica appunto i pesi dovuti alla posizione di ciascun bit: 1 per il primo bit, 2 per il secondo bit, 4 per il terzo bit ed infine 8 per il bit più significativo. Realizzare una macchina che riceva in ingresso una sequenza di bit, ogni volta che nella sequenza d'ingresso si riconoscono 4 bit che compongono un numero valido nella codifica 8421 la macchina emette un segnale in uscita.

L'insieme di ingresso è costituito da bit e quindi non è necessario alcuna codifica. Come uscita decidiamo di assegnare alla funzione logica che la realizzerà il valore 1 se una delle sequenze valide comprese da 0 a 9 è stata riconosciuta, altrimenti l'uscita sarà il valore 0.

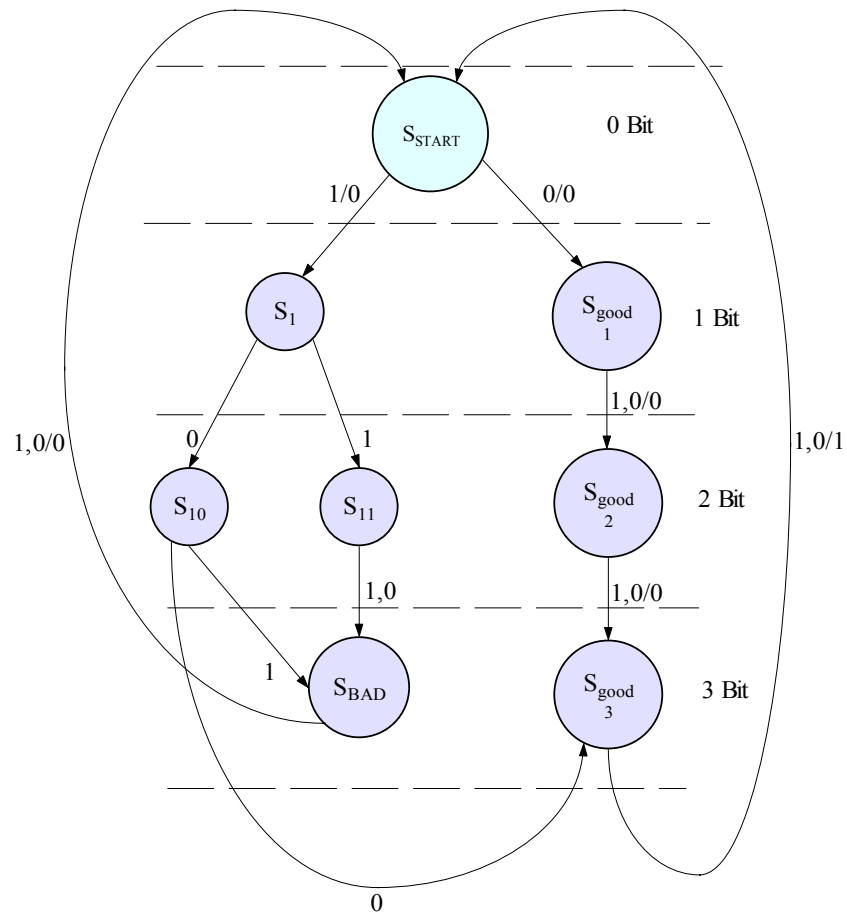
Le specifiche delle macchina sono molto generali ed è possibile fare alcune semplificazioni al progetto se immaginiamo di dare come primo ingresso alla macchina il bit più significativo e a seguire tutti gli altri. Ovviamente è possibile anche fornire come primo ingresso il bit meno significativo. Questo modo di procedere è più chiaro se si osservano le sequenze di bit che la macchina deve riconoscere:

A ₃	A ₂	A ₁	A ₀	N°
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Se infatti ipotizziamo che in ingresso sia fornito prima il bit A₃ è possibile stabilire già dal primo bit se la stringa che seguirà è corretta. Infatti nel caso in cui A₃=0 la stringa che seguirà è sicuramente idonea secondo la codifica 8412. Tuttavia il progetto della macchina sarà ripetuto sarà fatto per tutti e due i casi.

Riconoscitore 8421 Ver.A
(Primo bit in ingresso A₃)

L'automa che realizza la macchina è il seguente:



La tabella di transizione:

Stato	0	1
S _{START}	S _{good1}	S ₁
S ₁	S ₁₀	S ₁₁
S _{good1}	S _{good2}	S _{good2}
S ₁₀	S _{good3}	S _{BAD}
S ₁₁	S _{BAD}	S _{BAD}
S _{good2}	S _{good3}	S _{good3}
S _{BAD}	S _{START}	S _{START}
S _{good3}	S _{START}	S _{START}

Codifica degli stati:

Occorrono 3 bit per rappresentare gli 8 stati che compongono l'automa, questa che segue è la codifica:

S_{START} = 000
 S₁ = 001
 S_{good1} = 010
 S₁₀ = 011

$S_{11} = 100$
 $S_{good2} = 101$
 $S_{BAD} = 110$
 $S_{good3} = 111$

La macchina da progettare è di tipo sincrono, il flip-flop scelto per la progettazione è il flip-flop D, segue la tabella codificata:

Stato				Nuovo stato			
A	B	C	Ingr.	A'	B'	C'	U
0	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	1	0
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	1

Mappe di Karnough:

Funzione logica A' :

AB \ CI		00	01	11	10
0			1		1
0			1		1
1			1		1
1	1	1	1		1
1			1		1
0					

$$A' = \overline{A}B + A\overline{B} + \overline{A}CI$$

Funzione logica B' :

AB CI	00	01	11	10
0	1			1
0				
1				1
1		1		1
1	1	1		1
0				

$$B' = \overline{A}\overline{B} + \overline{B}\overline{I} + \overline{A}BC + \overline{A}C\overline{I}$$

Funzione logica C' :

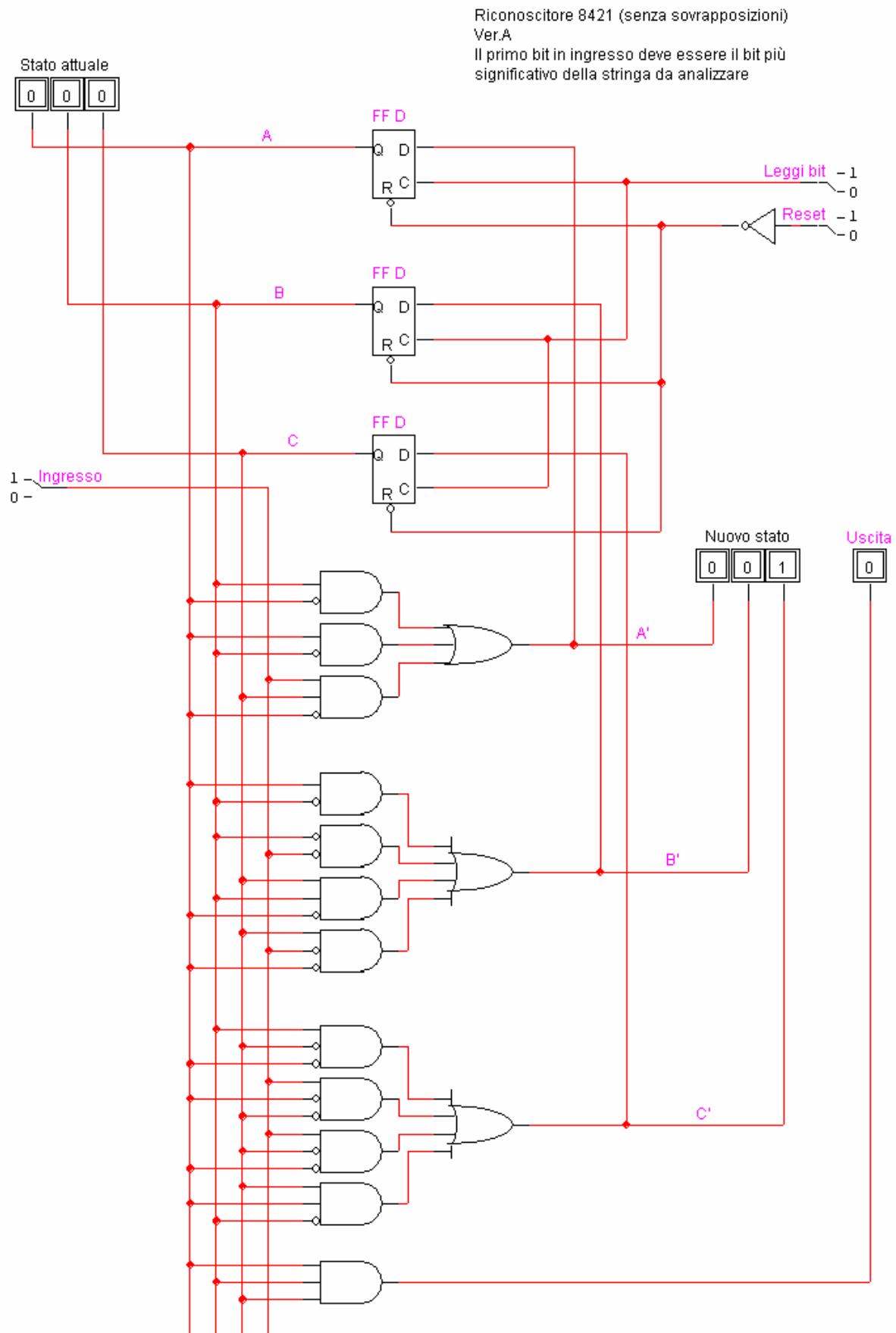
AB CI	00	01	11	10
0		1		
0				
1	1	1		
1				1
1	1	1		1
0				

$$C' = \overline{A}\overline{B}\overline{C} + \overline{C}\overline{I}\overline{A} + \overline{A}C\overline{I} + \overline{A}\overline{B}C$$

Funzione logica U:
(Uscita)

$$U = ABC$$

Circuito logico:



Riconoscitore 8421 Ver.B
(Primo bit in ingresso A_0)

Automa:

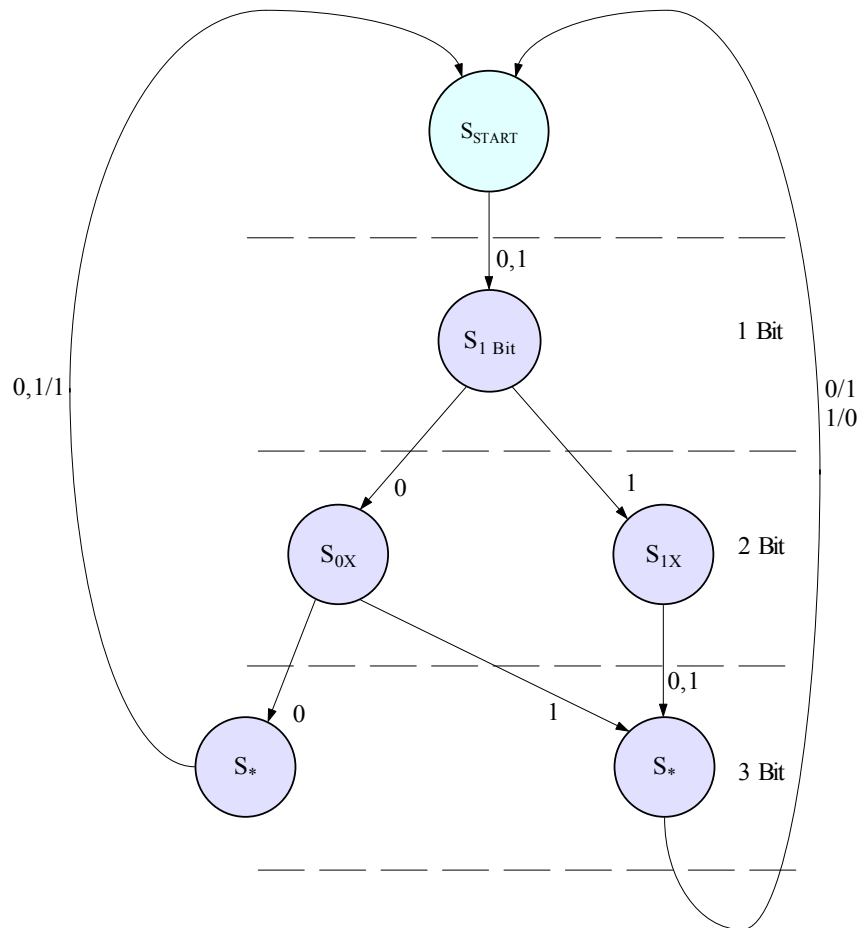


Tabella di transizione:

Stato	0	1
S_{START}	$S_1 \text{ Bit}$	$S_1 \text{ Bit}$
$S_1 \text{ Bit}$	S_{0X}	S_{1X}
S_{0X}	S_{00X}	S_*
S_{1X}	S_*	S_*
S_{00X}	$S_{START}/1$	$S_{START}/1$
S_*	$S_{START}/1$	$S_{START}/0$

Codifica degli stati, D è il flip-flop scelto:

$S_{START} = 000$
 $S_1 \text{ Bit} = 001$
 $S_{0X} = 010$
 $S_{1X} = 011$
 $S_{00X} = 100$
 $S_* = 101$

Codifica della tabella:
(Flip-flop D)

Stato				Nuovo stato			
A	B	C	Ingr.	A'	B'	C'	U
0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	0	1	0	0	0	0	1
1	0	1	1	0	0	0	0
1	1	0	0	-	-	-	-
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

Mappe di Karnough:

Funzione logica A' :

AB CI \		00	01	11	10
0	0		1	-	
0	1		1	-	
1	1		1	-	
1	0		1	-	

$$A' = B$$

Funzione logica B' :

AB \ CI	00	01	11	10
00			-	
01			-	
11	1		-	
10	1		-	

$$B' = \overline{A} \overline{B} C$$

Funzione logica C' :

AB \ CI	00	01	11	10
00	1		-	
01	1	1	-	
11	1	1	-	
10		1	-	

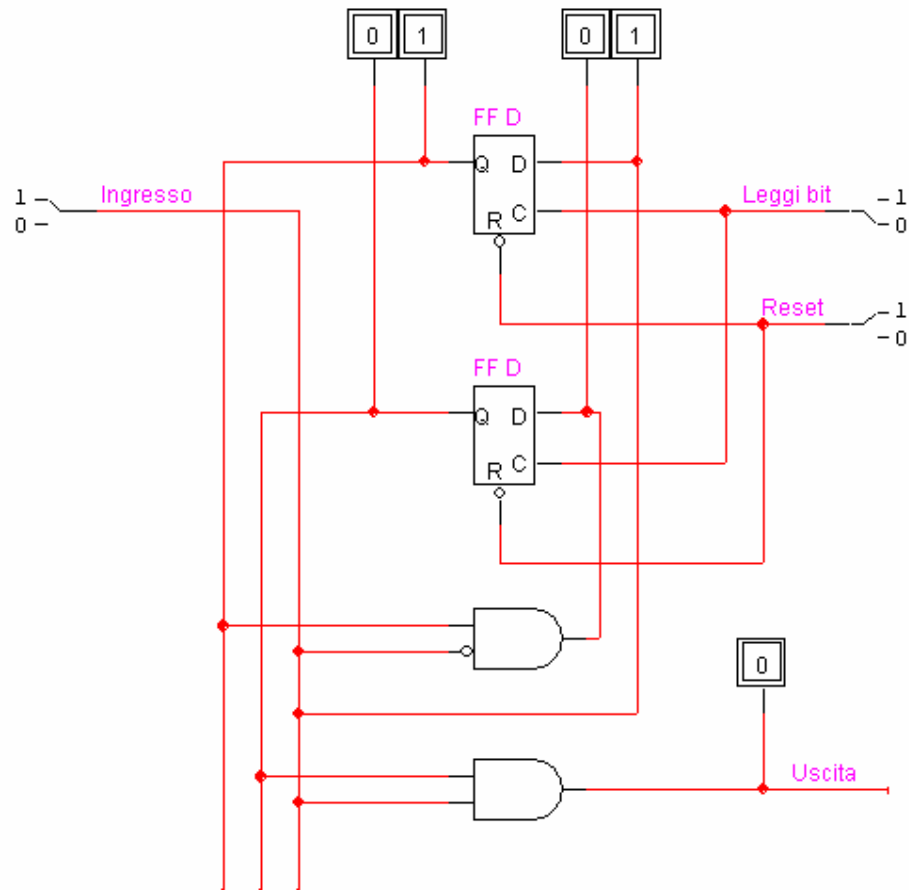
$$C' = \overline{A} \overline{B} C + \overline{A} B C + A \overline{B} C + A B C$$

Funzione logica U:
(Uscita)

AB \ CI	00	01	11	10
00			-	1
01			-	1
11			-	
10			-	1

$$U = A \overline{C} + A \overline{I}$$

Circuito logico:



Lezione N°8: "La tempificazione edge trigger e master slave, il flip-flop di tipo JK e T"

La tempificazione edge-trigger

La tempificazione edge-trigger (le cui iniziali ET sono appunto usate per specificare in modo sintetico il tipo di tempificazione adottato dal flip-flop) può avvenire sul fronte di salita oppure sul fronte di discesa di uno dei bit in ingresso al flip-flop, ad esempio, per il flip-flop D questa tempificazione è adottata per il bit di abilitazione a. Per un flip flop D sono allora valide le tempificazioni ETS (per un FF che agisce sul fronte di salita della variabile a) ed ETD (per un FF che agisce sul fronte di discesa della variabile a). Per fronte di salita s'intende la variazione e quindi l'evento $0 \rightarrow 1$ alla quale il bit di abilitazione è sottoposto.

Ancora prima di vedere la tabella che mostra la transizione degli stati è opportuno fare qualche osservazione. Le prime due righe si riferiscono a condizioni di funzionamento che prevedono in uscita $Q=0$, le ultime due righe sono invece riferite a condizioni di funzionamento in cui l'uscita è $Q=1$. Gli stati stabili vengono identificati nella tabella evidenziando questi ultimi in corrispondenza delle righe in cui i pedici risultano essere uguali a quelli riportati nella cella.

Flip-Flop D ETS

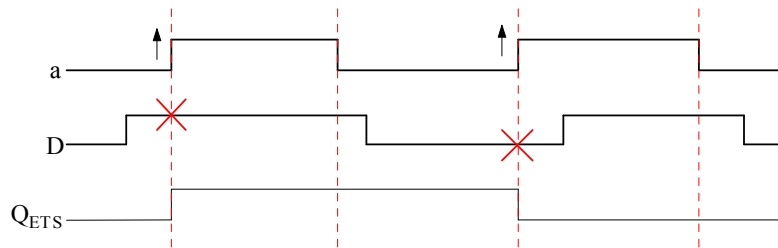
La tabella di transizione per gli stati di un FF D ETS è la seguente:

a D	00	01	11	10	
q ₀₀	q ₀₀	q ₀₁	q ₀₀	q ₀₀	Q = 0
q ₀₁	q ₀₀	q ₀₁	q ₁₁	-	
q ₁₁	q ₁₀	q ₁₁	q ₁₁	q ₁₁	Q = 1
q ₁₀	q ₁₀	q ₁₁	-	q ₀₀	

in cui q_{xy} identifica uno dei possibili stati con:

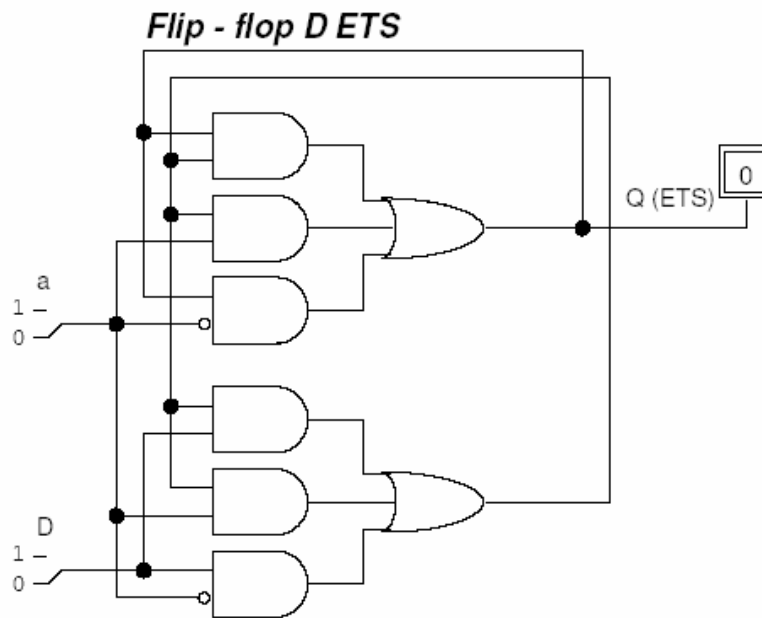
- x valore di uscita del FF;
- y valore letto o memorizzato dal FF;

Un esempio di come possono evolvere nel tempo i segnali che adottano questo tipo di tempificazione è il seguente:

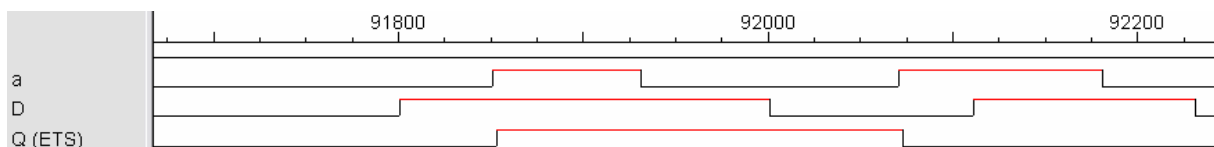


in cui si nota come in corrispondenza di un gradino verso l'alto compiuto dal segnale di abilitazione a il dato D venga considerato valido e proposto in uscita fino ad un nuovo gradino che ne può variare il valore oppure mantenere quello già esistente. Nella figura il gradino verso l'alto è riconoscibile da una freccia in corrispondenza della quale sono tracciate (sul segnale D) con una croce i valori validi per l'uscita.

Il flip - flop D ETS realizzato con Logic Works:



La sua tempificazione vista al Logic Works:

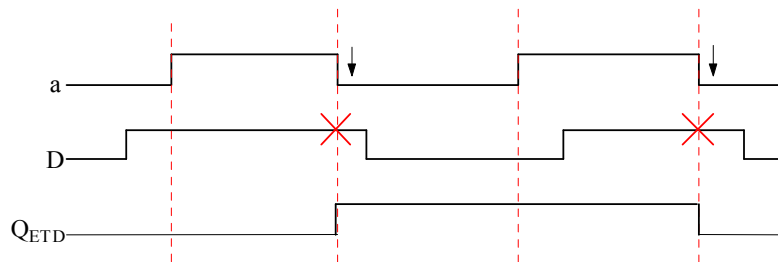


Flip-Flop D ETD

La tabella di transizione per gli stati di un FF D ETS è la seguente:

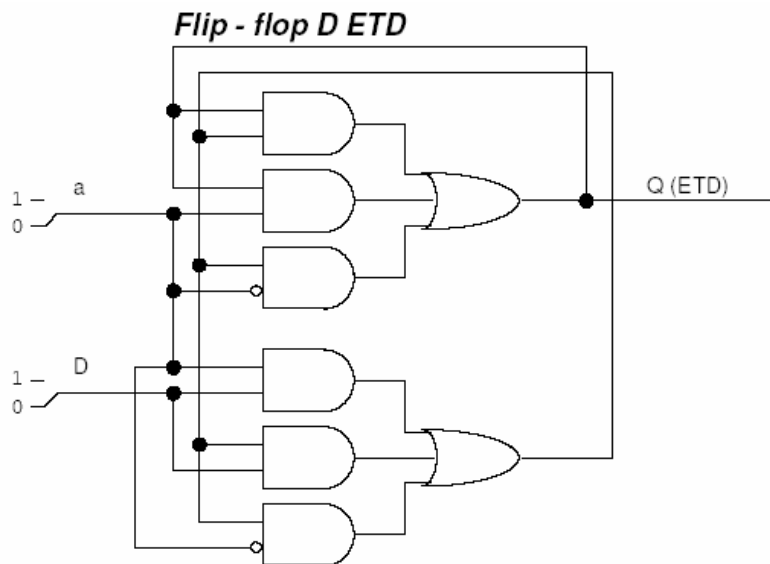
a D	00	01	11	10	
q ₀₀	q ₀₀	q ₀₀	q ₀₁	q ₀₀	Q = 0
q ₀₁	-	q ₁₁	q ₀₁	q ₀₀	
q ₁₁	q ₁₁	q ₁₁	q ₁₁	q ₁₀	Q = 1
q ₁₀	q ₀₀	-	q ₁₁	q ₁₀	

Un esempio di come possono evolvere i segnali in base a questo tipo di tempificazione è il seguente:

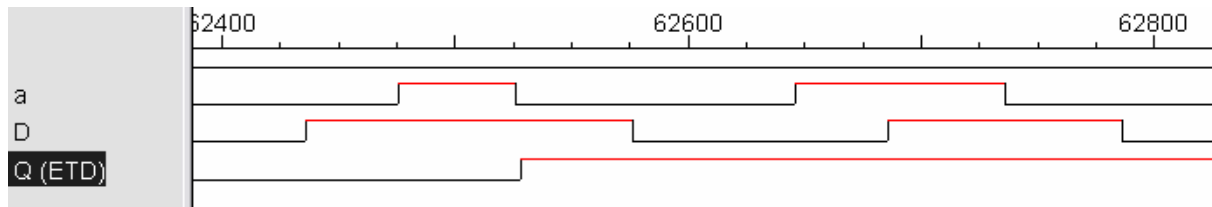


Il FF D ETD è un caso duale a quello ETS, basta osservare le due tabelle per notare il fatto che a partire da una è possibile ottenere l'altra semplicemente traslando le colonne, scambiando cioè la colonna 00 con quella 10 e la colonna 01 con quella 11.

Il flip - flop D ETD realizzato con Logic Works:



La sua tempificazione vista al Logic Works:



Nella tabella fin qui viste si notano alcune simmetrie, è consuetudine dividere questa tabella in due parti, una riguarda le prime due righe (dove si fa riferimento all'uscita $Q=0$) e l'altra riguarda le restanti due righe (che fanno quindi riferimento all'uscita $Q=1$). Ogni stato è caratterizzato da due pedici ciascuno dei quali specifica alcune situazioni tipiche di quell'evento. In particolare:

q_{xy} indica con x il valore di uscita e con y il valore letto

Come si costruisce la tabella D ETS:

Si parte da uno stato stabile e si prosegue alternando i possibili ingressi ad esso adiacenti, ricordando che è possibile far mutare un solo letterale per volta.

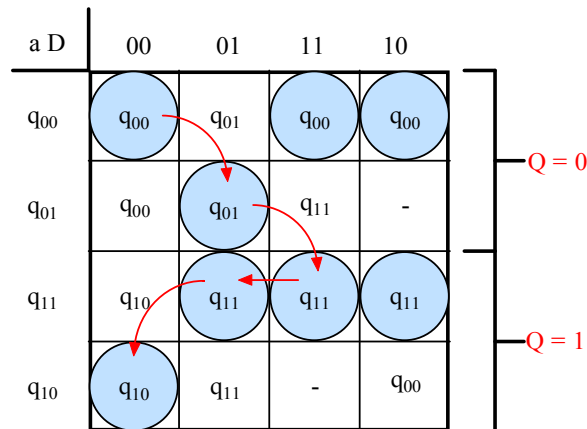
Supponiamo di partire dallo stato q_{00} , stato stabile per il FF ETS, caratterizzato da un uscita $Q=0$ (che è il valore del primo pedice) e da un valore letto in precedenza $D=0$ (che è il valore del secondo pedice); quindi per q_{00} (cerchiato poiché corrisponde al particolare stato indicato da quella riga e quindi stabile) può capitare che si alzi il bit D (cioè $aD=01$) oppure può succedere che si alzi il bit a (cioè $aD=10$), stato di partenza da noi scelto è $aD=00$.

Se accade che $aD=01$, non vi è dubbio che non si verifica alcuna variazione di salita ($00 \rightarrow 01$), anche se il dato D letto (ma non memorizzato) risulta alzarsi l'uscita continuerà ad essere zero (è questo il valore del primo pedice che dovremo scrivere) mentre il valore che si è letto è 1 (valore del secondo pedice). Pertanto in corrispondenza di quella cella scriveremo quello stato rappresentativo di quest'evento che è q_{01} .

Lo stato q_{01} non è però uno stato stabile per la riga in cui ci troviamo (quindi non andrà cerchiato), si procede ripetendo tale stato nella colonna in cui ci troviamo fino ad arrivare alla riga rappresentativa dello stato q_{01} che quindi cerchieremo (nel nostro caso q_{01} è ricopiato nella riga sottostante).

Torniamo ad analizzare l'altro caso, e cioè, supponiamo lo stato q_{00} e consideriamo adesso l'altro possibile ingresso che ci rimane da analizzare che è quello in cui da $00 \rightarrow 10$; qui si verifica una variazione sul fronte di salita, il dato D andrà letto e trattandosi di zero avrò in uscita $Q=0$ (valore del primo pedice) mentre il valore del dato letto è $D=0$ (valore del secondo pedice). Lo stato rappresentativo di quest'evento è q_{00} (che andrà cerchiato). La tabella va quindi completata in questo modo procedendo per i soli stati stabili, in corrispondenza della caselle vuote che rimarranno alla fine di tutto si inserisce un don't care -.

Gli stati non stabili sono detti di transizione e servono a portare la macchina in uno stato stabile facendo compiere a quest'ultima un movimento verticale sulla tabella del flip-flop, la tabella di flusso può essere vista anche in questo modo:



così facendo si evidenziano le transizioni (freccie rosse) che la macchina compie nel passare da uno stato stabile ad un altro.

Analizziamo la seconda riga, ci troviamo nello stato q₀₁ con aD=01. Sono allora possibile due casi: il bit a diventa alto aD=11 oppure il bit D si abbassa aD=00.

Da aD=01 ad aD=11 si verifica un fronte di salita sul bit di abilitazione, tale evento è importante per il nostro flip-flop che passa quindi a considerare il dato D il quale è quindi memorizzato. Pertanto, da q₀₁ il flip-flop si sposta in q₁₁. Tale stato non è poi stabile per la riga in cui ci troviamo ed è quindi ricopiato nella cella sottostante.

Da aD=01 ad aD=00 non avviene alcun fronte di salita per cui l'uscita rimane immutata (il valore del primo pedice è quindi 0) così come l'ultimo valore letto è ancora lo stesso (il valore del secondo pedice è 0).

Da aD=01 non possiamo passare direttamente ad aD=10 poiché i bit in ingresso devono necessariamente cambiare uno per volta, nella cella caratterizzata da un ingresso aD=10 il flip-flop non transiterà in nessun modo per cui si segna quella cella con un don't care.

Analizziamo la terza riga partendo dallo stato stabile q₁₁ con ingresso aD=11. E' questa la situazione più tranquilla per il flip-flop, poiché per avvenire un fronte di salita il bit di abilitazione si deve prima abbassare e poi nuovamente rialzare, inoltre pur variando il bit D lo stato del flip-flop non cambia per i motivi prima detti a riguardo del bit di abilitazione.

Pertanto da aD=11 ad aD=10, si abbassa cioè il bit D, lo stato continua ad essere q₁₁ così come da aD=11 ad aD=01. Ci spostiamo quindi in una nuova cella caratterizzata da uno dei due nuovi stati da noi segnati in precedenza ed analizziamo le possibili variazioni, trascurando gli ingressi che prevedono un ritorno alla cella di partenza. Supponiamo di trovarci nello stato q₁₁ con aD=01, il successivo ingresso sia aD=00, il bit D si abbassa.

In tale circostanza il flip-flop fornisce in uscita sempre il valore prima memorizzato e cioè $Q=1$, tuttavia il bit 0 va considerato poiché se al prossimo ingresso si dovesse alzare il bit di abilitazione (fronte di salita) è allora necessario memorizzare tale bit, per cui il nuovo stato è q_{10} .

Analizziamo la quarta ed ultima riga partendo quindi dallo stato q_{10} stabile per quella riga e quindi cerchiato in cui $aD=00$.

I possibili ingressi sono $aD=01$ (si alza il bit D) oppure $aD=10$ (si alza il bit a, nuovo fronte di salita)

Supponiamo che si verifichi la transizione degli ingressi da $aD=00$ ad $aD=10$, si verifica cioè un nuovo fronte di salita, il bit D viene quindi memorizzato per cui $Q=0$ (primo pedice) ed il valore letto è appunto 0 (secondo pedice), il nuovo stato è q_{00} .

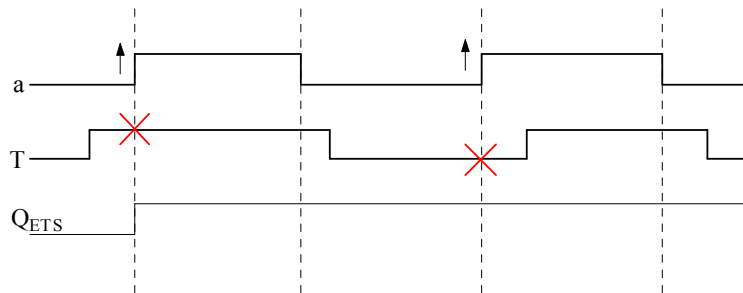
Supponiamo ora la transizione degli ingressi da $aD=00$ ad $aD=01$, la colonna degli ingressi $aD=01$ è di pre-allarme per il nostro flip-flop che opera sul fronte di salita del bit a, se infatti nel successivo ingresso dovesse capitare $a=1$ allora il bit D che in precedenza è stato posto in ingresso al flip-flop va memorizzato, l'uscita quindi non cambia ed è $Q=1$, la stessa di prima non essendoci variazione sul fronte di salita, il valore letto è allora il bit $D=1$, letto ma non memorizzato. Pertanto il nuovo stato è q_{11} . Infine, in ingresso, non potendo variare più di un bit l'ingresso $aD=11$ non è contemplato per questa riga.

Flip-flop T

Il FF T è anche detto a commutazione, esso presenta un valore tipico dei bit in ingresso per la quale è prevista la commutazione degli stati di uscita. Il FF T è caratterizzato da:

- Due ingressi (bit a, bit T);
- Un uscita;

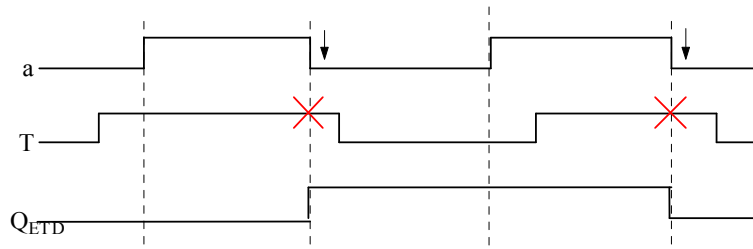
Il segnale a svolge il ruolo di abilitazione, il segnale T commuta il valore memorizzato dal FF, l'uscita è il valore memorizzato dal flip flop. Al FF T, così come a quelli fin qui visti, è possibile associare una delle tempificazioni note (~~latch~~, edge trigger, e master slave). Tuttavia il FF T non deve mai essere di tipo latch, vediamone il perché: il FF T commuta quando il segnale di abilitazione è alto, se il segnale rimane alto il FF commuta ancora e lo fa ininterrottamente (cioè all'infinito) alternando i valori di uscita. Una tempificazione lecita, per un flip flop di tipo T è quella ETS, cioè a variazione sul fronte di salita di cui ne mostriamo il diagramma temporale degli ingressi:



La tabella del FF di tipo T ETS è:

a T	00	01	11	10	
q ₀₀	q ₀₀	q ₀₁	q ₀₀	q ₀₀	Q = 0
q ₀₁	q ₀₀	q ₀₁	q ₁₁	-	
q ₁₁	q ₁₁	q ₁₀	q ₁₁	q ₁₁	Q = 1
q ₁₀	q ₁₁	q ₁₀	q ₀₀	-	

FF T ETD



La tabella del FF di tipo T ETD è:

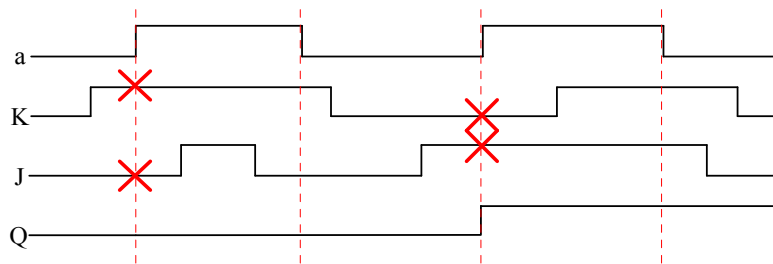
a T	00	01	11	10	
q ₀₀	q ₀₀	q ₀₀	q ₀₁	q ₀₀	Q = 0
q ₀₁	-	q ₁₁	q ₀₁	q ₀₀	
q ₁₁	q ₁₁	q ₁₁	q ₁₀	q ₁₁	Q = 1
q ₁₀	-	q ₀₀	q ₁₀	q ₁₁	

Il flip-flop JK

Il flip-flop JK è un particolare F che riassume in unico componente le funzionalità di un FF RS e di un FF T. Il suo comportamento è infatti simile a quello del FF RS, tuttavia nel caso in cui in ingresso si presenta $J=1$ e $K=1$ (ricordiamo a proposito che un FF RS non prevede un ingresso $R=1$ $S=1$) esso compie una commutazione del proprio stato interno, in altre parole si comporta come un FF T. Quando l'ingresso è tale che $K=J$, allora il flip-flop si comporta come T, quando invece $J \neq K$ allora il comportamento del FF JK è riconducibile a quello di un FF RS. La tabella di transizione per questo FF è:

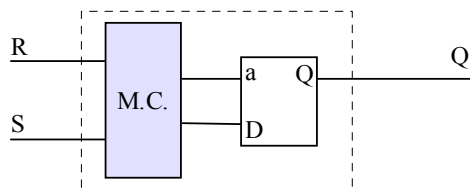
(R)(S) K J	A=0				A=1				
	00	01	11	10	00	01	11	10	
q ₀₀	q ₀₀	q ₀₁	q ₀₁	q ₀₀	q ₀₀	q ₀₀	q ₀₀	q ₀₀	Q = 0
q ₀₁	q ₀₀	q ₀₁	q ₀₁	q ₀₀	-	q ₁₁	q ₁₁	-	
q ₁₁	q ₁₁	q ₁₁	q ₁₀	q ₁₀	q ₁₁	q ₁₁	q ₁₁	q ₁₁	Q = 1
q ₁₀	q ₁₁	q ₁₁	q ₁₀	q ₁₀	-	-	q ₀₀	q ₀₀	

Una possibile tempificazione dei segnali è:



Il problema del pilotaggio di un flip-flop

Il flip-flop JK si comporta da RS e da T ma non da FF D, come usarlo se si vuole pilotarlo in tale modo? Questo è il problema del pilotaggio di un flip-flop, usare cioè un diverso flip-flop pilotandolo come se fosse realmente il flip-flop che si vuole usare. Un problema di questo tipo è risolto interponendo tra il flip-flop a disposizione e i segnali che si vogliono in ingresso una opportuna macchina combinatoria, ad esempio:



nel disegno appena mostrato si è pensato di pilotare un flip-flop D come se questo fosse un flip-flop RS. Con M.C. si intende la macchina combinatoria da realizzare affinché vengano implementati i segnali R ed S. Tale macchina accetterà in ingresso i segnali R ed S e trasformerà tali segnali in uno dei valori dell'insieme $aD=\{00,01,10,11\}$.

a	D	R	S
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Questa tabella va così interpretata:

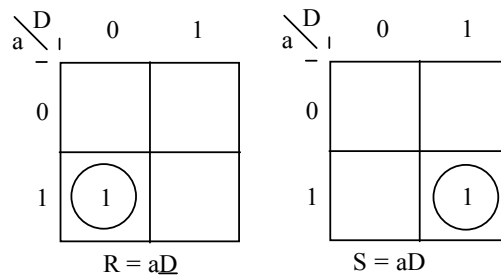
Per $aD=00$ in ingresso si dovranno avere $RS=00$; Per $aD=01$ in ingresso si dovranno avere $RS=00$; Per $aD=10$ (memorizza zero) in ingresso si dovranno avere $RS=10$; Infine, per $aD=11$ (memorizza uno) in ingresso si dovranno avere $RS=01$.

Osservazione: i bit RS mai diventano alti per cui il nostro progetto (almeno su carta) è corretto. Il flip-flop dovrà memorizzare uno zero quando l'abilitazione è alta (a) e il dato è basso (D) e quindi $R=a\bar{D}$, si dovrà invece memorizzare un bit alto quando l'abilitazione è alta (a) e il bit è anch'esso alto (D), quindi $S=aD$:

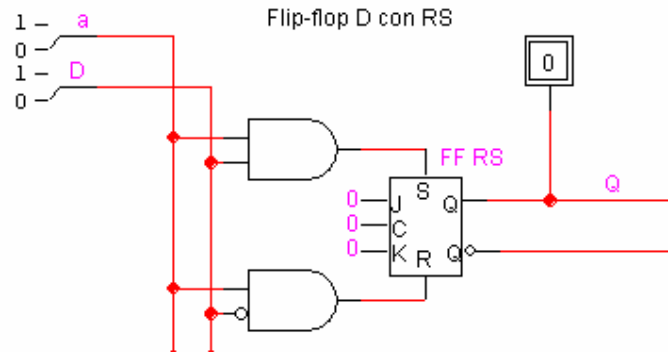
$$R = a\bar{D}$$

$$S = aD$$

Come è tra l'altro possibile ricavare dalle mappe di Karnough:



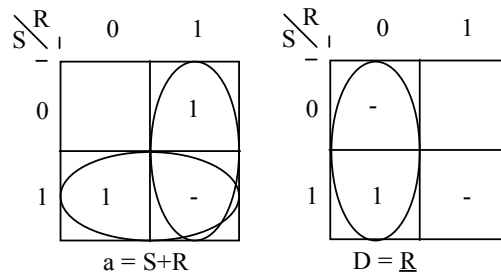
Questo il circuito:



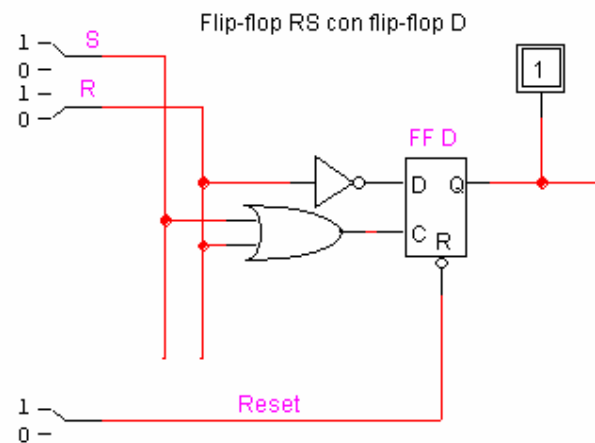
Ad esempio, realizzare un FF RS pilotando un FF D:

Dai possibili ingressi di R ed S si ricava la tabella che associa a tali bit i segnali a e D del flip-flop D:

R	S	a	D
0	0	0	-
0	1	1	1
1	0	1	0
1	1	-	-



Circuito logico:

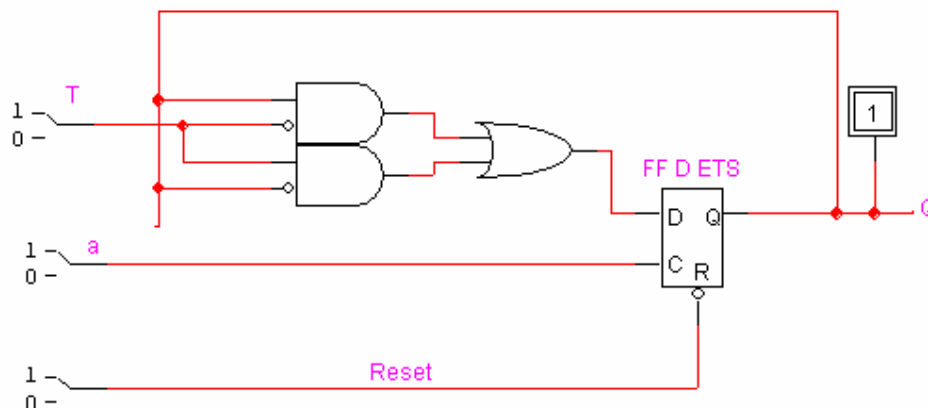


Un diverso approccio alla progettazione è invece richiesto per la realizzazione ad esempio di un flip-flop T pilotando un flip-flop D. Per prima cosa è bene precisare che il flip-flop da usare sarà tempificato secondo un fronte di salita, nell'esempio che vedremo tra poco si è scelto di usare un flip-flop D ETS. Se infatti si usa un flip-flop D latch questo commuterebbe ininterrottamente il valore di uscita, cosa che non rispetta il progetto da realizzare. L'abilitazione del flip-flop D è di fatto anche l'abilitazione del flip-flop T poiché esse hanno lo stesso comportamento, inoltre, nella costruzione della tabella di verità è opportuno prendere in considerazione l'uscita Q del flip-flop T, è in base a questo valore che si dovrà poi scegliere come comandare il bit T:

D	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

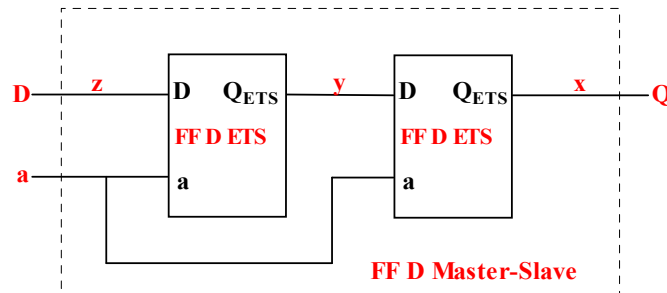
Da cui si ottiene:

$$T = \overline{D}Q + D\overline{Q}$$



La tempificazione Master-Slave

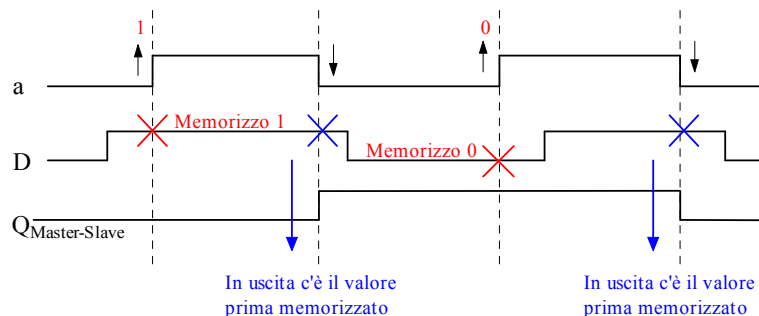
I flip-flop che adottano questo tipo di tempificazione leggono il dato sul fronte di salita del segnale di abilitazione e lo ripropongono in uscita sul fronte di discesa del medesimo segnale di abilitazione. E' possibile realizzare un FF D Master-Slave (analogamente un FF T) usando un FF D ETS ed uno ETD, così come si vede in figura:



Il primo FF memorizza il dato sul fronte di salita del segnale di abilitazione **a** mentre il secondo lo propone in uscita sul fronte di discesa del medesimo segnale. Il significato e l'uso dei simboli **x, y, z**, è il seguente:

- **x**, valore di uscita del FF;
- **y**, valore memorizzato;
- **z**, valore da memorizzare;

ed a breve sarà più chiaro, quando cioè si osserverà la tabella di transizione per gli stati del FF Master-Slave. Una possibile tempificazione per i segnali coinvolti nell'uscita del flip flop è la seguente:



Dalla tempificazione dei segnali notiamo che, sul fronte di salita di **a** viene memorizzato il valore 1 (è possibile individuare quest'evento notando che in corrispondenza di tale salita una croce rossa segna appunto il valore memorizzato dal primo flip-flop, quello ETS).

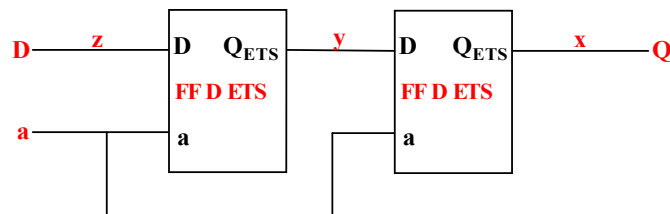
Quando il segnale di abilitazione compie una discesa il valore prima memorizzato viene letto dal secondo flip flop (quello ETD) e messo in uscita. L'uscita del secondo FF costituisce di fatto l'uscita del flip flop D Master-Slave.

La tabella di transizione per questo tipo di tempificazione abbinata alla funzionalità di un FF D è la seguente:

a D	00	01	11	10	
q ₀₀₀	q ₀₀₀	q ₀₀₁	q ₀₀₀	q ₀₀₀	Q = 0
q ₀₀₁	q ₀₀₀	q ₀₀₁	q ₀₁₁	-	
q ₀₁₁	q ₁₁₀	q ₁₁₁	q ₀₁₁	q ₀₁₁	
q ₁₀₀	q ₀₀₀	q ₀₀₁	q ₁₀₀	q ₁₀₀	Q = 1
q ₁₁₁	q ₁₁₀	q ₁₁₁	q ₁₁₁	q ₁₁₁	
q ₁₁₀	q ₁₁₀	q ₁₁₁	-	q ₁₀₀	

La costruzione di questa tabella risulta più agevole se fatta tenendo conto dello schema generale di un FF D Master-Slave(quello cioè visto all'inizio di questa lezione) e alternando ai valori x, y e z quelli risultanti dagli ingressi a e D.

Ad esempio:



Se ci troviamo sulla riga q₀₀₀ e l'ingresso è 00, non succede nulla quindi il FF rimane fermo nello stato q₀₀₀ (Stato stabile per questa riga e quindi cerchiato). Procedendo con i possibili ingressi, ipotizziamo che aD=01, quindi il segnale D si alza, tuttavia non è avvenuta alcuna salita sul segnale di a e quindi il primo flip-flop non memorizza niente, l'uscita è zero ed il valore da memorizzare è proprio D (nel caso in cui si dovesse alzare solo l'abilitazione a allora il dato da memorizzare è preso in considerazione dal primo FF).

Lo stato precedentemente ottenuto non è stabile rispetto alla riga in cui ci si trova per cui si ricopia tale stato in corrispondenza della giusta riga (che è la riga appena sotto), inoltre lo stato precedentemente ottenuto non ci consente di procedere alla stessa maniera di come finora stavamo facendo (la tabella che si stava costruendo procedeva infatti verso destra) per cui dobbiamo ritornare allo stato q₀₀₀ e ipotizzare in ingresso aD=10: per tale ingresso si nota una salita sul segnale di abilitazione a per cui il valore di D=0 è letto dal primo FF che lo memorizza, l'uscita

nonostante ciò non cambia poiché non si è verificata ancora un fronte di discesa sul segnale di abilitazione a.

Procediamo adesso con la seconda riga. Nel ricavare la riga precedente avevamo già ottenuto per questa riga un valore utile (quello di q_{001} in corrispondenza di $aD=01$), questo valore costituisce adesso il nuovo punto di partenze.

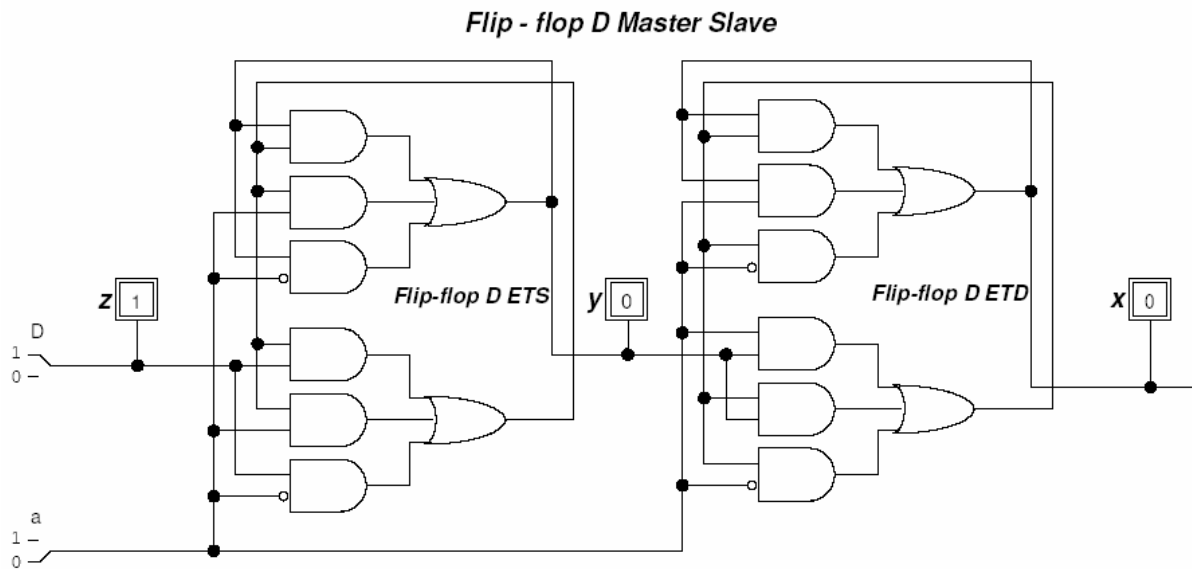
Siamo in q_{001} (stato stabile per riga la in questione) ed ipotizziamo un ingresso che da $aD=01$ si sposta ad $aD=11$, si verifica quindi un fronte di salita per il segnale di abilitazione a, quindi il valore $D=1$ è letto e memorizzato dal primo FF (valore del pedice y), l'uscita del FF Master-Slave non cambia, per cui $z=0$ e il valore da memorizzare è proprio D, quindi ci spostiamo nello stato q_{011} che non essendo stabile per la riga in cui ci troviamo deve essere ricopiato in quella successiva (che è appunto in riferimento a q_{001}), inoltre questo stato ricavato non essendo stabile comporta (come fatto in precedenza) un passo indietro (considerare cioè lo stato di partenza q_{001}) ed ipotizziamo l'altro ingresso $aD=00$. Nel passare da $aD=01$ ad $aD=00$ si verifica l'abbassamento del segnale D mentre al segnale di abilitazione non accade nulla, per cui il valore da memorizzare (nel caso in cui dovesse avvenire un fronte di salita sul segnale a) è 0 che è anche il valore del pedice z, il valore memorizzato è sempre 0 ed infine il valore di uscita è ancora 0 ($x=0$, $y=0$, $z=0$) quindi il nuovo stato sarà q_{000} .

Nel completare la tabella con questa riga non vi è stato alcuno modo di arrivare alla casella individuata dall'ingresso $aD=10$ per cui questa casella sarà occupata da un don't care.

La tabella riprende ora dalla riga inerente allo stato q_{011} in cui precedentemente si era arrivato attraverso l'ingresso $aD=11$, ipotizziamo ora l'ingresso che da $aD=10$ si sposta ad $aD=00$, si verifica dunque un fronte di discesa sul segnale di abilitazione che interesserà ora il secondo flip-flop, infatti: il valore memorizzato sarà considerato dal secondo FF, per cui $x=1$; il valore memorizzato è ancora quello di prima (e che adesso è considerato dal FF ETD), quindi $y=1$, mentre il valore da memorizzare è $z=0$, il nuovo stato sarà q_{110} (che non è stabile per la riga in questione, sarà quindi copiato in corrispondenza della riga a cui fa riferimento quello stato). Il nuovo stato ricavato non ci consente di proseguire nel senso in cui stavamo procedendo, consideriamo quindi di nuovo il punto di partenza e ipotizziamo questa volta gli ingressi che da $aD=10$ prevedono $aD=11$: si verifica un cambiamento del solo segnale D, tuttavia non essendoci nessuna variazione sul fronte di a lo stato prossimo continua ad essere q_{011} (che è quindi ancora stabile per la riga in questione).

L'ultimo ingresso da prevedere è quello che da $aD=11$ prevede l'ingresso $aD=01$, si verifica cioè un variazione sul fronte di discesa sul segnale di abilitazione a, quindi il valore memorizzato in precedenza (cioè il valore 1) viene ora preso in consegna dal secondo FF (quello ETD per intenderci) che lo propone in uscita, quindi $x=1$, il valore memorizzato è sempre quello di prima $y=1$ infine il valore da memorizzare è quello indicato da D e quindi 1 ($z=1$), il nuovo stato sarà q_{111} , non stabile per la riga in esame e

quindi ricopiato in corrispondenza della riga a cui esso fa riferimento. La restante tabella è ricavata alla stessa maniera. Il flip-flop D Master-Slave realizzato al Logic Works:



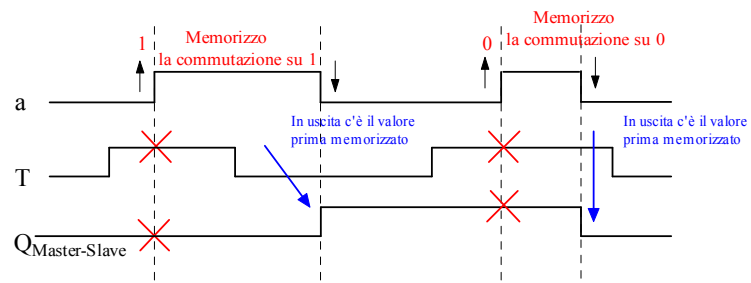
Nel disegno è possibile notare i due FF D con le rispettive tempificazioni ETS (la parte di circuito più a sinistra del foglio) ed ETD (la parte di circuito più a destra del foglio), i probe disposti sul foglio permettono appunto di notare i valori x,y e z usati per la tabella di transizione degli stati.

Applichiamo ora la tempificazione Master slave alle funzionalità del flip-flop T, questa la tabella:

a T	00	01	11	10	
q ₀₀₀	q ₀₀₀	q ₀₀₁	q ₀₀₀	q ₀₀₀	Q = 0
q ₀₀₁	q ₀₀₀	q ₀₀₁	q ₀₁₁	-	
q ₀₁₁	q ₁₁₁	q ₁₁₀	q ₀₁₁	q ₀₁₁	
q ₁₁₁	q ₁₁₁	q ₁₁₀	q ₁₁₁	q ₁₁₁	
q ₁₁₀	q ₁₁₁	q ₁₁₀	q ₁₀₀	-	Q = 1
q ₁₀₀	q ₀₀₀	q ₀₀₁	q ₁₀₀	q ₁₀₀	

Da notare è la sequenza degli stati ordinati e disposti lungo la colonna in questo modo poiché evidenziano di più le simmetrie esistenti tra la prima parte della tabella, quella per Q=0 e la seconda parte, quella per Q=1.

Un esempio di tempificazione:



Lezione N°9: "Contatori"

I contatori sono particolari macchine sequenziali, che, come lascia intendere la parola stessa contano un valore dato come ingresso alla macchina. L'importanza dei contatori è di notevole rilevanza poiché essi costituiscono il comportamento di numerose macchine (complesse e non) ottenute per decomposizione.

Tipicamente si usa chiamare un contatore come **contatore modulo-N** poiché esso conta in modulo N gli impulsi e le variazioni di livello di un segnale assunto come segnale di conteggio. Un contatore modulo-N si presenta con **n** stati ordinati, se a partire dall'uscita u_i si applica una sequenza di ingresso che contiene **r** volte la sequenza di conteggio **c**, il contatore finirà nello stato u_k con:

$$k = (i + r) \bmod n$$

contando cioè in modulo n il numero di volte che si è presentato l'evento.

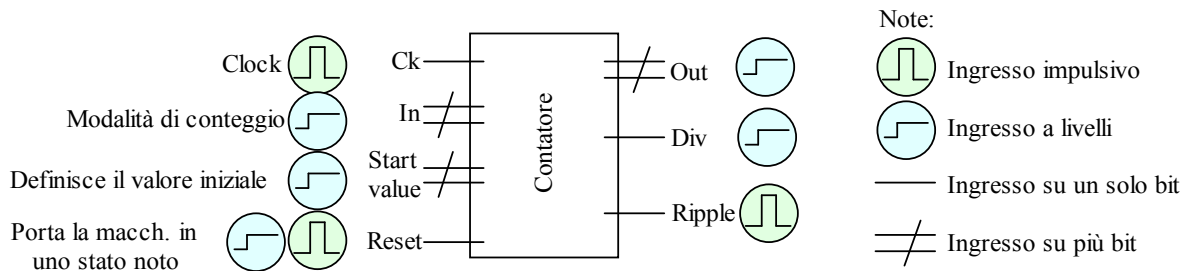
Generalmente un contatore riceve in ingresso i segnali di:

- incremento del conteggio (può trattarsi di un fronte di salita o discesa, oppure di un impulso);
- abilitazione al conteggio (fornisce al contatore l'inizio di un nuovo conteggio);
- altri segnali ausiliari come:
 - o segnale per individuare se il conteggio è a crescere oppure a decrescere;
 - o segnale di reset, che azzerà il contatore, oppure lo porta in uno stato noto;
 - o segnale per identificare lo stato o il valore iniziale del contatore;

I segnali di uscita, tipici di un contatore, sono invece:

- valore del conteggio;
- segnale di **div**, per indicare che il contatore ha un valore di conteggio pari ad N-1 si usa il segnale di div alto nello stato più significativo della macchina per contatori a crescere (ad esempio un contatore modulo 8 ha un segnale div alto quando la macchina è in S_7 che costituisce l'uscita massima 0,1,2,...,7), mentre per contatori a decrescere **div** è alto in S_0 . Il segnale div è un segnale a livelli ed è mantenuto attivo per tutta la durata del conteggio;
- segnale di **ripple**. Si tratta di un segnale, di tipo impulsivo, per indicare la transizione dal valore (N-1)→0 per contatori a crescere oppure da 0→(N-1) per quelli a decrescere;

La figura che segue riassume brevemente quanto abbiamo finora detto:



La classificazione dei contatori è basata ovviamente sulla sincronizzazione della macchina e dagli ingressi che essa può accettare. Si avranno allora contatori sincroni o asincroni, che contano impulsi o sequenze a livello sui fronti di salita e/o discesa.

Per realizzare un contatore modulo- N si usano, tipicamente, contatori modulo- k , in cui sia necessariamente $k < N$, ed in particolare si adoperano quelli per cui $k=2$ e cioè contatori modulo 2 per i quali (supponendo che il contatore sia a crescere):

- a- L'uscita di conteggio coincide con l'uscita **div**;
- b- Il segnale **ripple** è associato alla variazione da 1 a 0 e quindi sul fronte di discesa

Un contatore modulo-2 non è altro che un flip-flop di tipo T, non sono rari i casi in cui a volte si usano anche flip-flop JK come contatori modulo-2.

Esistono altre classificazioni sui contatori:

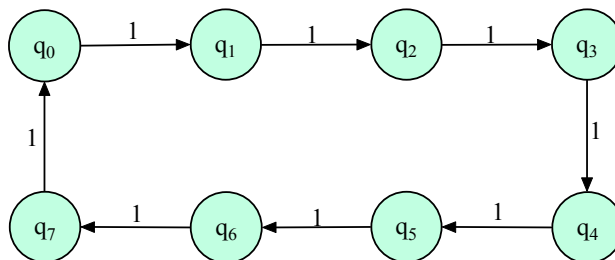
- **contatore binario**: se la sequenza delle uscite $u_0 u_1 u_2 \dots u_{n-1}$ è codificata secondo l'aritmetica binaria;
- **contatore decimale**: se cioè $n=10$ e le uscite sono un codice per la rappresentazione delle cifre decimali;
- **contatore a crescere**: se la sequenza di uscita è una sequenza ordinata di valori in modo crescente, ad esempio 0,1,2,3,...,9 per un contatore decimale;
- **contatore a decrescere**: se la sequenza di uscita, contrariamente a quanto avviene per quello a crescere, è una sequenza ordinata in modo decrescente, ad esempio 9,8,7,...,0;
- **contatore bilaterale**: se possiede due valori di conteggio, uno a crescere ed uno a decrescere;

Esempio 1: Contatore modulo 8 con FF D ETS

La macchina che vogliamo realizzare è caratterizzata da 2 ingressi (Ck = clock e Reset = reset) e da 2 uscite (Out = è una uscita su tre bit, più avanti sarà specificato; l'altra uscita è DIV). Il grafo che realizza l'automa è il seguente:

Unico valore in ingresso alla macchina è di fatti il solo bit 1, il ramo che riporta uno zero in ingresso non è infatti segnato nel grafo non essendo un ingresso valido. Inoltre, i valori delle uscite sono i valori attualmente contati, per cui la codifica più ovvia è quella che associa allo stato attuale l'equivalente in binario.

- DIV è alto nello stato S_7 ;
- OUT corrisponde allo stato in cui ci troviamo;



La tabella degli stati e delle uscite è:

0	1	Div	Out
S_0	S_1	0	0
S_1	S_2	0	1
S_2	S_3	0	2
S_3	S_4	0	3
S_4	S_5	0	4
S_5	S_6	0	5
S_6	S_7	0	6
S_7	S_0	1	7

Gli stati rappresentano il valore attualmente contato, il modo più semplice di sviluppare la codifica è rappresentare ciascuno stato con tre bit la cui interpretazione con la notazione posizionale è proprio il valore memorizzato.

Quindi dei 2 ingressi, quello su tre bit si riferisce allo stato da assegnare alla macchina.

Anche le due uscite devono essere codificate, nel nostro caso l'uscita DIV è già codificata (poiché dalle specifiche della macchina DIV è un bit, che se alto indica l'avvenuta conta in modulo 8 della macchina) mentre l'uscita OUT corrisponde ai valori contati, una giusta codifica per questa uscita è quella già adottata per l'ingresso su tre bit. Ecco allora la tabella che mostra la codifica degli stati:

$S_0 = 000$
 $S_1 = 001$
 $S_2 = 010$
 $S_3 = 011$
 $S_4 = 100$
 $S_5 = 101$
 $S_6 = 110$
 $S_7 = 111$

La scelta del flip-flop è ricaduta sul FF di tipo D ETS, questo condiziona fortemente la tabella che andremo a codificare, unica uscita da realizzare per la macchina combinatoria è il bit DIV, alto in S_7 .

Quando il Clock è zero la macchina non varia né l'uscita né lo stato. Il Reset forza le uscite (sia quelle dello stato che quelle della macchina sequenziale) a zero. La tabella della verità che realizza la macchina combinatoria è:

b_2	b_1	b_0	b'_2	b'_1	b'_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Come per la progettazione di una qualunque delle macchine combinatorie, si procede anche in questo caso con la costruzione delle mappe di Karnough ed alla minimizzazione delle funzioni logiche:

$b_2 \backslash b_1$	00	01	11	10
0			1	1
1		1		1

$b_2 \backslash b_1$	00	01	11	10
0		1	1	
1	1	1		1

$b_2 \backslash b_1$	00	01	11	10
0	1	1	1	1
1				

$$b'_2 = b_2 \bar{b}_0 + b_2 \bar{b}_1 + \bar{b}_2 b_1 b_0$$

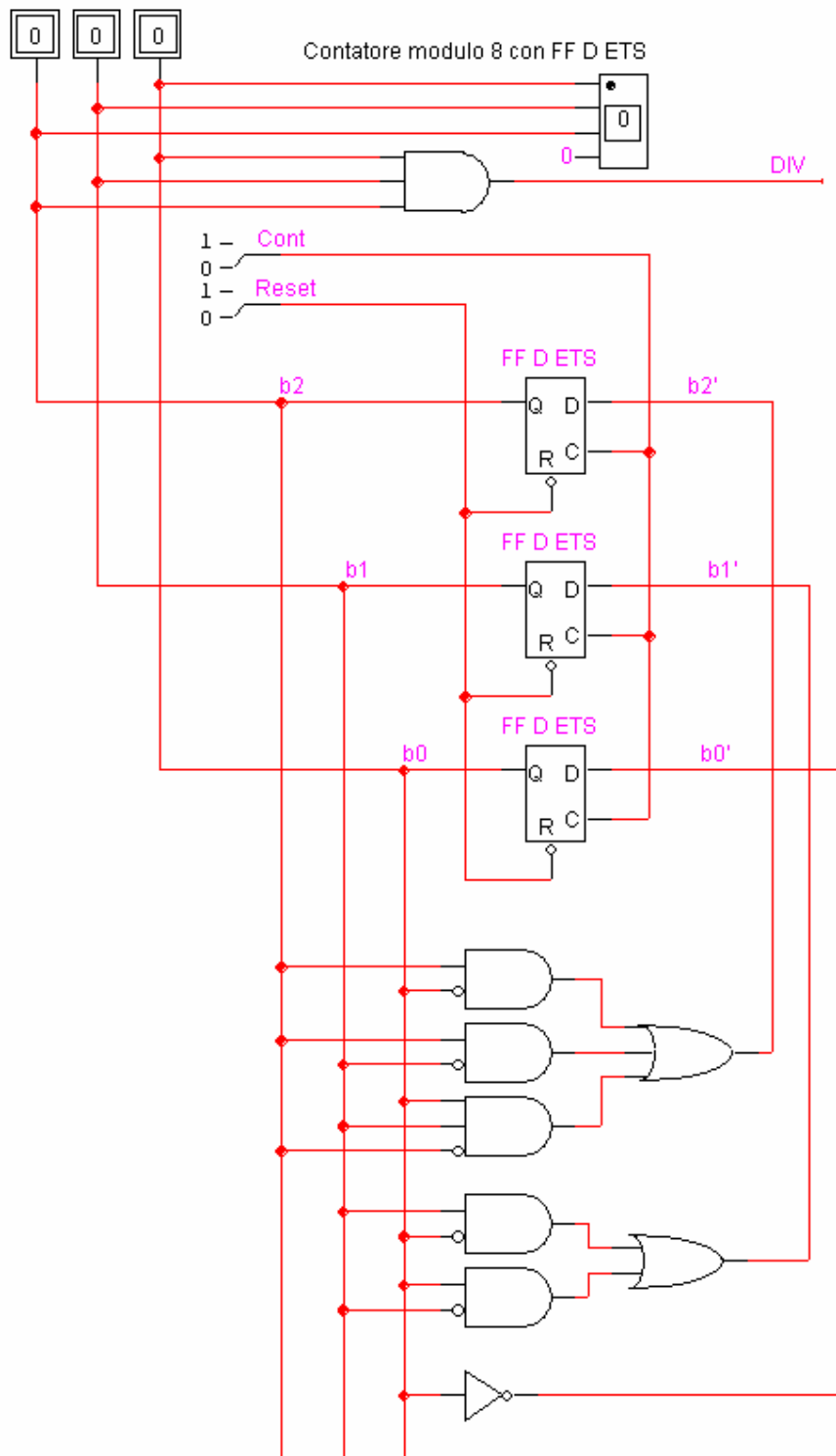
$$b'_1 = b_1 \bar{b}_0 + \bar{b}_1 b_0$$

$$b'_0 = \bar{b}_0$$

Per il DIV invece:

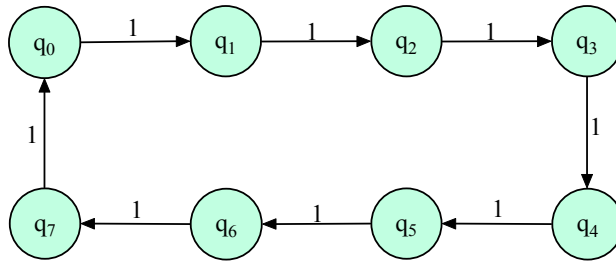
$$DIV = b_2 b_1 b_0$$

Circuito logico:



Esempio 2: Contatore modulo 8 con flip-flop T ETS

Il grafo visto in precedenza è ancora valido per cui:



anche la tabella di transizione e la codifica rimane immutata rispetto a quanto visto prima:

0	1	Div	Out
S ₀	S ₁	0	0
S ₁	S ₂	0	1
S ₂	S ₃	0	2
S ₃	S ₄	0	3
S ₄	S ₅	0	4
S ₅	S ₆	0	5
S ₆	S ₇	0	6
S ₇	S ₀	1	7

S₀ = 000

S₁ = 001

S₂ = 010

S₃ = 011

S₄ = 100

S₅ = 101

S₆ = 110

S₇ = 111

La scelta di un flip-flop T ETS, come detto in precedenza, influenza la codifica della tabella:

b ₂	b ₁	b ₀	b' ₂	b' ₁	b' ₀
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

Ricaviamo le mappe di Karnough:

b_2b_1 b_0	00	01	11	10
0				
1		1	1	

$b_2 b_1$ b_0	00	01	11	10
0				
1	1	1	1	1

	b_2b_1 00	01	11	10
b_0 0	1	1	1	1
1	1	1	1	1

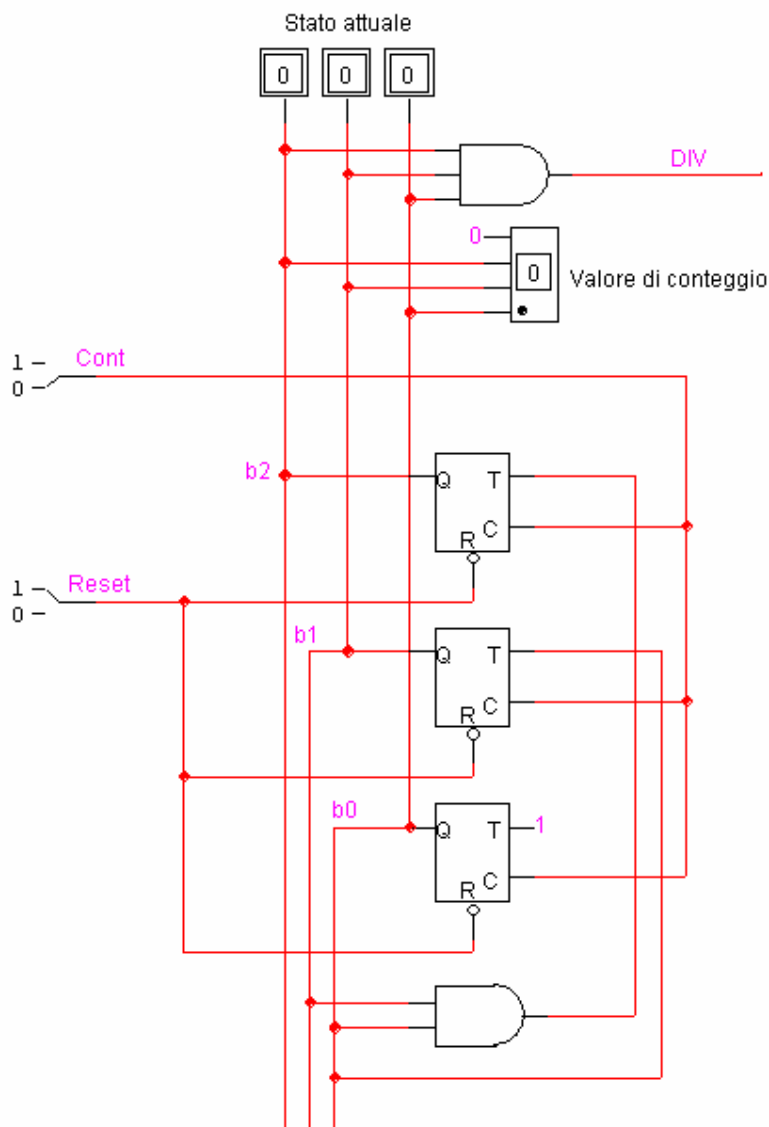
$$b_2' = b_1 b_0$$

$$b_1' = b_0$$

$$b_0' = 1$$

Circuito logico:

Contatore modulo 8 Ver.1.T



La progettazione dei contatori risulta più agevole se fatta scegliendo un flip-flop T oppure un flip-flop JK. Infatti, il flip-flop JK non è altro che un contatore modulo 2.

Il contatore modulo 8 appena progettato può essere realizzato a partire da alcune considerazioni fatte sulla tabella codificata:

T_2	T_1	T_0	T_2'	T_1'	T_0'	t_2	t_1	t_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1
Stato attuale			Stato prossimo					

In particolare, t_0 (bit meno significativo) cambia in continuazione; t_1 invece commuta quando t_0 è uguale ad 1; infine t_2 commuta quando t_1 e t_0 sono entrambi pari ad 1. Con queste considerazioni si perviene alla progettazione del contatore fatto in precedenza.

Esempio 3: Contatore modulo 8 bilaterale con FF D ETS

Tralasciando l'automa riportiamo di seguito la codifica degli stati e la tabella in forma codificata:

$S_0 = 000$
 $S_1 = 001$
 $S_2 = 010$
 $S_3 = 011$
 $S_4 = 100$
 $S_5 = 101$
 $S_6 = 110$
 $S_7 = 111$

Quando $U/D = 0$ il contatore effettua un conteggio a crescere, quando $U/D = 1$ il conteggio è invece a decrescere:

b_2	b_1	b_0	U/D	b_2'	b_1'	b_0'
0	0	0	0	0	0	1
0	0	0	1	1	1	1
0	0	1	0	0	1	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	0	0	1
0	1	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	0	1	1
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	0	1
1	1	1	0	0	0	0
1	1	1	1	1	1	0

Mappe di Karnaugh:

$b_0 \ U/D \backslash b_2 b_1$	00	01	11	10
00			1	1
01	1		1	
11			1	1
10		1		1

$$b_2' = b_2 \underline{b_1} \underline{U/D} + b_2 \underline{b_0} \underline{U/D} + b_2 b_1 \underline{b_0} + b_2 b_1 \underline{U/D} + b_2 \underline{b_0} \underline{U/D} + \underline{b_2} b_1 b_0 + \underline{b_2} b_1 \underline{b_0} \underline{U/D} + \underline{b_2} \underline{b_1} \underline{b_0} \underline{U/D}$$

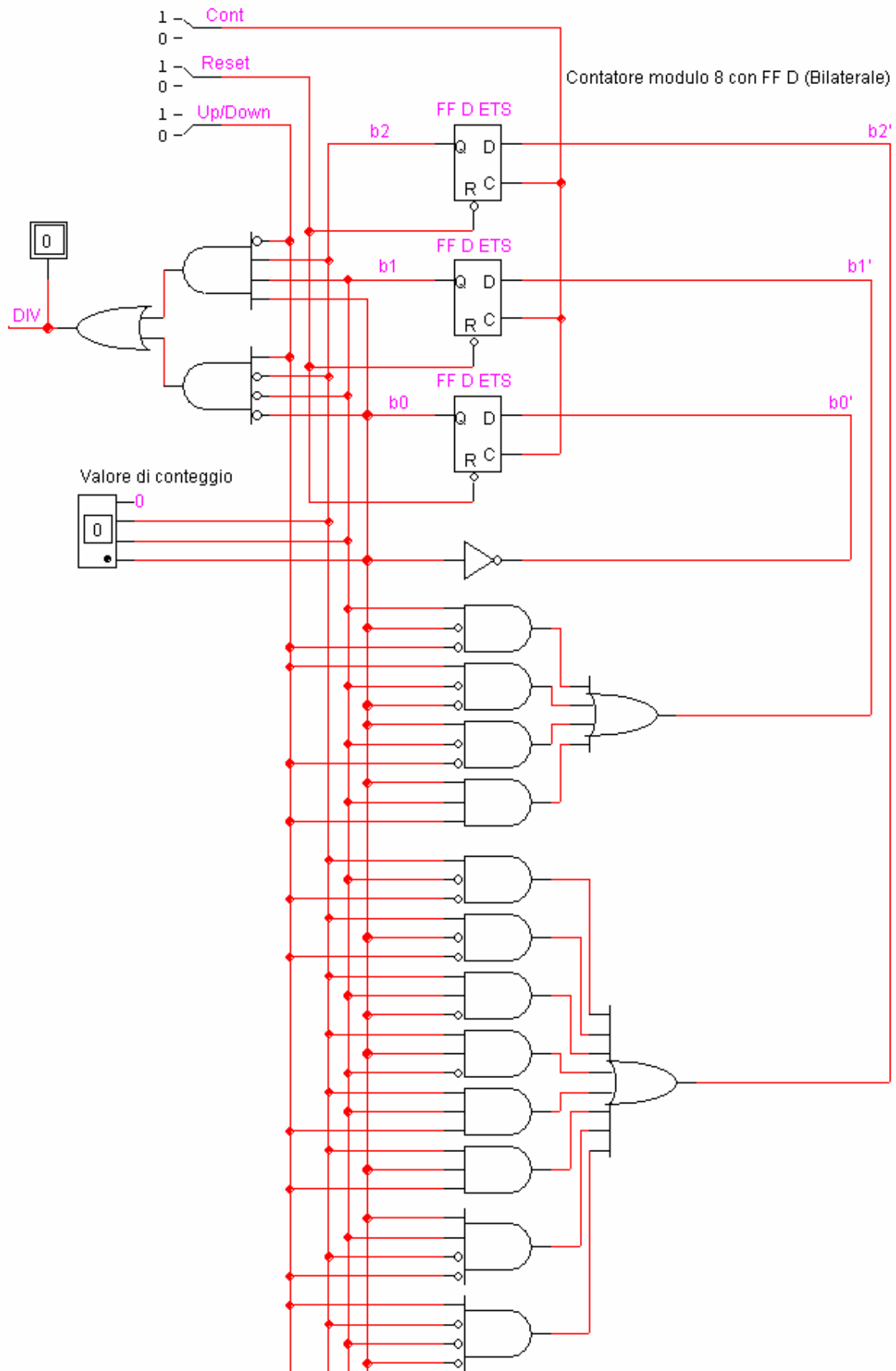
$b_0 \backslash b_1$	00	01	11	10
00		1	1	
01	1			1
11		1	1	
10	1			1

$$b_1' = b_1 \underline{b_0} \underline{U/D} + b_1 b_0 U/D + \underline{b_1} \underline{b_0} U/D + \underline{b_1} b_0 \underline{U/D}$$

$b_0 \backslash b_1$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11				
10				

$$b_0' = \underline{b_0}$$

Circuito:



Esempio 4: Contatore modulo 8 bilaterale con FF T ETS

Tralasciando l'automa riportiamo di seguito la codifica degli stati e la tabella in forma codificata:

$S_0 = 000$
 $S_1 = 001$
 $S_2 = 010$
 $S_3 = 011$
 $S_4 = 100$
 $S_5 = 101$
 $S_6 = 110$
 $S_7 = 111$

Quando $U/D = 0$ il contatore effettua un conteggio a crescere, quando $U/D = 1$ il conteggio è invece a decrescere:

b_2	b_1	b_0	U/D	b_2'	b_1'	b_0'
0	0	0	0	0	0	1
0	0	0	1	1	1	1
0	0	1	0	0	1	1
0	0	1	1	0	0	1
0	1	0	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
0	1	1	1	0	0	1
1	0	0	0	0	0	1
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	0	1	1	0	0	1
1	1	0	0	0	0	1
1	1	0	1	0	1	1
1	1	1	0	1	1	1
1	1	1	1	0	0	1

Mappe di Karnaugh:

$b_0 \backslash b_2 b_1$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$b_0' = 1$$

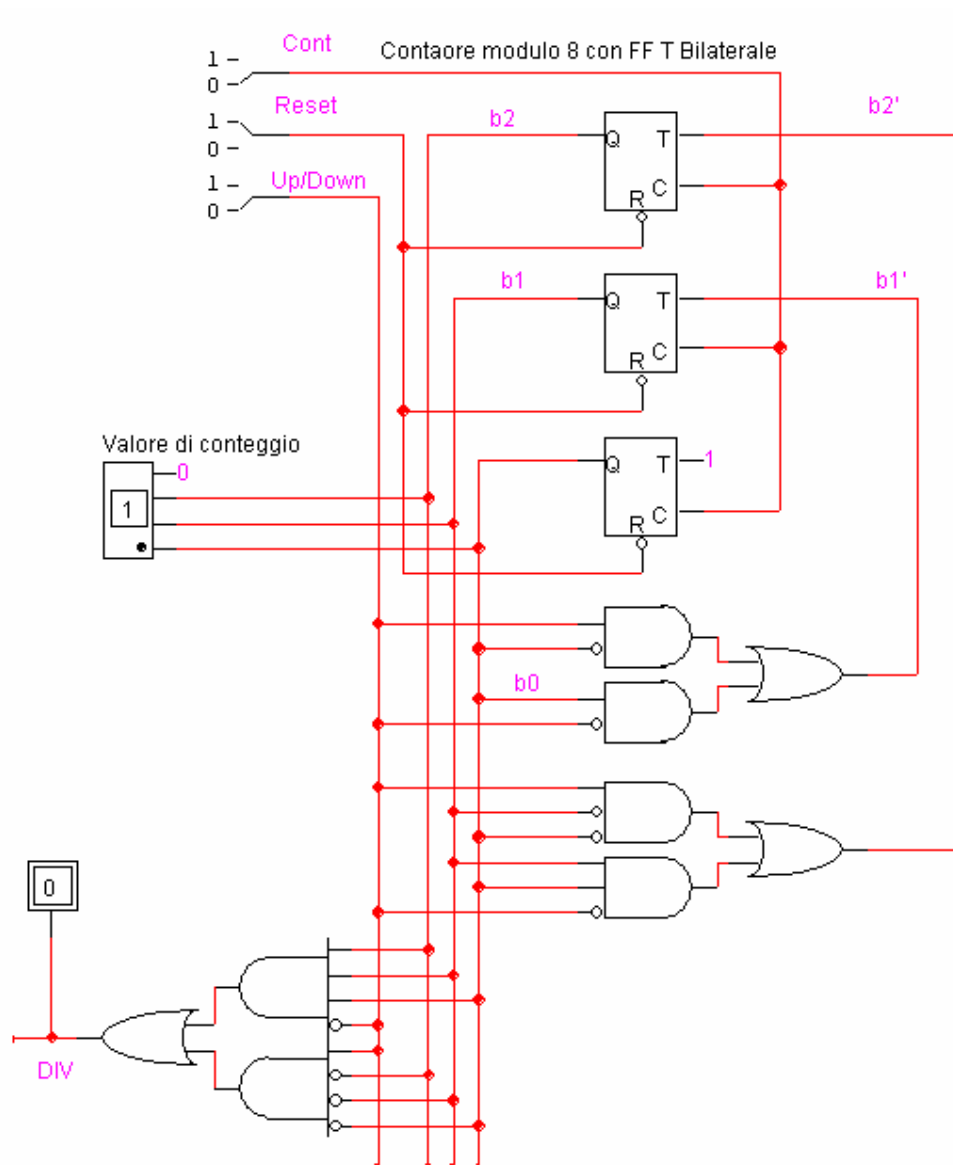
$b_0 \text{ U/D} \backslash b_2 b_1$	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$$b_1' = \underline{b_0} U / D + b_0 \underline{U} / D$$

$b_0 \text{ U/D} \backslash b_2 b_1$	00	01	11	10
00				
01	1			1
11				
10		1	1	

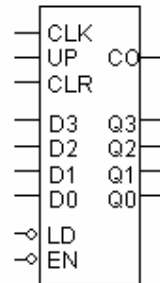
$$b_2' = \underline{b_1} \underline{b_0} U / D + b_1 b_0 \underline{U} / D$$

Circuito:



Il componente contatore in Logic Works

Nel simulatore di riferimento per queste note (Logic Works) così come in altri simulatori logici sono disponibili diversi contatori che ci evitano la progettazione di nuovi qualora in fase di progetto si dovesse riconoscere l'utilità di tale componente. Infatti, nella libreria di componenti (Simulation logic.clf) sono presenti contatori modulo 4 e modulo 8, in entrambi i casi esistono anche le versioni bilaterali di tali contatori e le versioni minime, dotate cioè di un numero minore di ingressi e di uscite.

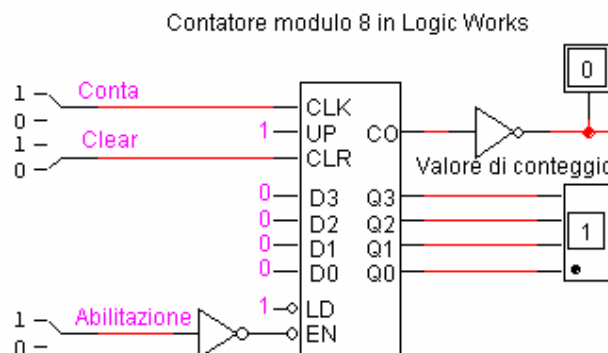


Quello in figura è un contatore modulo 16, capace cioè di effettuare un conteggio da 0 a 15. Nella parte a sinistra notiamo gli ingressi al componente.

In alto a sinistra (CLK) va inserito il valore di conteggio o clock, procedendo per ordine subito dopo notiamo il bit UP. Tale bit consente al contatore di effettuare due tipologie di conteggio, una modalità a crescere per UP = 1 ed una modalità a decrescere per UP = 0.

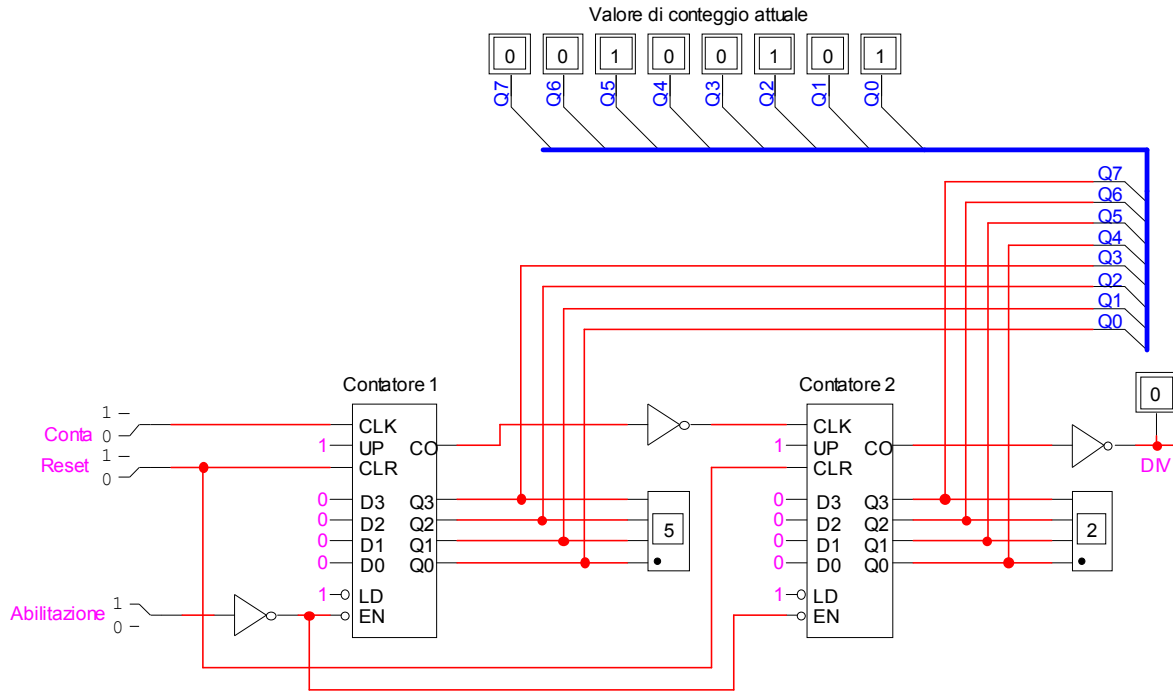
Il bit clear (CLR) effettua un reset del contatore conducendo quest'ultimo ad uno stato noto che è tipicamente il valore di conteggio iniziale.

Nei bit D3, D2, D1 e D0 è possibile settare un valore iniziale di conteggio se questo deve iniziare da tale valore. Il valore iniziale di conteggio viene caricato sul fronte di salita del bit load (LD). E' opportuno osservare che tale bit è noto in forma negata così come il bit di abilitazione (EN) che ha il compito di accendere o spegnere il contatore, arrestando quindi il conteggio in corso o facendolo riprendere:



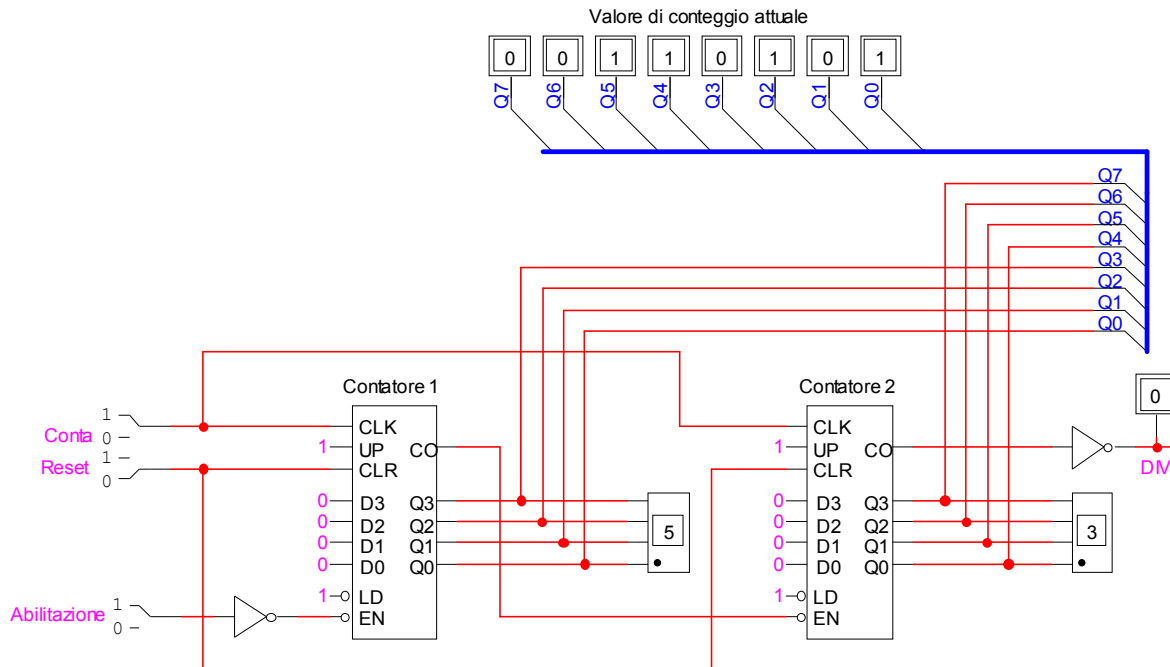
E' possibile effettuare una composizione di contatori per realizzarne uno con capacità estese, infatti, con due contatori

modulo 4 è possibile realizzare un contatore modulo 16 ($4 \times 4 = 16$). E' poi possibile effettuare una composizione asincrona in cui il valore di conteggio è dato ad uno solo dei contatori:

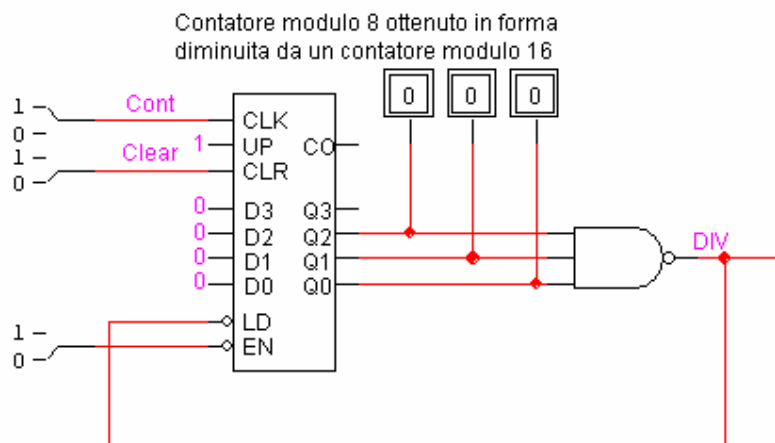


L'uscita DIV del primo contatore, quando è alta, fa incrementare di un unità il valore di conteggio del secondo contatore.

Una diversa composizione è poi possibile se si pensa di dare ad entrambi i contatori il valore di conteggio e abilitare il secondo contatore con l'uscita DIV del primo:



Infine, a partire da un contatore modulo N è possibile ottenere un contatore modulo K con $N > K$ facendo arrestare il conteggio al valore $K-1$ e caricando quindi nel contatore il valore di conteggio iniziale:



Lezione N°10: "La progettazione dei sistemi"

A volte la progettazione di una macchina risulta più agevole se fatta mediante la composizione di macchine già progettate in un precedente progetto, la composizione tra macchine può effettuarsi sia per le macchine combinatorie che per quelle sequenziali.

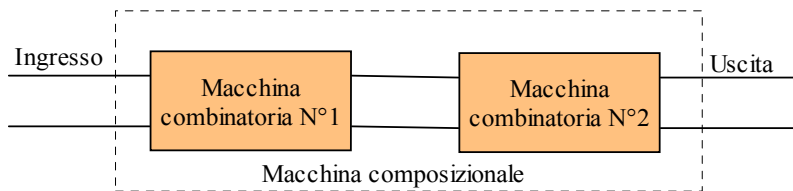
Le macchine usate per la composizione si dicono macchine componenti.

E' infatti utile un'attenta valutazione del numero di componenti che si andrà ad impiegare qualora il progetto sia effettuato per composizione, in altre parole è buona norma considerare la progettazione della macchina seguendo anche il suo tipico schema fin qui adottato e passando quindi per le varie fasi di progettazione (un'analisi quantitativa è opportuna per stabilire quale delle due strade intraprendere, ovviamente è preferibile una progettazione che impiega il minor numero di componenti possibili, mentre a parità di componente sarebbe più idonea una progettazione per composizione, in tal caso la scelta spetta comunque al progettista). Un parametro importante da non sottovalutare è la velocità di elaborazione della macchina, componendo più macchine i tempi di ritardo della macchina complessiva saranno influenzati dai rispettivi ritardi delle macchine componenti.

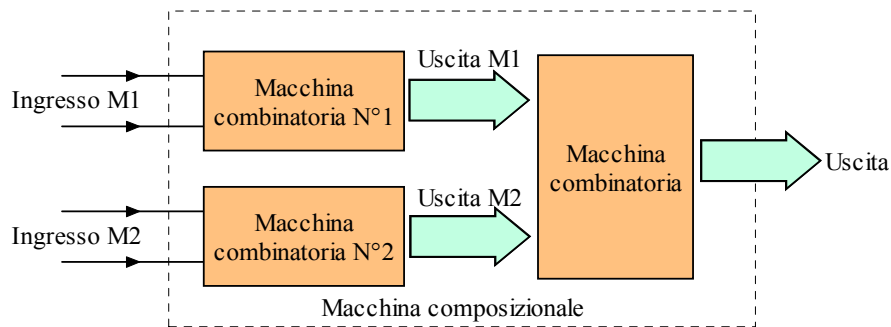
E' allora necessario fare un grosso sforzo iniziale per definire in modo chiaro gli obiettivi del progetto e le specifiche facendo attenzione alle limitazioni che vengono imposte al progettista. Allo stesso problema esistono spesso molte soluzioni.

Composizione di macchine combinatorie

La metodologia di composizione per le macchine combinatorie prevede due connessioni fondamentali: la composizione serie e la composizione parallelo. Nella composizione serie le macchine vengono poste una dietro l'altra e l'uscita di una è l'ingresso della successiva, in particolare per due sole macchine combinatorie la connessione serie prevede il seguente schema:

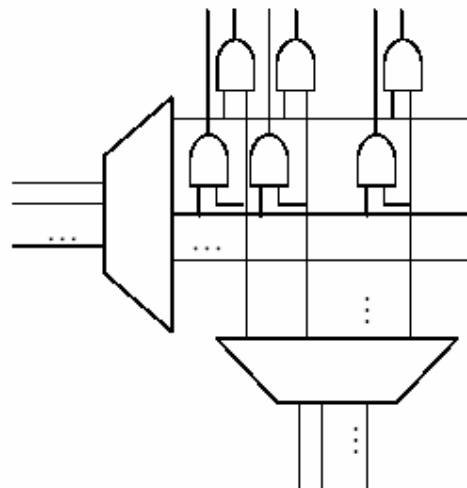


La connessione duale alla connessione serie è invece la connessione parallelo in cui le macchine operano in modo indipendente e producono ciascuna una parte dell'ingresso. Nella composizione parallelo di macchine combinatorie è spesso necessaria una ricombinazione delle uscite, ciò richiede la progettazione di un'altra macchina combinatoria che presi in ingresso le uscite delle macchine componenti produca le uscite della macchina composizionale. Con riferimento a due sole macchine combinatorie, la connessione parallelo di macchine combinatorie prevede il seguente schema:



Composizione per semiselezione di decodificatori

I decodificatori costituiscono insieme ai multiplexer le macchine combinatorie più usate in fase di progettazione. A volte può capitare di effettuare una composizione anche per queste macchine. Nella configurazione per semiselezione si utilizzano 2 decodificatori di metà ampiezza ed una porta AND per ciascuna uscita. Ogni uscita del decoder è intrecciata con tutte le altre uscite dell'altro decoder mediante una porta AND. Se ad esempio facciamo riferimento a 2 decodificatori da 2 ingressi e 4 uscite, mediante questo tipo di composizione si realizzerà un nuovo decodificatore da 4 ingressi e 16 uscite come mostrato in figura:



Composizione ad albero di decodificatori

La composizione di decodificatori può avvenire anche secondo una struttura nota come **struttura ad albero** dove per realizzare un decodificatore da n uscite e $\log_2 n$ ingressi, utilizzando decodificatori da:

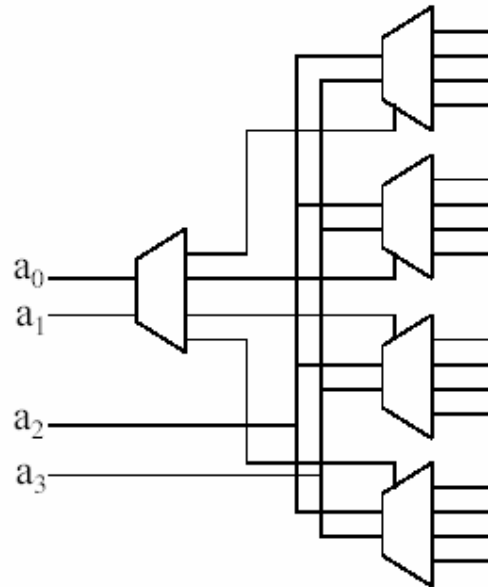
- m uscite;
- $\log_2 m$ ingressi;

Si impiegano un numero di decoder pari ad:

$$- 1 + m + m^2 + m^k;$$

Dove k identifica il numero di livelli dell'albero.

Ad esempio:



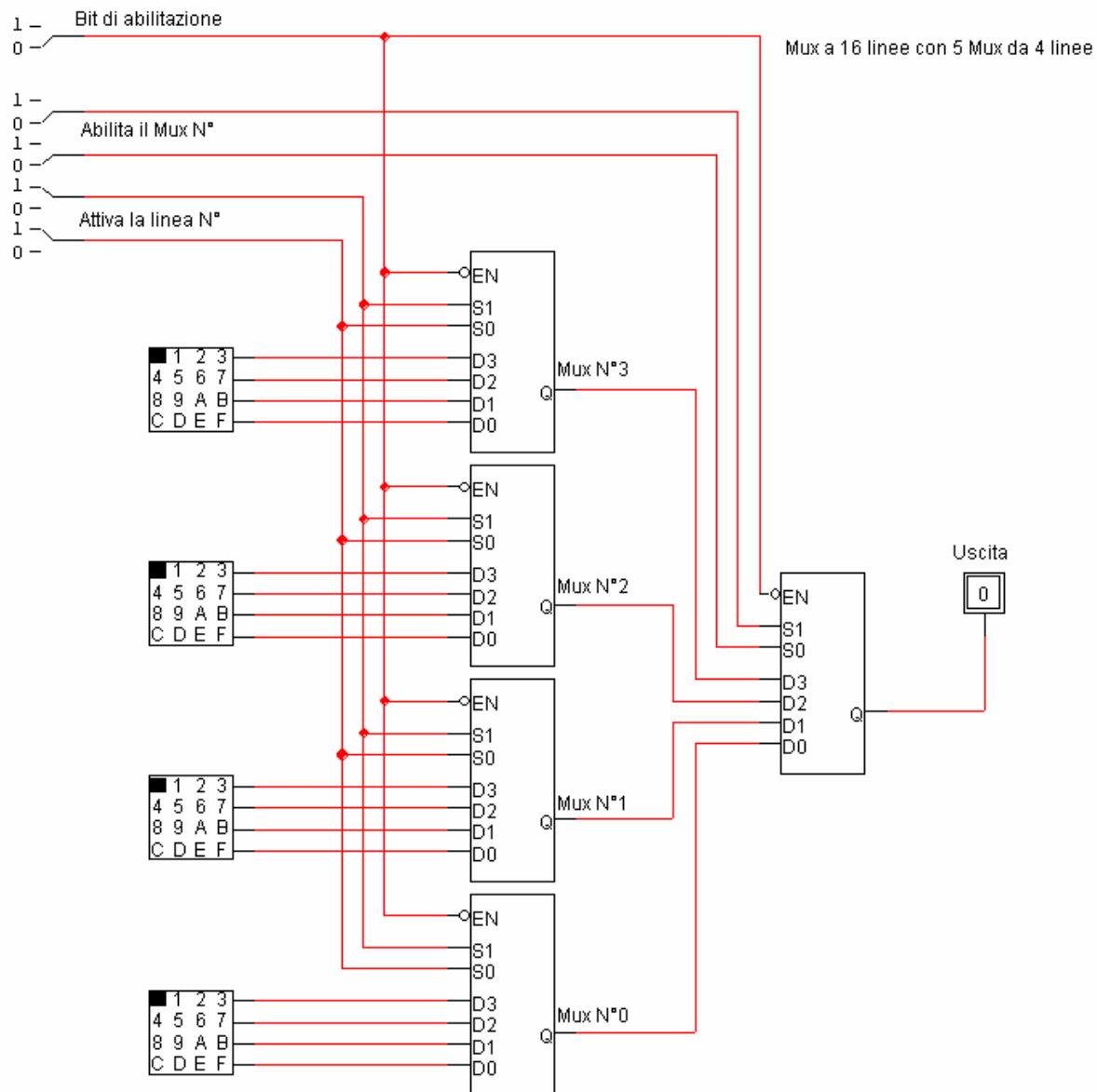
Nella combinazione di più decoder secondo la struttura ad albero, come è mostrato in figura, le prime linee sono utili a selezionare il decoder da attivare (in figura a_0 ed a_1 sono le linee che ci permettono di selezionare uno dei quattro decoder), mentre le restanti linee servono invece a selezionare una delle linee in uscita del decoder attivato (in figura a_2 ed a_3 permettono di selezionare le linee in uscita che vanno da 0 a 3).

Composizione di multiplexer

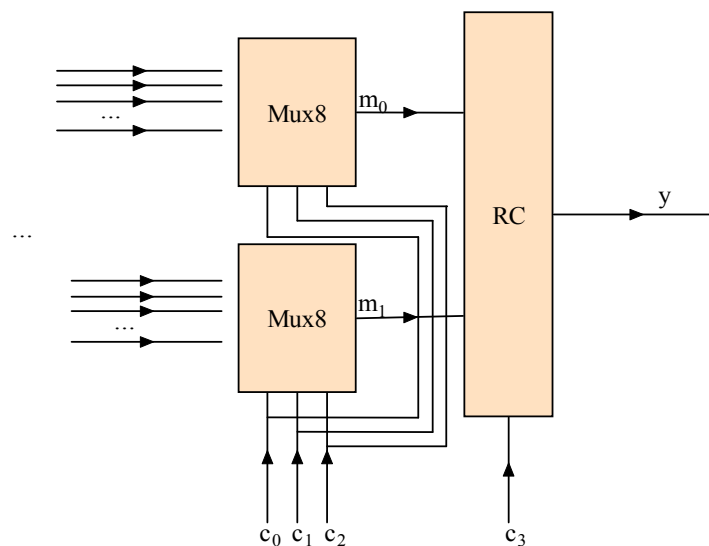
Altra macchina combinatoria largamente usata è il multiplexer. Le considerazioni fatte per le macchine combinatorie a proposito della progettazione si estendono anche per questa macchina.

E' possibile realizzare un multiplexer da 16 ingressi con 5 multiplexer da 4 ingressi, adottando una struttura ad albero. (in realtà lo schema si presenta come un albero capovolto e può essere visto anche come un ramo dalle tante ramificazioni).

Due degli stati di selezione permettono di scegliere il multiplexer da leggere o da abilitare, con 2 bit possiamo abilitare 4 multiplexer le cui uscite sono raccolte in ingresso da un altro multiplexer. Altri due bit permettono per ciascun multiplexer di selezionare una delle sue quattro linee come ingresso:



Oppure, è possibile realizzare un Multiplexer composto a 16 ingressi con 2 multiplexer da 8. In tal caso è opportuno realizzare una rete combinatoria che ri-elabora l'uscita.



La rete combinatoria avrà 3 ingressi con:

- m_0 , m_1 e c_3 ;
- 1 bit di uscita y ;

tale che:

- $y=m_0$ se $c_3=0$;
- $y=m_1$ se $c_3=1$;

Risulta:

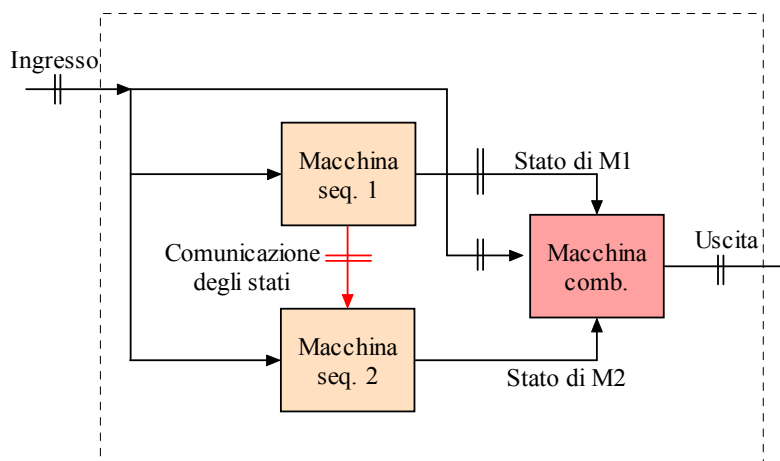
$$y = m_0 \cdot \underline{c_3} + m_1 \cdot c_3$$

Una volta che un dispositivo sia stato costruito, esso può fungere da componente di sistemi più complessi o comunque da "base" per una sua evoluzione verso dispositivi più sofisticati.

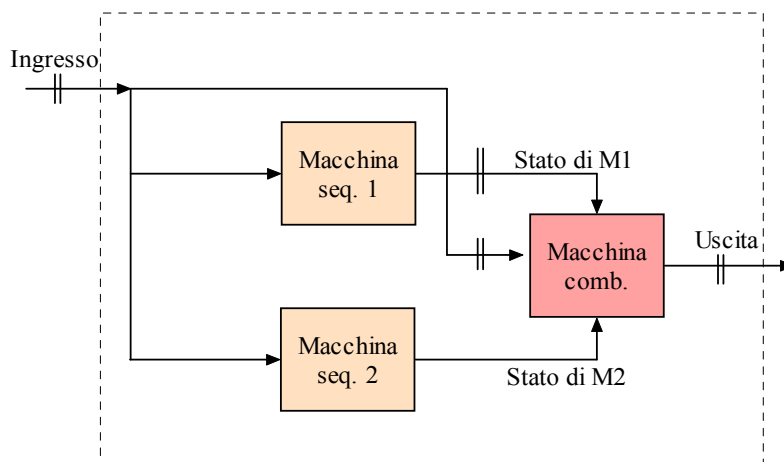
Nella connessione in parallelo è consentita l'esecuzione contemporanea di più operazioni. Le linee di ingresso vengono fornite a tutte le macchine connesse in parallelo ed i tempi di elaborazione dipendono soltanto dalla macchina più lenta, a differenza della connessione in serie in cui i tempi di elaborazione sono più alti in quanto il tempo di elaborazione è la somma dei tempi di elaborazione di tutte le singole macchine.

Composizione di macchine sequenziali

Una macchina sequenziale può essere realizzata mediante apposito collegamento di due o più macchine sequenziali, con particolare riferimento a due sole macchine sequenziali è allora possibile realizzare una composizione serie ed una parallela. Nella **connessione serie** lo stato di una viene comunicata all'altra, infine gli stati delle due macchine vengono composti per ricavarne le uscite, quello che segue è uno schema tipico di una connessione serie tra due macchine sequenziali:



Nella **connessione parallela**, non c'è scambio di stato tra le due macchine, ma vi è soltanto la composizione finale degli stati (notare infatti l'assenza della linea di connessione che permette la comunicazione degli stati).



Riconoscitore 8421 mediante composizione

La progettazione di un riconoscitore 8421 è stata già affrontata quando abbiamo introdotto la progettazione delle macchine sequenziali. La progettazione che effettueremo sarà basata sulla metodologia di composizione delle macchine, in particolare si farà uso di un contatore modulo 4.

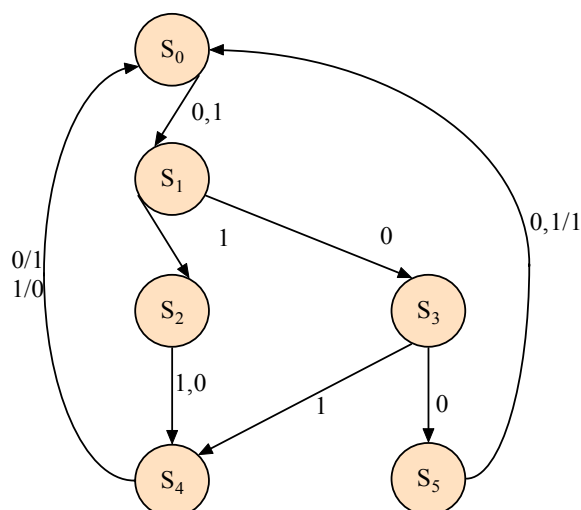
Un riconoscitore 8421 è una macchina che riceve in ingresso un flusso di bit ed emette un segnale alto ogni 4 bit, se questi corrispondono ad un numero valido della sequenza 0,1,2,3...9. I valori di input sono:

- le sequenze di bit, gli ingressi possono essere 0 o 1;

I valori di output sono:

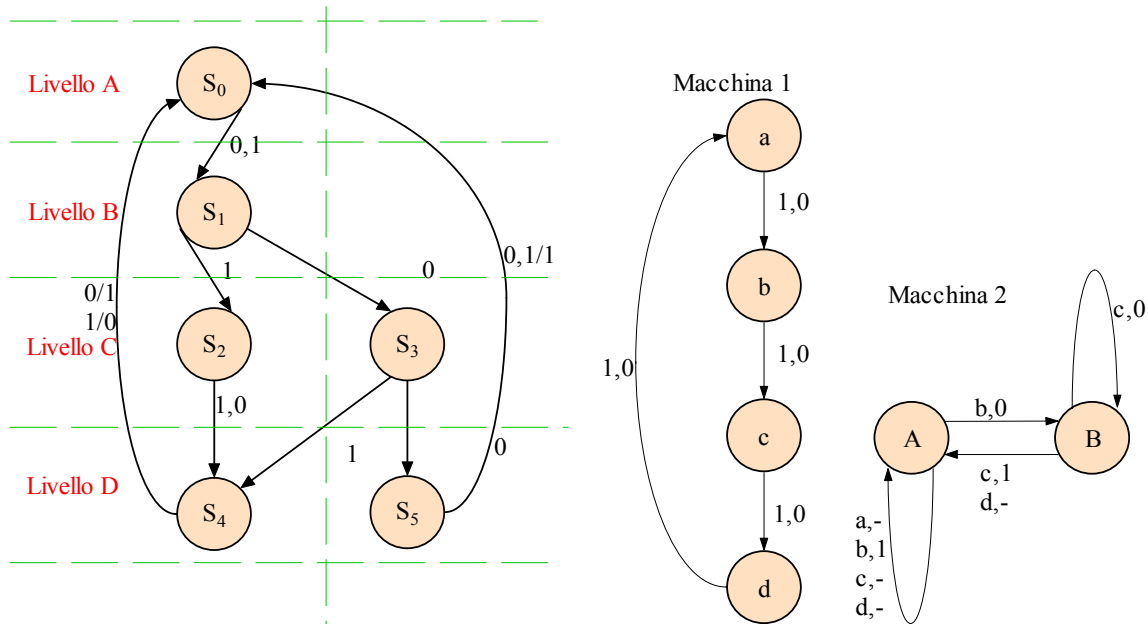
- 0 se ho un valore valido o una sequenza incompleta;
- 1 se ho un valore valido nella codifica 8421;

La progettazione dell'automa è la stessa di quella già affrontata in alcune lezioni precedenti, per comodità riportiamo di seguito il grafo dell'automa e la sua tabella di transizione per gli stati:



STATO \ I	0	1
S ₀	S ₁	S ₁
S ₁	S ₂	S ₃
S ₂	S ₄	S ₄
S ₃	S ₅	S ₄
S ₄	S ₀ /1	S ₀
S ₅	S ₀ /1	S ₀ /1

L'automa può essere visto, mediante una linea separatrice immaginaria, come uno spaccato di due blocchi di macchina, lo spaccato a sinistra sarà denotato con la lettera A e lo spaccato a destra con la lettera B. La parte A dell'automa (quella a sinistra) si compone degli stati S_0, S_1, S_2 ed S_4 mentre la parte B (quella a destra) si compone degli stati S_3 ed S_5 . La prima parte procede in maniera autonoma mentre la seconda evolve in funzione della prima.

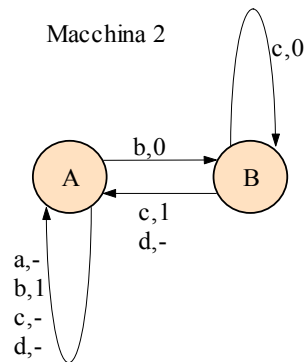


Per realizzare l'automa si dovranno allora realizzare due macchine: la macchina 1 e la macchina 2. La macchina 1 ha un comportamento noto, essa è infatti riconducibile ad un contatore modulo 4, ad ogni passo la macchina cambia stato passando in quello successivo, infine dall'ultimo stato passa al primo come un comune contatore.

La macchina 1 permette all'automa di spostarsi in verticale ed occupare quindi uno degli stati della macchina (livelli). La macchina 2 invece permette all'automa di spostarsi in orizzontale, il suo comportamento non è riconducibile a macchine note e deve pertanto essere elaborato con i metodi tradizionali. Gli ingressi della macchina sono:

- lo stato della macchina 1;
- l'ingresso della macchina completa;

L'uscita è lo stato. L'automa che realizza la macchina 2 è



La lettura dell'automa è il seguente, occorre ricordare che A è la parte di macchina a sinistra mentre B è la parte di macchina a destra. Ci troviamo in A (parte o macchina sinistra), proseguendo per i livelli a,b,c e d si considerano i diversi possibili ingressi: con gli ingressi **a,1** ed **a,0** la macchina procede in verticale rimanendo nella parte sinistra dell'automa; per l'ingresso **b,1** la macchina procede in verticale e rimane quindi in A mentre per **b,0** la macchina si sposta a destra portandosi in B; per gli ingressi **c,0** e **c,1** la macchina si porta in uno stato posto a sinistra dell'automa rimanendo quindi in A; per l'ingresso **d,1** la macchina si porta in uno stato a sinistra della macchina e quindi continua rimanere in A, per l'ingresso **d,0** la macchina si porta sempre in a (stato posto a sinistra dell'automa) ma questa volta con un valore dell'uscita alto. Supponiamo adesso il caso in cui ci si trova in B, ossia in uno stato a destra dell'automa: per gli ingressi dei livelli a e b (quindi **a,0** **a,1** **b,0** ed **b,1**) la parte B non è interessata (don't care); per l'ingresso c,1 la macchina si porta in A mentre per l'ingresso c,0 la macchina si porta in B; per gli ingressi d,1 e d,0 la macchina si porta in A con uscita 1. La tabella degli stati che riassume quanto detto è:

M1,1/M2	a,1	a,0	b,1	b,0	c,1	c,0	d,1	d,0
A	A	A	A	B	A	A	A	A/1
B	-	-	-	-	A	B	A/1	A/1

Occorre allora svolgere il progetto della macchina 2 che ha due soli stati: essa entra nello stato B solo quando la macchina 1 è nello stato B e l'ingresso è 0; ritorna poi in A se la macchina 1 è in C e l'ingresso è 1 e tutte le volte che l'ingresso è 1.

Gli stati vengono così codificati:

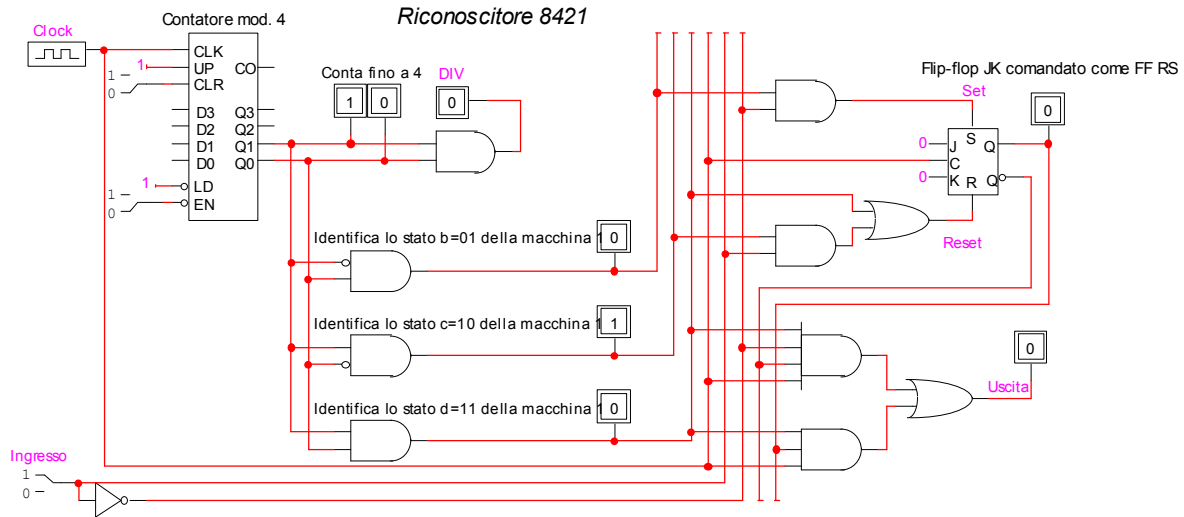
- A = 0;
- B = 1;

ed i valori che permettono di scegliere la macchine A o quella B (che chiamiamo valori set e di reset) sono:

- Set = $b\bar{1}$;
- Reset = $c1 + d$;

Il tipo di composizione che andiamo ad effettuare è di tipo seriale, la macchina M2 evolve in funzione di M1 poiché ha come ingresso proprio uno stato di M1. Per l'uscita: l'uscita è alta quando la macchina 1 è nello stato d e l'ingresso è 0, oppure quando la macchina 2 è nello stato d e l'ingresso è 0 od 1:

$$z = dI + d$$



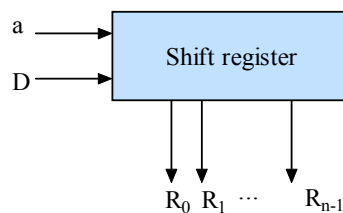
Lezione N°11: "I registri a scorrimento"

Shift register

Oltre a svolgere le proprie funzioni, i registri a scorrimento vengono usati nella composizione di macchine più complesse. Un registro a scorrimento è una macchina sequenziale a sincronizzazione esterna composta da n flip-flop e che in presenza di un segnale di abilitazione acquisisce serialmente un dato D nella prima posizione a sinistra del registro, al successivo ingresso il nuovo bit prende il posto più sinistra del registro (quello cioè occupato in precedenza dal primo bit letto) mentre quello che occupava la prima posizione viene spostato in avanti (shift). La macchina ha pertanto due ingressi:

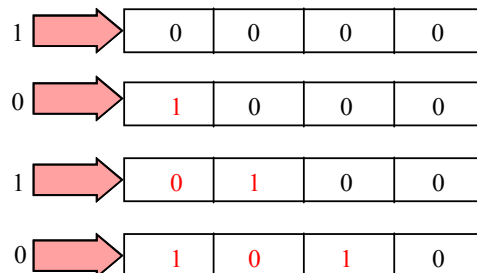
- il segnale di abilitazione, bit a ;
- il bit dato D ;

mentre le uscite sono n e rappresentano lo stato attuale;

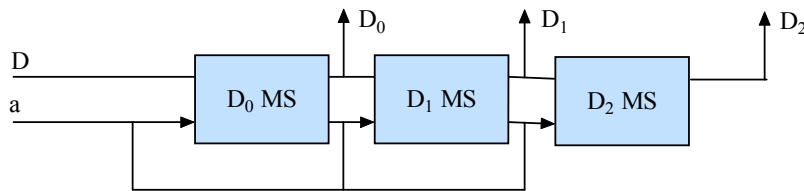


Ad ogni colpo di clock il dato D in ingresso è letto e messo nella posizione più a sinistra del registro, se il registro si compone di n celle per la memorizzazione del bit, cosa succede all'ultimo bit quando il registro è pieno?

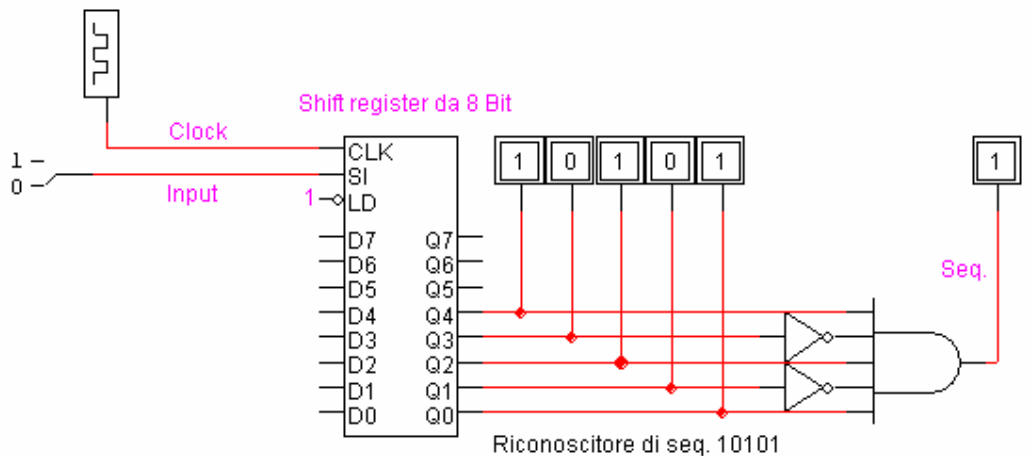
Esistono due tipi di registri, in alcuni registri a scorrimento quando la posizione $n-1$ è già occupata da un bit questi eliminano tale bit al successivo colpo di clock, altri registri a scorrimento invece riportano tale bit nuovamente in testa al registro (posizione più a sinistra) compiendo così uno scorrimento ciclico dei bit in esso contenuti. Ecco cosa succede ad esempio in un registro a scorrimento dotato di quattro celle per la memorizzazione del dato D :



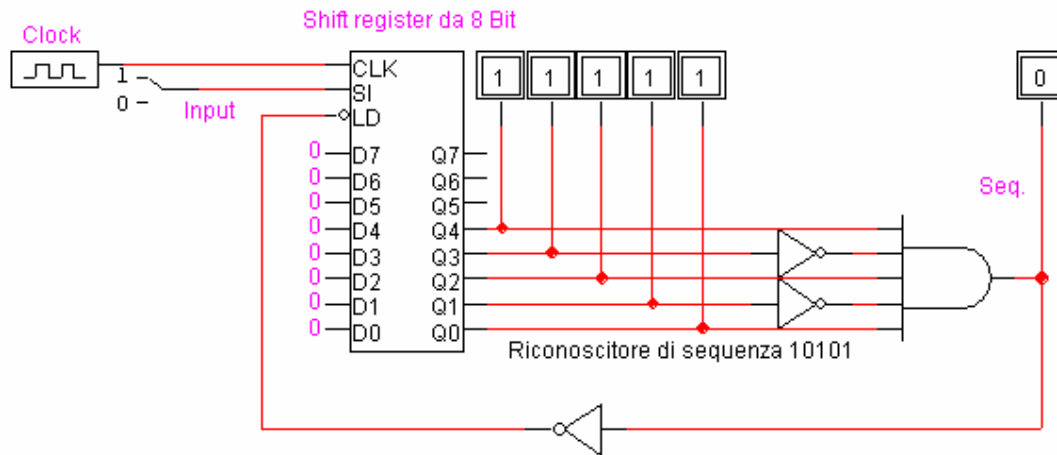
La macchina può essere realizzata ponendo n flip-flop in serie, il flip-flop più comodo da usare è quello di tipo D , mentre la tempificazione **deve essere master-slave oppure edge-trigger**:



Un uso pratico dei registri a scorrimento sono i riconoscitori di sequenze, si realizza ad esempio il riconoscitore di sequenza per la stringa 10101 usando un registro a scorrimento ad 8 bit. La realizzazione di un riconoscitore di sequenza mediante uso di shift register è estremamente semplice come si può vedere dal circuito:



Come si può notare lo shift register oltre ad avere 8 uscite è dotato anche di 8 ingressi per caricare un valore in memoria, il bit LD (bit di load) carica il valore posto in ingresso. Il riconoscitore di sequenza appena visto è inoltre capace di riconoscere le sequenze valide anche se sovrapposte. Questo avviene poiché quando il registro riconosce la sequenza non effettua il reset e continua nella sua lettura. Per realizzare un riconoscitore di sequenza che non consideri le possibili sovrapposizioni è opportuno collocare il bit di uscita al bit di reset, non appena la sequenza è riconosciuta, e quindi l'uscita è alta, tale bit (che abbiamo posto in ingresso al bit di reset) realizza per noi la condizione di reset per il registro:



La rete di parità:

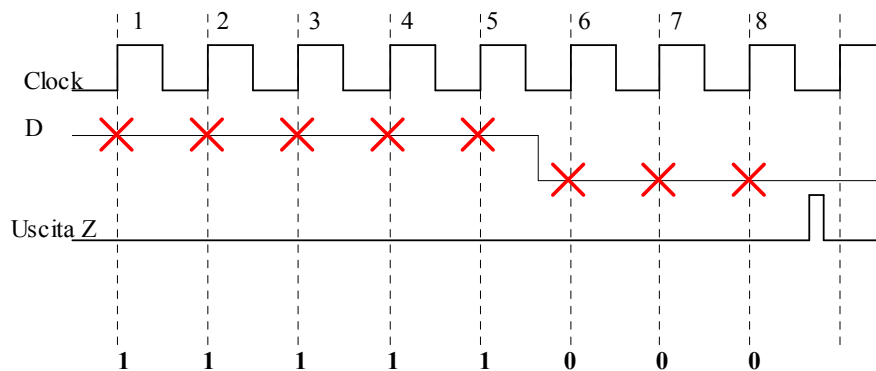
Progettare una macchina caratterizzata da:

- due ingressi (clock C e bit dati In);
- un uscita Z' di tipo impulsivo;

tale che:

- ogni 8 colpi di clock, se il numero di bit alti ricevuti è dispari, z' mostra un impulso;

Ecco una possibile tempificazione per la macchina:



Notare nel disegno delle tempificazioni che dopo 8 colpi di clock (corrispondenti ad otto gradini di salita e/o discesa) sul bit z si verifica l'impulso indicante appunto che il numero di bit alti è dispari. Il bit D è letto ad ogni colpo di clock (notare i riferimenti nel grafico, ad ogni croce corrisponde il bit letto). L'automa deve tener conto di:

- quanti bit sono arrivati;
- se i bit alti arrivati fino ad ora sono pari o dispari;

I bit sono letti sui fronti di salita del clock, l'impulso che realizza l'uscita della macchina avviene tra un colpo di clock ed il successivo, ovviamente esso si presenterà non subito (a causa

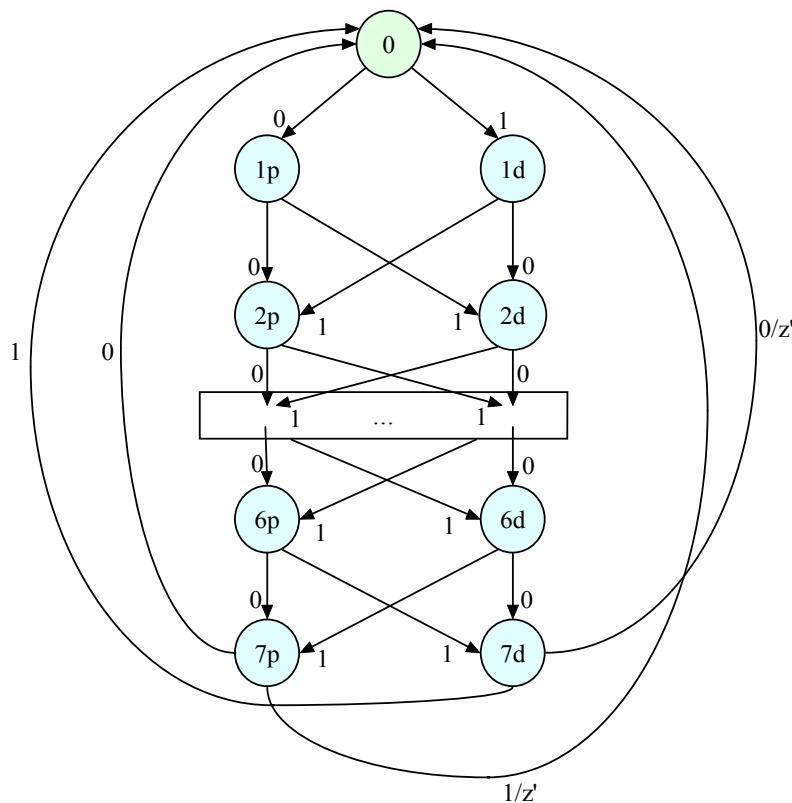
dei ritardi) per cui è ragionevole rappresentarlo nel mezzo dei due colpi di clock.

Nella realizzazione di questa macchina ci accorgiamo di come le specifiche siano del tutto generali, ciò ci permette di scegliere alcuni parametri. Ad esempio la durata del clock che non è specificata e il fronte di salita su cui leggere i bit D.

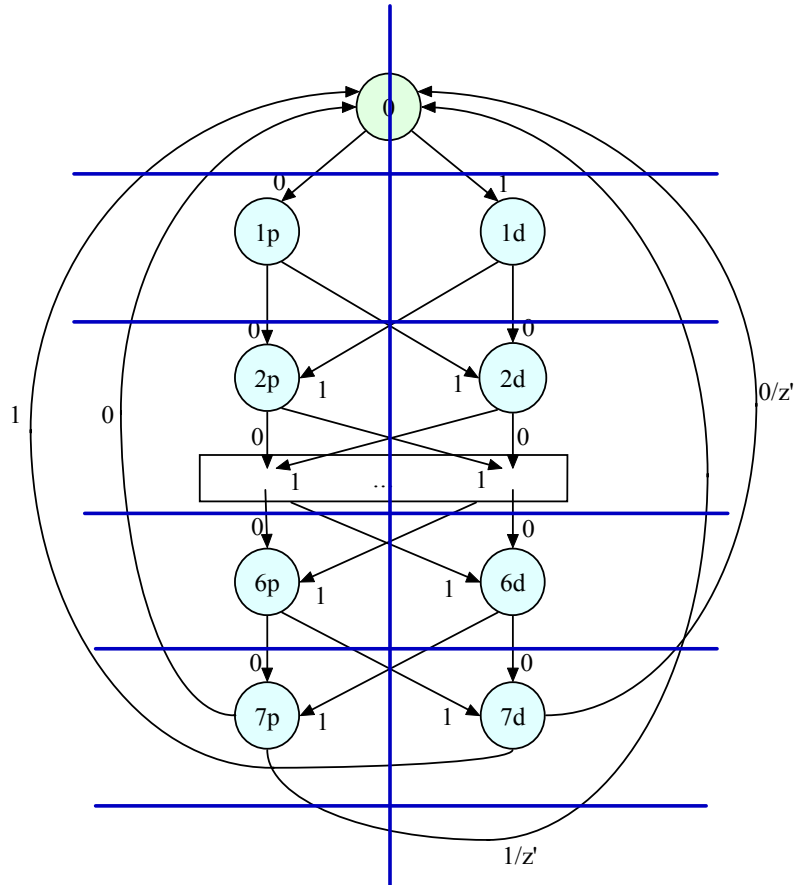
Dovendo realizzare un uscita impulsiva associamo tale uscita all'arco e non allo stato. Infatti, essendo l'arco una transazione si realizzerà z' impulsivo in questo modo. Tipicamente un uscita a livelli è associata allo stato e un uscita impulsiva è data invece da una transazione della macchina.

E' inoltre possibile un accostamento di questo tipo, l'automa di Mealy realizza uscite di tipo impulsivo mentre l'automa di Moore realizza uscite a livelli.

Nel nostro caso, l'uscita impulsiva è ottenuta realizzando l'automa secondo Mealy.



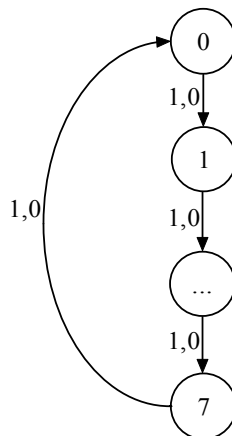
L'automa viene decomposto:



individuando così:

- livelli che ad ogni ingresso l'automa deve cambiare per memorizzare il numero di bit arrivati;
- stati, ad ogni livello l'automa deve memorizzare se ha ricevuto un numero pari o dispari di bit;

Ad ogni passo la macchina cambia stato passando così al successivo, inoltre dall'ultimo stato passa al primo, questo comportamento lascia intendere che una macchina componente sarà composta da un contatore modulo 8.

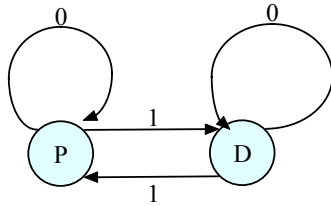


La macchina 2 è dotata di:

- un ingresso della macchina completa;

e di:

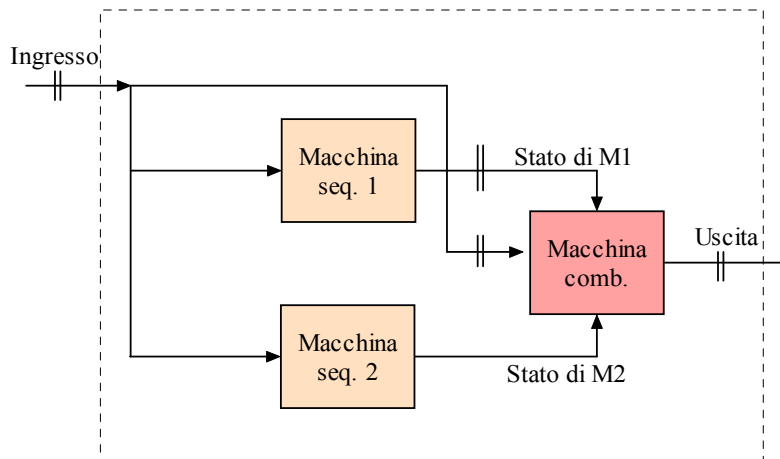
- un uscita, che è lo stato;



M2\I	0	1
0	0	1
1	1	0

(P=0, D=1) la macchina 2 è un flip-flop T.

La macchina composta che andremo a realizzare è ottenuta mediante una composizione parallela, nel circuito si noterà l'assenza di scambio di stato tra le due macchine. Lo schema da seguire è quello già mostrato nelle precedenti lezioni:



Rimane da progettare la macchina combinatoria per realizzare l'uscita.

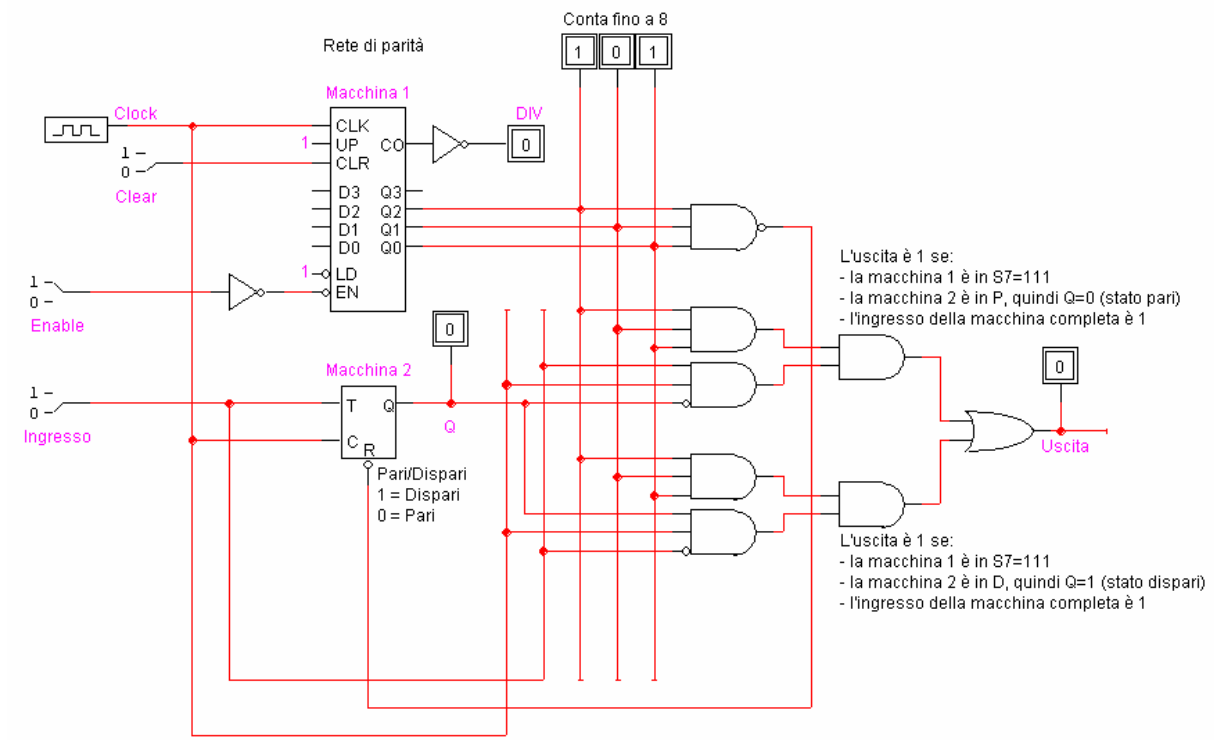
L'uscita è alta quando:

- la macchina 1 (il contatore mod. 8) si trova in S_7 , la macchina 2 si trova in P (pari) e l'ingresso della macchina completa è 1;

oppure

- la macchina 1 si trova in S_7 (ultimo stato per il contatore), la macchina 2 si trova in D (dispari) e l'ingresso della macchina completa è 0.

Il circuito:



Lezione N°12: "Il progetto di macchine per composizione"

Dal libro teoria e progetto delle reti logiche:

Con il termine sistema si vuole intendere un sistema di reti logiche, cioè un sistema di reti tra loro opportunamente interconnessi, tali da implementare una macchina che, pur se raggiunge un certo livello di sofisticazione, è pur sempre una unità componente di un sistema più ampio. La scomposizione della macchina in più reti componenti o sottoinsiemi è utile ai fini del progetto per diversi motivi:

- rappresenta un processo logico naturale: un sistema complesso si suddivide in parti componenti per poterlo meglio capire e controllare; analoghi concetti si applicano nella produzione di software;
- i metodi di progetto di macchine a molti stati sono complessi;
- il progetto modulare che ne deriva semplifica l'analisi del funzionamento, l'individuazione di guasti, l'allocazione su circuiti stampati;
- è possibile l'impiego di componenti standard esistenti in commercio;

Pertanto il metodo più diffuso per affrontare un problema di sintesi di un sistema è quello di decomporre il sistema stesso in sottosistemi ed eventualmente iterare il procedimento fino a realizzare tante reti molto semplici, per le quali è facile effettuare individualmente una sintesi ottimale, vediamo subito un primo esempio:

Esempio di progettazione di macchina per composizione

Problema: Progettare una macchina attraverso un approccio compositivo, che data una sequenza di bit in ingresso (**In**) tempificata tramite un segnale di sincronizzazione (**Sinc**), conti se il numero di sequenze **010** giunte tra una sequenza **111** ed una sequenza **000** è pari, emettendo due segnali, uno per indicare che sono stati emessi sia i segnali di start che di fine (**seq** di tipo impulsivo), ed uno che indichi la parità (**P**) Ad esempio alla fine della sequenza il dispositivo deve emettere:

- Seq=1;
- P=0;
- Per la sequenza 1010**111**001101011010010011**000** ;
(inizio) (fine)

Indicare i dispositivi utilizzati, il loro comportamento ed i problemi di tempificazione che possono sorgere. Mostrare una tempificazione completa come esempio di evoluzione della macchina con tutti i segnali.

Una volta che il comportamento della macchina è stato individuato (ciò può essere fatto ipotizzando una possibile tempificazione per la macchina e rileggendo la traccia del problema più volte in modo tale da comprendere il comportamento della macchina sotto certi ingressi) è possibile passare alla fase di progetto in cui occorre individuare i componenti o le macchine componenti per la macchina. Nel nostro caso, saranno macchine componenti per il progetto:

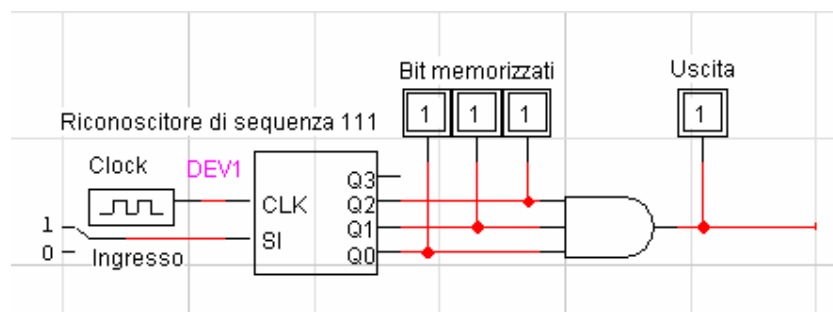
- un riconoscitore di sequenza 111 che decreta l'inizio del conteggio per la macchina;
- un riconoscitore di sequenza 000 che stabilisce la fine del conteggio;
- un riconoscitore di sequenza 010 utile a contare la sequenza assunta come riferimento;
- Rete di parità seriale per verificare la parità o la disparità della sequenza contata;
- Circuito On/Off per attivare e disattivare il dispositivo in base ai segnali riconosciuti in ingresso;

La prima considerazione da fare è legata alla necessità di riconoscere tre differenti sequenze: 111, 000 ed 010. Ci sono molti modi di realizzare tali macchine, quello scelto è basato sull'utilizzo di un registro a scorrimento (shift register) e di semplici macchine combinatorie (porte AND e NOT) per riconoscere la sequenza corretta. I riconoscitori di sequenza che si andranno a progettare avranno la capacità di riconoscere anche le sequenze sovrapposte, le specifiche del problema non esplicitano questa necessità, tuttavia decidiamo comunque di accettare anche le sequenze sovrapposte. E' possibile risolvere il problema azzerando il registro a scorrimento qualora fosse riconosciuta la sequenza binaria.

Un secondo problema è legato alla prima partenza della macchina, poiché i valori all'interno dei registri non sono definiti il comportamento della macchina risulterà imprevedibile, per ovviare a questo problema sarà necessario far scorrere il dispositivo per un tempo sufficientemente lungo dando in ingresso il valore 0.

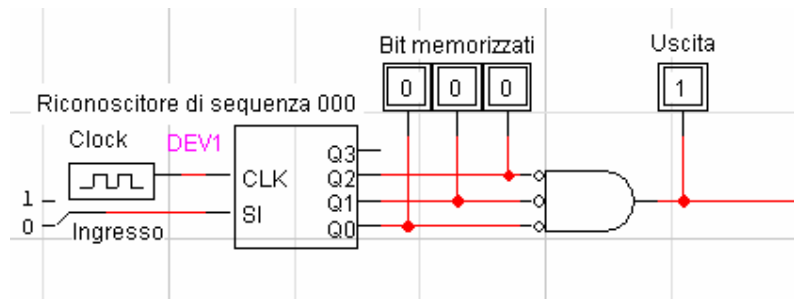
Altra osservazione, è possibile realizzare un solo riconoscitore di sequenza, in grado di riconoscere le tre sequenze 000, 111 e 010, tuttavia in base alle specifiche della macchina è conveniente tenere separati i riconoscitori poiché ciascuno di essi azionano una parte della rete.

Riconoscitore per le sequenza 111:

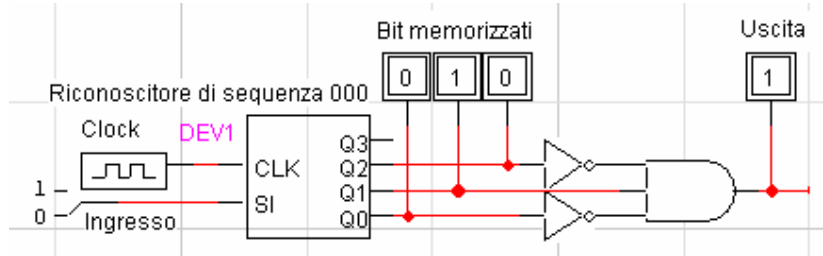


L'uscita del riconoscitore è Q_0 .

Riconoscitore per la sequenza 000:



l'uscita del riconoscitore è Q_1 . Riconoscitore per la sequenza 010:

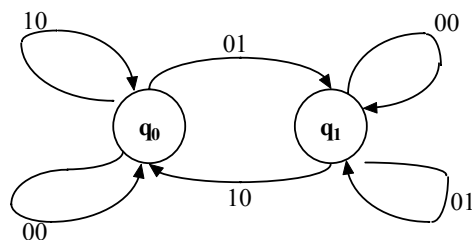


Rete di parità seriale:

Abbiamo già sviluppato in una precedente lezione la rete di parità, per valutare la parità o la disparità del numero di volte in cui si verifica la sequenza 010 adoperiamo un flip-flop T. Il flip-flop T è un contatore modulo due ed il suo uso è corretto in questo contesto poiché sarà sufficiente associare al dato memorizzato un opportuno significato (0 se la sequenza di stringhe 010 si è verificata per un numero pari di volte pari ed 1 se viceversa).

Il circuito On/Off stabilisce come e quando abilitare correttamente il circuito. Alla macchina sequenziale che andremo a realizzare daremo come ingressi le due uscite dei riconoscitori di sequenze, ossia Q_0 e Q_1 :

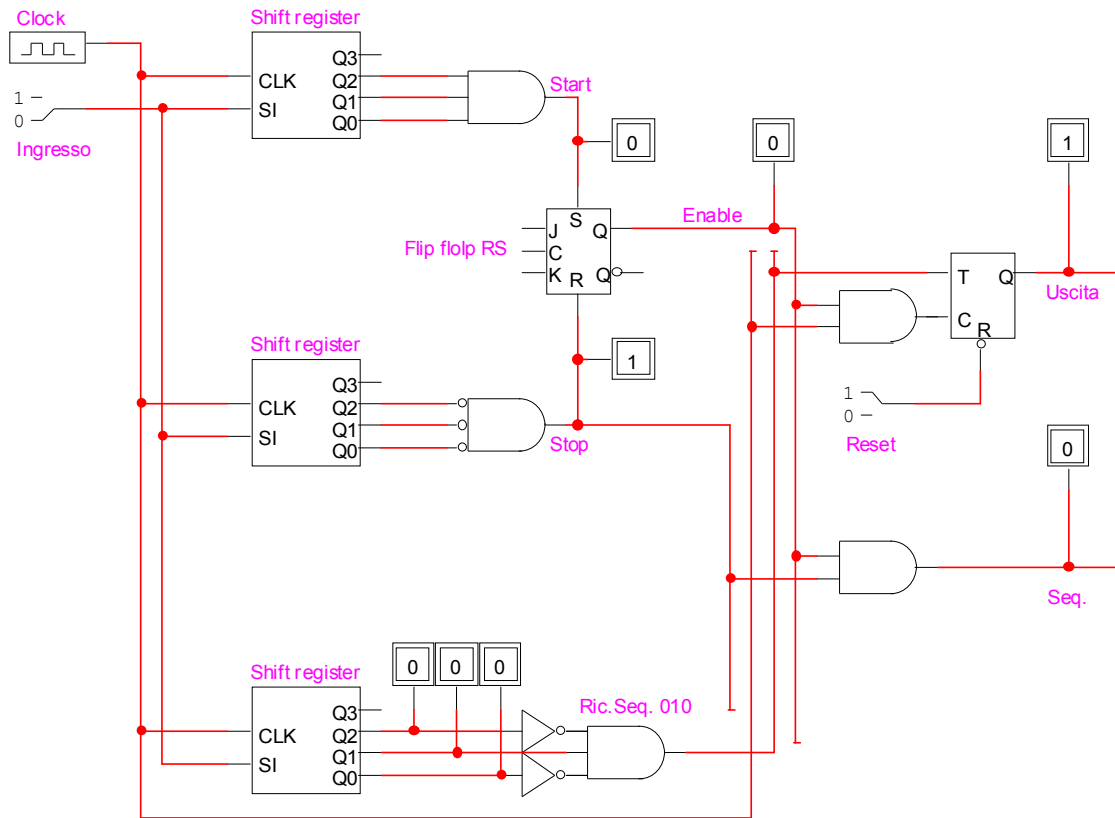
Q_0	Q_1	Cosa fare?
0	0	Niente
0	1	Off
1	0	On
1	1	Non si verifica mai!!!



Lo stato q_0 è significativo dell'evento: non effettuare il conteggio(off); lo stato q_1 è invece significativo dell'evento effettua il conteggio (on). Attenzione, l'ingresso 11 per la

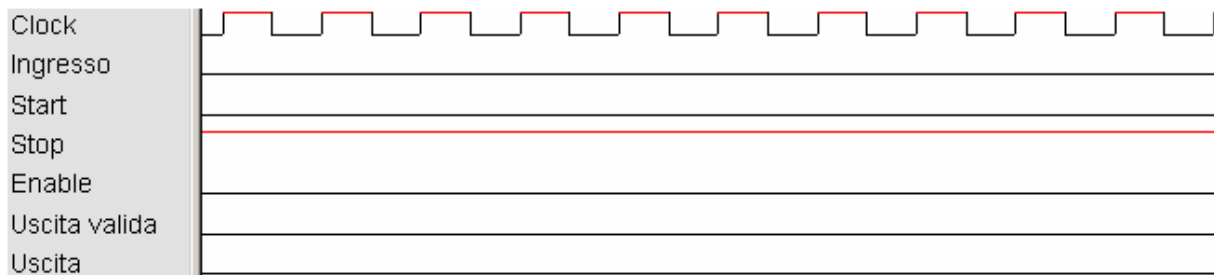
macchina sequenziale non si verifica mai, 11 indicherebbe $Q_0=1$ e $Q_1=1$ e ciò significherebbe aver riconosciuto la sequenza 111(start) e quindi $Q_0=1$ e la sequenza 000 e quindi $Q_1=1$, cosa che in una macchina progettata correttamente nei punti precedenti non accade mai!!! Inoltre il grafo della macchina che realizza il circuito On\Off è quello di un flip-flop RS.
Il circuito:

Esempio di progettazione per composizione

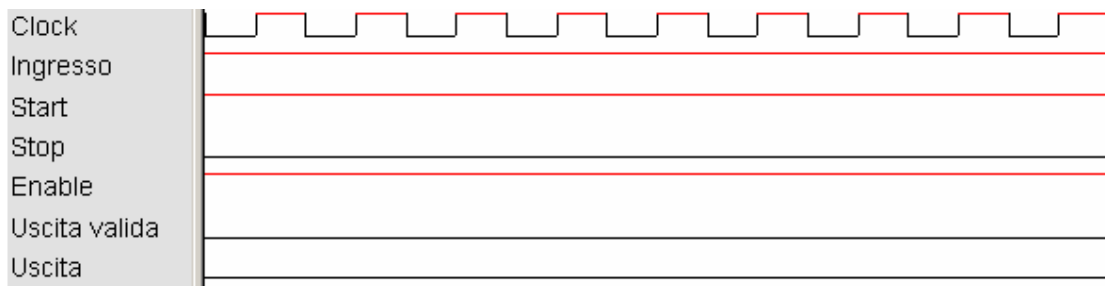


Mostriamo di seguito alcune tempificazioni della macchina realizzata, significative di alcuni momenti operativi della stessa:

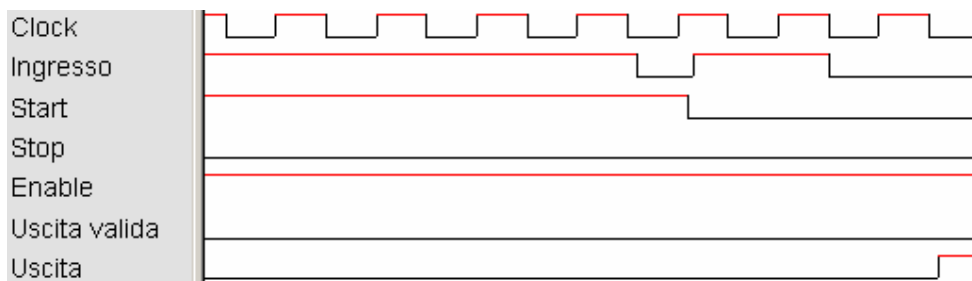
- Stato di Off, la macchina riceve in continuazione una sequenza di valori logici bassi e cioè pari a 0, in tale situazione, avremo che il segnale **enable** (quello inerente al circuito che realizza la condizione On/off) è 0, tutti gli altri segnali logici sono bassi, l'unico segnale alto è appunto quello di Stop:



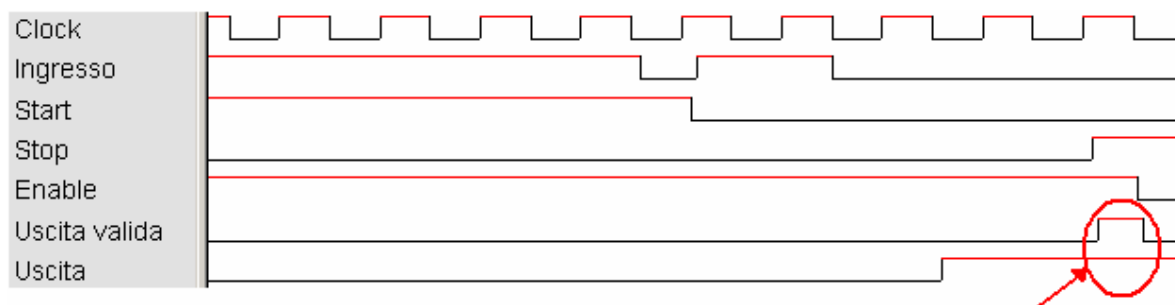
- Stato di On, la macchina riceve tre bit consecutivi di valore logico alto, il segnale di **start** dovrà diventare anch'esso alto, il segnale di **stop** dovrà invece assumere un valore basso, il segnale di **enable** dovrà anch'esso assumere un valore alto:



- Si verifica una sequenza 010, la macchina riconosce l'avvenuta sequenza che essendosi verificata una volta (e quindi in un numero di volte dispari) costituisce un'**uscita** avente valore logico 1 che tuttavia non è valida poiché il segnale 000 che decreta la fine dei calcoli non è ancora avvenuto:



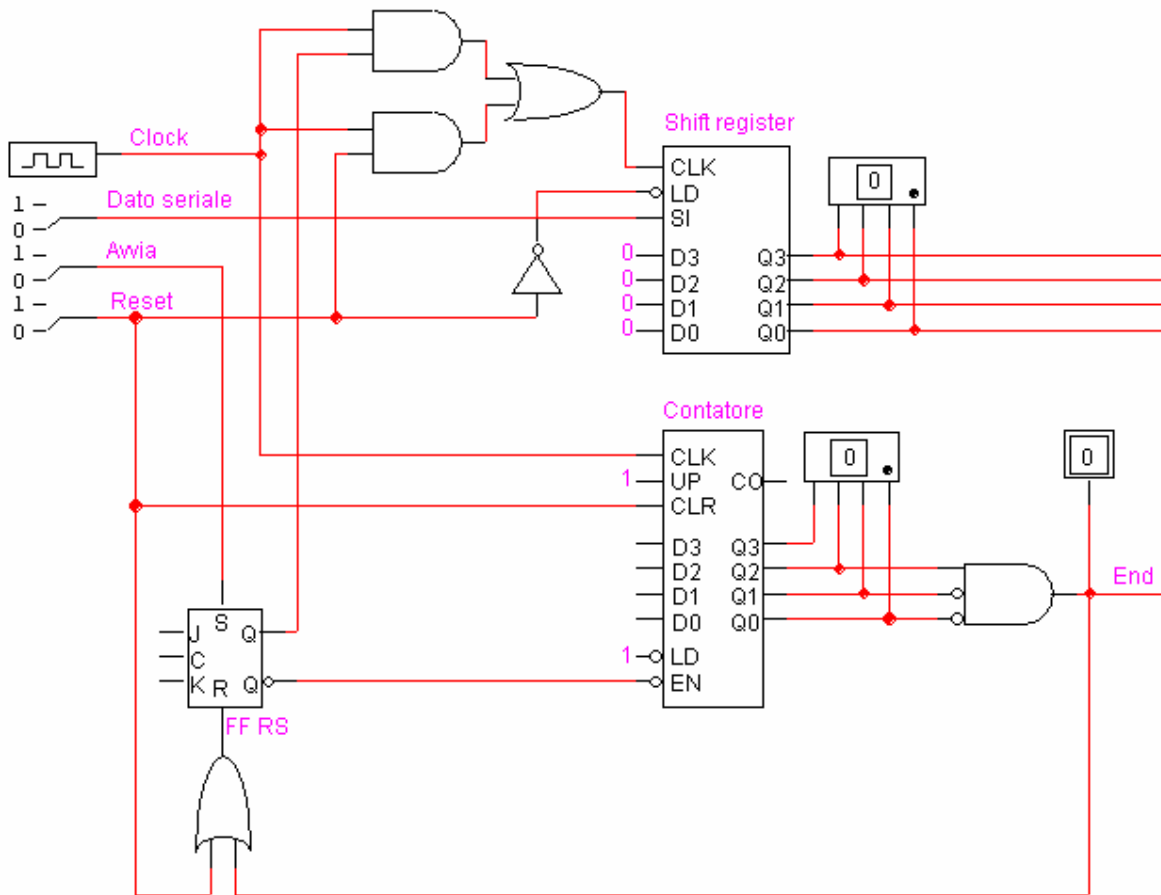
Si verifica la sequenza 000, significativa per la macchina della fine dei calcoli, il segnale **stop** assume un valore logico alto, il segnale di **enable** dopo un certo istante di tempo si abbasserà a 0, il segnale (impulsivo) inerente all'**uscita valida** assumerà per un breve istante di tempo un valore logico alto convalidando l'**uscita** che dovrà essere alta poiché la sequenza 010 si è verificata una sola volta:



Lezione N°13: "Dati in forma seriale e/o parallela, il progetto di macchine complesse"

Nella progettazione di macchine complesse (ad esempio sommatore seriali o paralleli) è spesso richiesto in fase di progettazione la conversione di un dato da una forma seriale ad una forma parallela (o viceversa).

Qualora un dato venga fornito in forma seriale è intuitivo convertire quest'ultimo in forma parallela servendosi di un adeguato shift register:



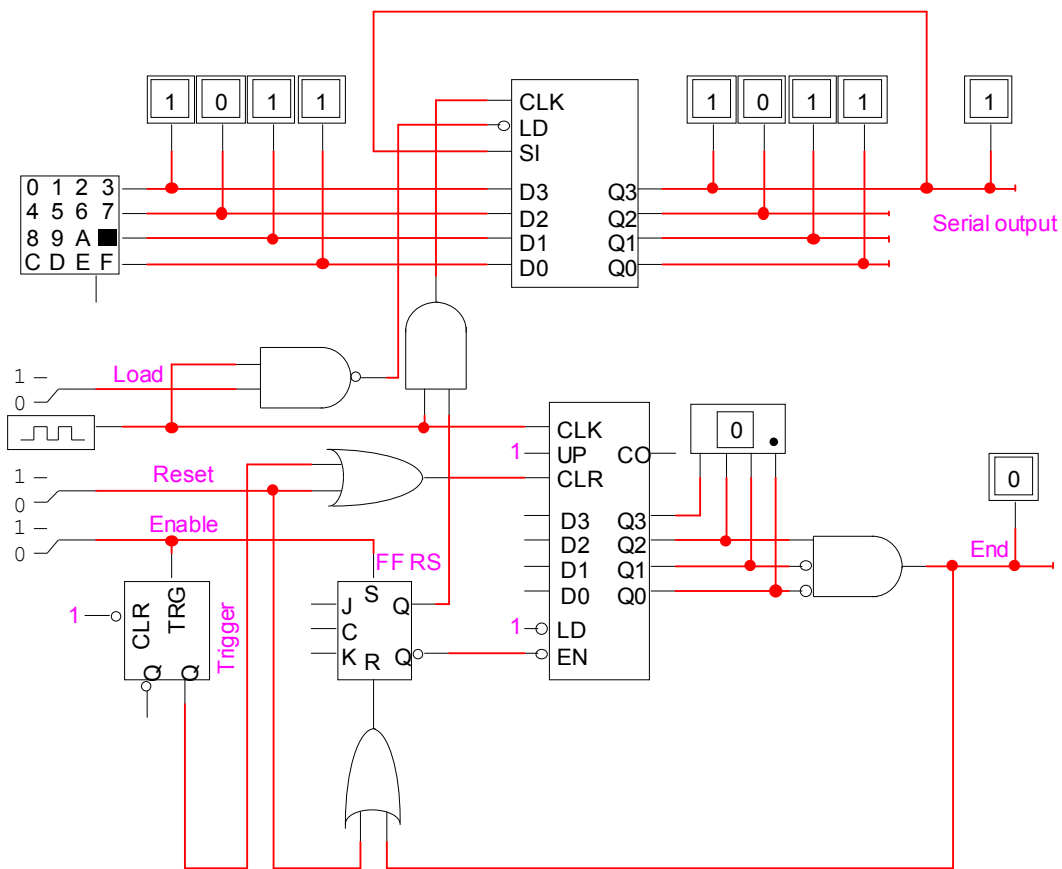
Il dispositivo realizzato ospita l'ingresso seriale mettendolo in uno shift register comandato da un contatore modulo n se n sono i bit da leggere in ingresso.

Per iniziare una conversione di un dato è indispensabile effettuare un "Reset", alzando il bit "Avvia" si dà inizio alla trasmissione, da questo momento in poi la sequenza di bit su "Dato Seriale" è letta dallo shift register che ne effettua la conversione. Il contatore arrivato all'n-simo bit disattiva la lettura del dato seriale che è ora disponibile in forma parallela.

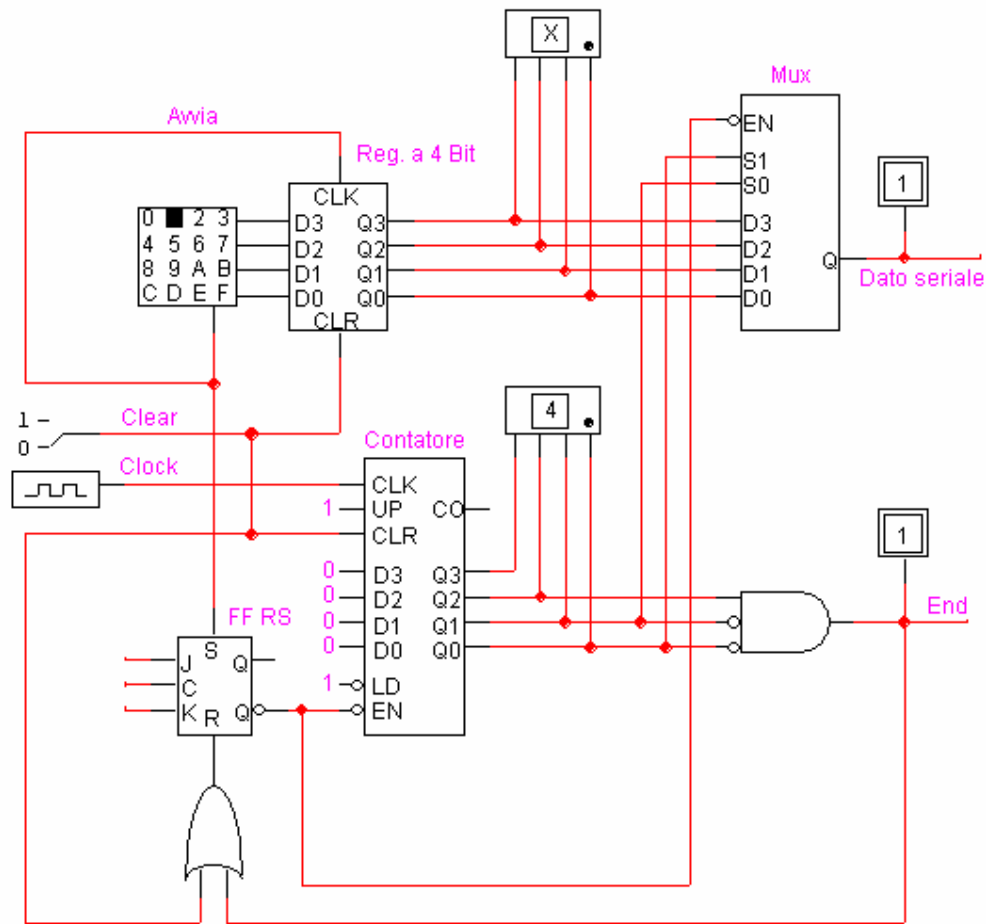
Allo stesso modo si può realizzare la conversione di un dato in forma parallela alla forma seriale, per fare ciò è necessario caricare nello shift register il valore da convertire.

Questa operazione può essere fatta ad esempio ponendo su D0, D1, D2 e D3 i bit che una tastiera numerica genera come ingresso, quindi si alza il bit load che operando sul fronte di salita caricherà i

dati nel registro. Un contatore modulo n, con n numero di bit da spedire, scandisce i tempi e arresta il conteggio ad n-1. Il dato in forma seriale è quello presente sulla linea Q3:



Leggermente diversa è la conversione di un dato dalla forma parallela a quella seriale se si pensa invece di non usare uno shift register:



Un registro legge sul fronte di salita del segnale "Avvia" gli n bit da convertire, lo stesso segnale "Avvia" abilita un contatore modulo n che per ogni valore di conteggio abilita uno alla volta i bit del registro selezionati da un multiplexer. La trasmissione termina quando tutti gli n bit sono stati convertiti ed il contatore ha raggiunto il massimo valore di conteggio.

Affrontiamo ora il problema della progettazione di una macchina complessa, problema:

Sviluppare, attraverso un approccio composizionale, il progetto di un **addizionatore binario** di interi positivi che operi serialmente su dati costituiti da stringhe di n bit (con n=8).

La macchina deve ricevere in ingresso i 2n da sommare ed ha in uscita gli n bit somma ed un segnale R tale che se alto si è verificato un **overflow**, un segnale detto **strobe** per segnalare la completata operazione. La macchina ha in ingresso due clock C1 per la sincronizzazione degli ingressi e C2 per la sincronizzazione delle uscite.

Gli elementi base che costituiscono la rete sono:

- due registri addendi A e B;
- un registro somma;
- un addizionatore seriale di n bit propriamente detto, ADD;
- Un sistema di controllo della tempificazione complessiva, CONTR;

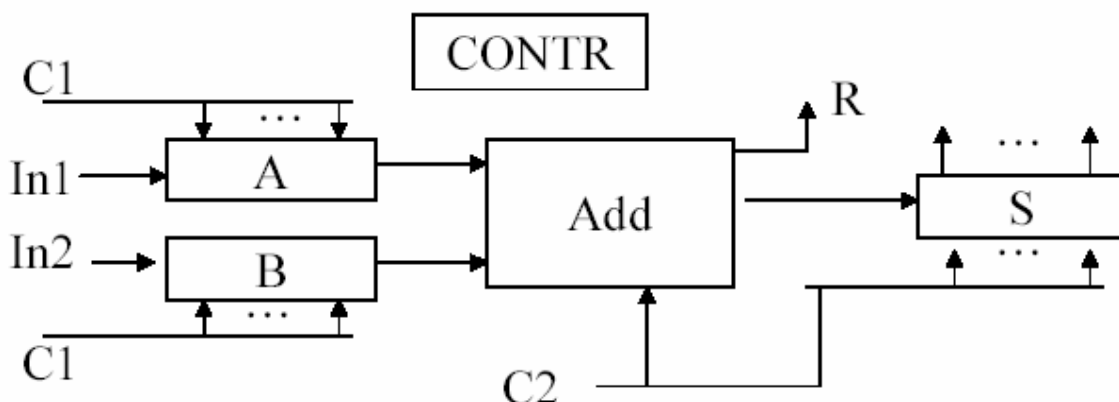
Oltre alla somma $S=A+B$, l'addizionatore ADD deve fornire, al termine dell'esecuzione della somma, un segnale binario R che indichi se l'addizione abbia provocato overflow, cioè se si verifica una condizione di questo tipo:

$$R = A + B \geq 2^n$$

Le fasi operative che caratterizzano la macchina sono:

1. Caricamento dei dati;
2. Esecuzione dell'addizione;
3. Generazione dell'addizione;

Schema di principio della macchina:



- C1 e C2 sono due clock sfasati;
- In1 ed In2 sono gli ingressi degli n bit da sommare;
- R ed S sono le uscite, R è il resto mentre S è il valore somma;
- A, B, e C sono dei registri a scorrimento;
- ADD è l'addizionatore vero e proprio;

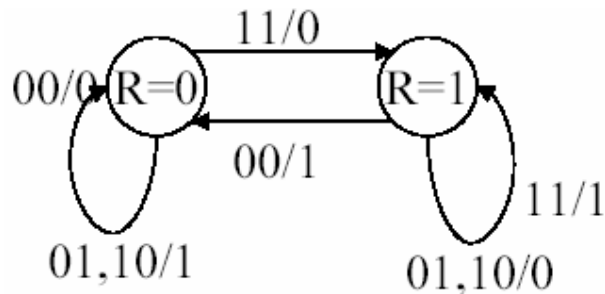
La macchina ADD si compone di due componenti:

- Un adder sequenziale di 1 bit che effettua la somma di due bit con un eventuale riporto;
- Un contatore modulo n (nel nostro caso $N=8$) per verificare che siano stati sommati effettivamente tutti gli n bit;

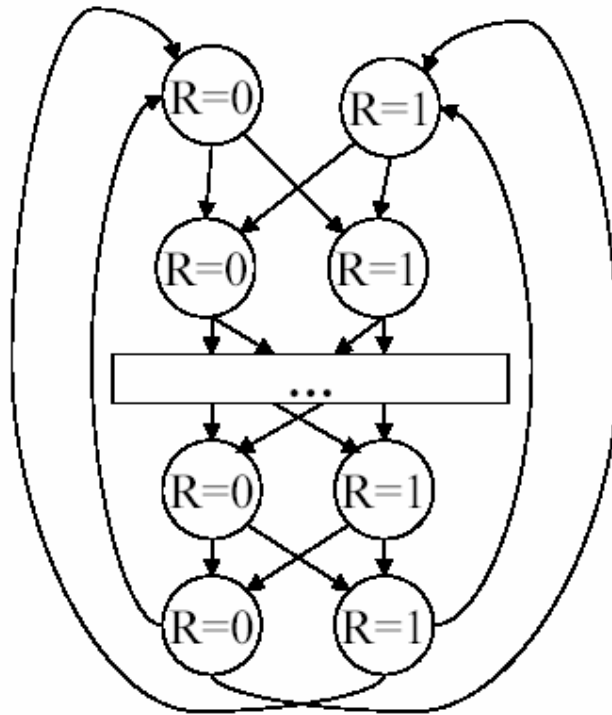
Le due reti sono opportunamente collegate fra di loro. Avviene overflow se la somma degli ultimi due bit costituisce un riporto; l'uscita deve essere fornita quando il contatore segnala che l'ultima somma è stata effettuata. L'addizionatore full-adder ha due stati interni che corrispondono al valore del resto che ogni singola somma di bit restituisce. Il valore di questo resto è utile alle successive somme, la macchina ha quindi in ingresso:

- 2 bit di ingresso corrispondenti ai singoli bit da sommare;
- 1 bit di resto;

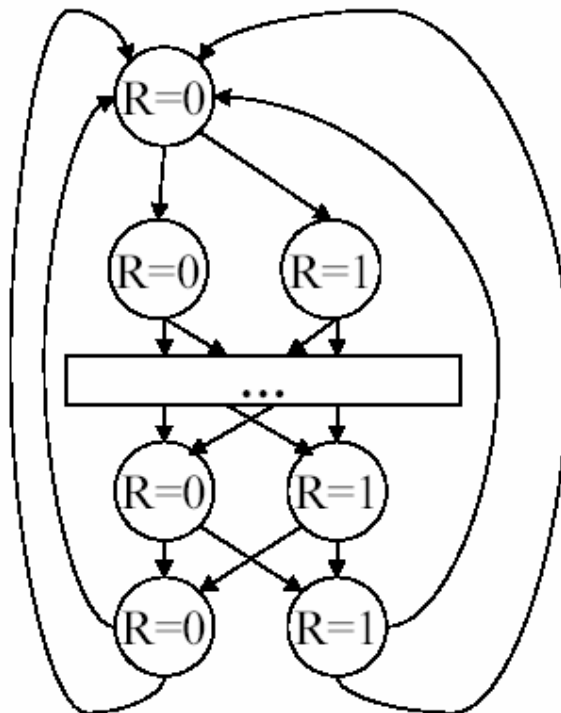
la macchina emette ad ogni colpo di clock il valore della somma e memorizza il valore del resto da utilizzare per le somme successive. Il diagramma secondo la quale opera l'addizionatore è il seguente:



La macchina completa oltre ad effettuare la somma ed a memorizzare i bit di resto deve contare gli n bit fino ad arrivare agli ultimi due bit da sommare. Sono percorribili due soluzioni per la realizzazione dell'addizionatore ad n bit. Una prima soluzione è quella caratterizzata da due stati iniziali significativi di precedenti somme di n bit e pertanto rappresentanti anche dell'avvenuto overflow, in tal caso è necessario un segnale di reset che pone inizialmente $R=0$ qualora si voglia effettuare una nuova somma.



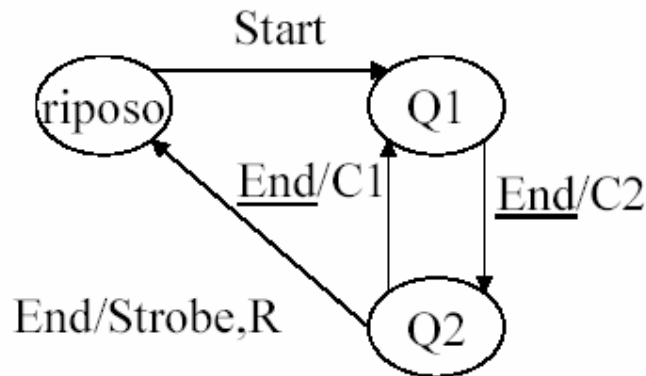
Una soluzione alternativa al problema è quella che prevede il ritorno allo stato iniziale $R=0$ alla fine del conteggio, in questo modo si perde l'informazione sull'eventuale overflow ed è quindi necessario emettere il segnale di overflow dall'ultimo stato alla transizione ad $R=0$ (il segnale verrebbe ad essere impulsivo). In questo modo si creano problemi di tempificazione



La scelta del progetto è ricaduta sulla prima alternativa al problema, quella cioè in cui gli stati iniziali sono due $R=0$ ed $R=1$, un bit di reset azzerà il tutto e prepara la macchina alle successive somme.

Il sistema di controllo: Distingueremo l'operazione complessiva nelle tre fasi:

- a) Caricamento dei dati nei registri-addendi: in questa fase vengono anche resettati il flip-flop riporto ed il contatore, in modo che tutto sia pronto per l'esecuzione della somma;
- b) Esecuzione dell'addizione: una volta che i registri-addendi siano stati caricati, l'addizione deve partire, nel senso che devono essere resi attivi ADD e i registri a scorrimento. Una rete di controllo, allora, può generare, dal clock generale di macchina, una sequenza di 8 clock da inviare alle macchine in questione;
- c) Inoltro del risultato: consiste semplicemente nell'inviare verso le apparecchiature-utenti la somma accumulatasi in S e il bit-overflow; è sufficiente allo scopo un apposito segnale strobe, generato quando il dato nel registro-somma è stabile, quindi all'ottavo colpo di clock di C1.



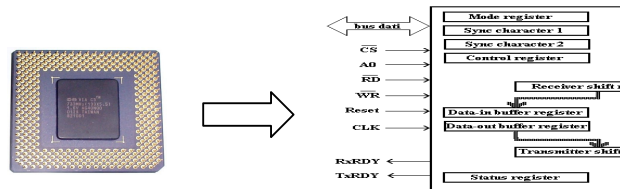
Il grafo delle transizioni della rete di controllo viene eseguito 8 volte ($n=8$) prima che la macchina si porti nella situazione di riposo.

Lezione N°14: "Il problema dell'I/O"

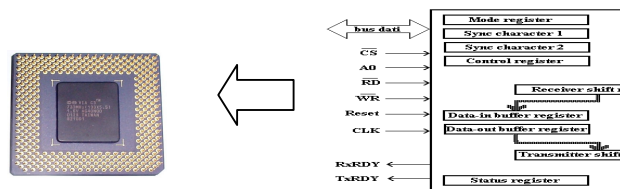
Una caratteristica importante di tutti i calcolatori è la capacità di scambiare dati con altri dispositivi (sia interni che esterni al calcolatore stesso). La comunicazione che avviene tra il processore e le periferiche è regolata da leggi predefinite che sono comunemente esplicitate nei protocolli di comunicazione.

L'interfacciamento tra sistemi a microprocessori può avvenire in svariate configurazioni, sia tra dispositivi che tra processore e dispositivo oltre che da sistemi, come viene mostrato nelle figure che seguono:

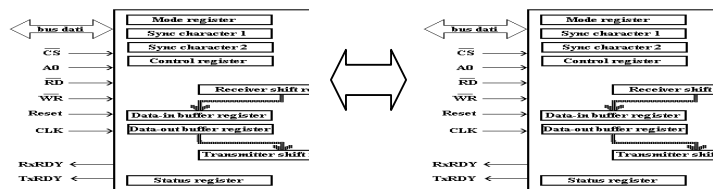
- Interfacciamento processore/dispositivo



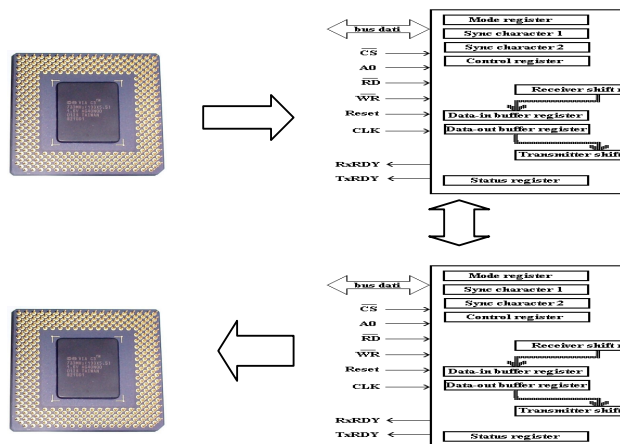
- Interfacciamento dispositivo/processore



- Interfacciamento tra dispositivi



- Interfacciamento di sistemi



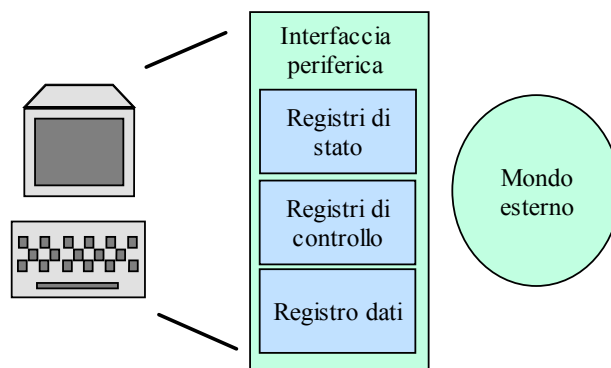
La maggior parte dei calcolatori moderni organizza la propria comunicazione tra processore e periferiche secondo una struttura a bus singolo, vale a dire che il **processore**, la **memoria** e i **dispositivi di I/O** sono collegati a questo bus.

Il bus alla quale le periferiche si vanno a collegare si compone poi di tre fili o linee, si ha infatti:

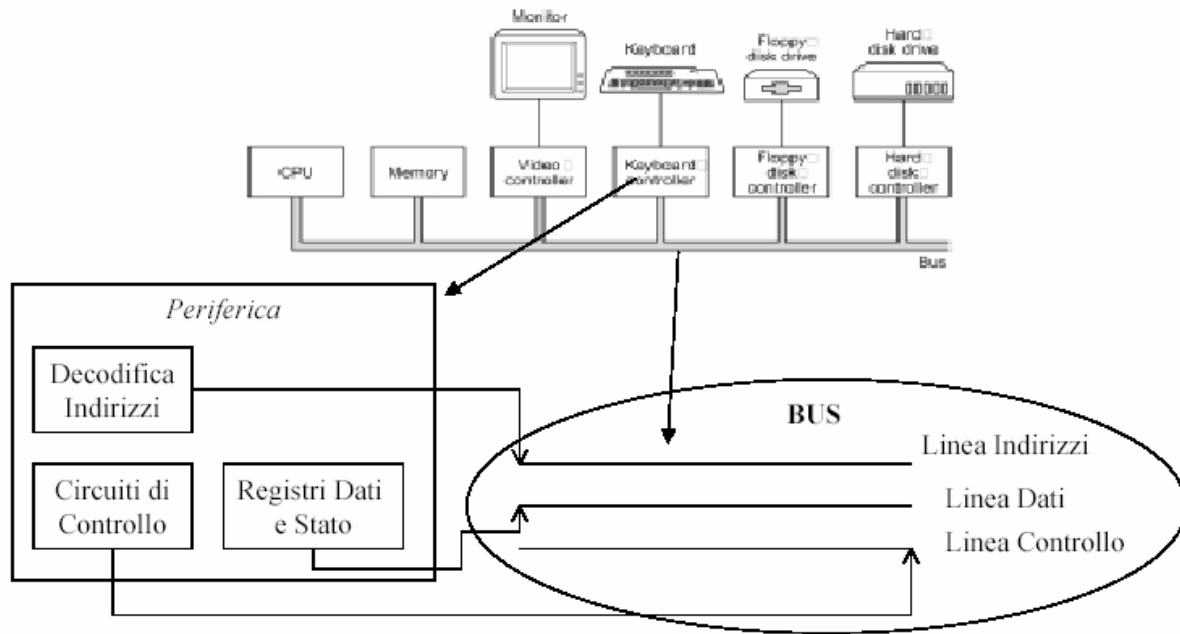
- una linea per gli indirizzi;
- una linea per i dati;
- una linea di controllo;

Ogni periferica è collegata a ciascuna di queste linee, la comunicazione si svolge attraverso i registri della periferica che sono collegati alle linee del bus e che costituiscono l'**interfaccia** della periferica. Diremo in generale interfaccia di I/O l'apparecchiatura che si interpone fra la periferica ed il processore, in genere è una collezione di registri che permettono di:

- Osservare lo Stato;
- Modificare lo Stato;
- Fornire Dati



Il processore vede l'interfaccia come l'oggetto da e verso la quale si effettua l'input o l'output e nei suoi riguardi l'interfaccia si presenta con una determinata architettura, suddivisa ad esempio in sezione di ingresso e sezione di uscita, con uno o più registri da e verso i quali il processore invia i suoi messaggi.



I dispositivi di I/O lavorano a velocità diverse e tipicamente sono molto più lenti del processore costringendo quest'ultimo a rallentare numerose volte il suo normale ciclo di funzionamento. Lo scambio di messaggi tra processore e periferiche prende il nome di colloquio, la disciplina che lo regola è il protocollo.

Dal libro di Teoria: I protocolli costituiscono un problema più ampio di quanto non interessi in questa sede: essi infatti riguardano in generale la vasta problematica delle reti di calcolatori, sono anche molto complessi e talora definiti in standard internazionali per permetterne la unificazione.

Sul piano concettuale, una **operazione di uscita** consiste nell'invio alla periferica da parte del sistema centrale (CPU) di un insieme di messaggi (dati comandi o segnali di controllo), tali che un blocco di dati venga trasferito dalla memoria ad uno specifico supporto periferico, quale il foglio di carta della stampante, lo schermo video (monitor) oppure una memoria di massa (ad esempio un hard disk).

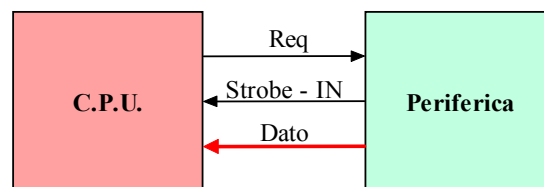
Analogamente, una **operazione di ingresso** produce il trasferimento da un supporto periferico (il buffer di una tastiera o un hard disk) alla memoria. Tra il processore e la periferica fluiscono dunque dati in entrambe le direzioni (in alcuni casi si sente anche dire che la comunicazione è bi-direzionale): verso la periferica per le operazioni di uscita, verso il processore per quelle di ingresso.

Oltre ai dati fluiscono anche altri messaggi che determinano l'esecuzione di una determinata operazione. I protocolli di comunicazione possono essere inquadrati in due classi:

- **Protocollo sincrono**: è previsto un segnale di clock che permette di gestire la temporizzazione delle macchine;
- **Protocollo asincrono (handshake)**: tutta la temporizzazione della comunicazione è gestita dal protocollo stesso attraverso lo scambio dei messaggi;

Un protocollo è l'insieme di regole che gestiscono la comunicazione tra due entità. La tecnica fondamentale maggiormente usata nei colloqui tra dispositivi è il protocollo asincrono detto anche **handshake** e che prevede per l'input: un messaggio di "richiesta dato" (tipicamente identificato sotto la sigla di segnale di **req** = request) da parte del processore ed una risposta del tipo "dato" da parte della periferica (in tal caso le risposte fornite dalla periferica verso la CPU sono varie e comprendono lo scambio di ulteriori messaggi per coordinare al meglio la comunicazione, tipicamente al segnale di req segue dopo un certo tempo un messaggio di ACK, ma sono possibili anche soluzioni proprietarie che dipendono dal costruttore); la tecnica elementare di sincronizzazione prevede inoltre che al dato in input sia associato un segnale di **Strobe-IN**, pertanto queste le azioni intraprese dal processore per il protocollo di ingresso:

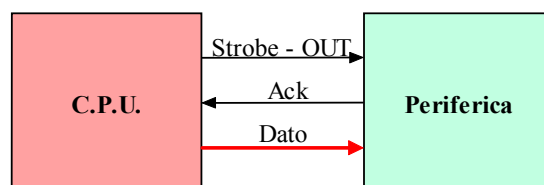
- Manda la richiesta (Req.);
- Aspetta il segnale ACK (Strobe-IN);
- Leggi il dato (Dato/i)



Schema di un protocollo HandShake per l'input di un dato

Analogamente, per l'output lo scambio è di un flusso di dati verso la periferica accompagnato da un **segnale di ack**, la sincronizzazione avviene con un segnale di **Strobe-OUT**. Queste le fasi per il protocollo di uscita:

- Scrivi i dati (Dato/i);
- Manda la richiesta (Req.);
- Aspetta un Ack (Strobe OUT);



Schema di un protocollo HandShake per l'output di un dato

I passi fondamentali del colloquio sono tipicamente contenuti in appositi programmi che costituiscono il nucleo dei driver di I/O.

La comunicazione oltre a seguire uno dei protocolli visti (sincroni o asincroni) può essere effettuata in due diverse modalità di collegamento. Esiste infatti la comunicazione parallela e la comunicazione seriale fra processore e periferica.

La comunicazione parallela avviene direttamente con un parola di un byte, vale dire che occorrono ben 8 fili per rendere possibile la comunicazione, ciascun filo trasporta informazione sulla comunicazione ed è collegato dal processore alla periferica.

La comunicazione seriale tra una CPU ed una periferica avviene un bit per volta, occorre pertanto un unico filo. Le comuni porte USB dei moderni calcolatori permettono la connessione delle periferiche al processore, USB sta per universal serial bus e si tratta di una comunicazione seriale. Nonostante il vantaggio che la connessione parallela sembra offrire poiché progettata su 8 fili, la connessione seriale è quella che tipicamente è più usata poiché più facile da gestire.

Nella comunicazione seriale, la periferica comunica con il processore attraverso una interfaccia (che si compone dei registri di controllo e stato, registri indirizzi e registri dati), tuttavia, **la comunicazione tra interfaccia e processore risulta essere sempre parallela**, è di tipo seriale la comunicazione tra interfaccia e periferica, è quindi necessario convertire la comunicazione da parallela a seriale. Problemi tipici per questo tipo di comunicazione sono i seguenti:

- Può capitare che il processore comunichi all'interfaccia il dato da spedire ma questo non viene immediatamente comunicato magari perché è in corso una precedente comunicazione, oppure perché al momento la periferica è impegnata in altre cose. Per comunicare un nuovo dato è necessario aspettare che termini la comunicazione precedente, per poter preservare il dato che si voleva spedire verso la periferica si ricorre ad un ampio uso dei buffer, i quali ospitano temporaneamente i dati che si vogliono spedire;
- Il dato che si vuole spedire non viene subito comunicato, è infatti necessario che la comunicazioni termini per avere a disposizione l'intero dato;
- Per comunicare un uovo dato è necessario aspettare che termini la comunicazione precedente;
- La comunicazione tra interfaccia e processore è sempre parallela, è necessario allora convertire la comunicazione da parallela a seriale;

Sono possibili comunicazioni seriali di tipo sincrono e asincrono, nella comunicazione seriale sincrona la sincronizzazione avviene grazie ad un segnale di clock che scandisce i tempi utili in cui dovranno avvenire le istruzioni.

Nella comunicazione seriale asincrona non è invece necessario alcun segnale di clock poiché i bit mandati uno per volta sono preceduti da una particolare sequenza di sincronizzazione, una sequenza di bit stabilisce l'inizio della comunicazione ed un'altra stabilisce invece la fine.

I driver implementano il protocollo del dispositivo rendendolo noto al processore che solo in questo modo saprà in che modo colloquiare con il dispositivo. Elementi fondamentali della comunicazione sono quelli visti dal lato periferica:

- Interfaccia della periferica;
- Protocollo della comunicazione;

e dal lato processore:

- Meccanismo di Controllo;
- Tipo di istruzioni di I/O
- Driver

Il processore può offrire nei confronti della comunicazione un set di istruzioni specifiche il cui compito è quello di ottimizzare la comunicazione con la periferica, queste istruzioni sono tipiche di un processore e tipicamente differenti da processore a processore poiché personalizzate dai produttori, è poi possibile stabilire un indirizzamento separato per le periferiche ed il processore, talvolta è anche adoperata la tecnica del memory mapping in cui lo spazio di memoria visto dal processore è condiviso tra la memoria vera e propria e quella delle periferiche.

In questo modo una operazione di scrittura o lettura può essere fatta esclusivamente rivolgendosi agli indirizzi di periferica mappati nella memoria del sistema e condivisa con il processore.

Quando si scrive ad uno di questi indirizzi si sta scrivendo alla memoria di una periferica e ciò causerà la modifica dello stato per quella periferica se le istruzioni ad essa comunicate sono tali da essere interpretate dal dispositivo.

Meccanismi di I/O

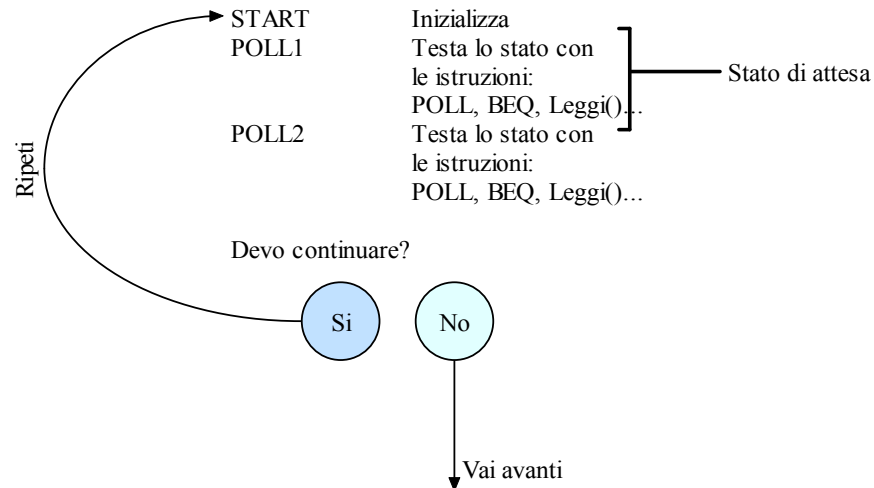
Uno dei problemi che il processore deve affrontare nella gestione delle periferiche ad esso collegato è quello della selezione della periferica con la quale colloquiare. I meccanismi per la gestione del flusso di I/O possono essere:

- Meccanismo **controllato da programma**: il processore controlla lo stato della periferica (o delle periferiche) in continuazione aspettando un cambiamento dello stato;
- Meccanismo di **accesso diretto alla memoria DMA**: la periferica è in grado di accedere alla memoria senza l'aiuto del processore;
- Meccanismo che prevede le **interruzioni**: il processore procede nel suo lavoro, quando la periferica cambia stato manda un segnale al processore che interrompe il lavoro corrente e procede a gestire l'evento;

Quindi nei dispositivi o meccanismi che prevedono il controllo da programma o le interruzioni, la CPU esegue direttamente le istruzioni di I/O operando attraverso l'interfaccia, nei dispositivi DMA la delega ad apposite apparecchiature attive quali canali o processori di I/O.

Il funzionamento di un DMA è possibile solo se è noto un indirizzo di partenza, un indirizzo di fine operazione e un indirizzo di destinazione. Esso può essere visto come un contatore che ciclicamente aggiunge all'indirizzo di destinazione un certo scostamento fino all'indirizzo di fine operazione.

Un modello di driver è quello già visto nel corso di "Calcolatori elettronici" e che si basa sulla tecnica del polling:



Quando si sviluppa un driver in questo modo ci sono enormi vantaggi dovuti alla modularità del programma che ne permette una più facile programmazione, tuttavia particolare attenzione va fatta nel momento in cui viene effettuata una operazione o istruzione di polling. Infatti, il ciclo di attesa potrebbe essere infinito e portare quindi ad una scarsa efficienza del driver stesso. La strada che adotteremo nella progettazione di un driver di fatto non sarà questa ma seguirà l'idea prima accennata che si basa sulla tecnica delle interruzioni che a breve indicheremo.

Le interruzioni

Il meccanismo delle interruzioni si basa sull'idea di dare alla periferica e ai dispositivi collegati al processore la possibilità di richiamare l'attenzione di quest'ultimo quando si verificano alcune condizioni che per essere risolte necessitano di un particolare programma contenuto nel driver.

Il processore è destinato ai suoi compiti abituali sintetizzati dal ciclo di istruzioni fetch, decode ed execute:

```
while (true)
{
    Fetch();
    Decode();
    Execute();
    CheckForInterrupt();
}
```

Le operazioni del processore seguono ciclicamente queste istruzioni; con l'introduzione delle interruzioni il processore può in alcuni casi, al verificarsi di una particolare condizione, compiere altre operazioni. Una interruzione è un segnale mandato direttamente dalla periferica al processore per interromperne l'esecuzione e lanciare l'esecuzione di un programma differente. Il controllo che il processore deve fare per sapere se è avvenuta o meno una interruzione si svolge all'interno del ciclo di istruzioni fetch, decode ed execute.

Tuttavia una interruzione non può bloccare il processore in un istante qualsiasi, infatti se durante le istruzioni di fetch e decode dovesse arrivare un segnale di interruzione, il processore continua l'istruzione corrente portandola a termine. Quindi, solo dopo la fase di esecuzione dell'istruzione (execute) il processore può dare ascolto alla periferica che ha provocato l'interruzione.

Questo modo di procedere è ragionevole se si pensa che il processore potrebbe essere continuamente interrotto qualora il controllo di una interruzione si verificasse ad esempio tra le fasi di fetch e decode. In tali circostanze il processore, impegnato a gestire le varie interruzioni che si susseguono, non porterebbe mai a termine il suo lavoro ordinario. Pertanto, le interruzioni interrompono il processore ma non lo fanno in modo casuale, esse possono solo interromperlo alla fine delle istruzioni di fetch, decode ed execute.

Gli eventi eccezionali in grado di generare interruzioni vengono tipicamente classificati in tre differenti categorie:

- **reset**, riporta la macchina in uno stato noto. Esistono due tipi di reset, esiste un **reset hardware** provocato dal tanto conosciuto tasto posto sul case di ogni computer ed un **reset software** come ad esempio la magica combinazione di tasti CTRL+ALT+CANC (anche questa famosa) che realizza la condizione di reset a mezzo di un programma. Il programma genera i segnali utili alla chiusura di tutti i programmi in

funzione e solo quando tutti questi si sono chiusi genera il segnale di reset hardware per spegnere il computer e tutte le sue apparecchiature e periferiche;

- **traps**, questo evento eccezionale è in grado di generare interruzioni mandate via software. La trap (trap = trappola) è un'istruzione del processore che serve ad emulare la presenza di un'interruzione. All'interno di un programma l'utente può mettere questa istruzione in modo tale da simulare una vera e propria interruzione;
- **interrupts**, è il segnale vero e proprio che genera una interruzione ed è quello che da ora in poi considereremo;

Sono da considerare elementi fondamentali alla gestione delle interruzioni il segnale di Interrupt (che interrompe il processore), segnale di Ack (che indica il riconoscimento dell'evento interruzione) e la ISR (Procedura di Servizio degli Interrupt), procedura lanciata quando giunge l'interrupt.

Quando si verifica una interruzione si susseguono i seguenti eventi o azioni:

- il dispositivo genera il segnale;
- il processore interrompe il programma in esecuzione;
- i registri PC e SR vengono salvati nello stack;
- il dispositivo viene informato che l'interrupt è stato ricevuto (segnale di ack);
- viene eseguita la procedura (ISR);
- si ripristina il programma originale;

Le fasi che abbiamo elencato sono visibili dall'esterno e costituiscono il protocollo di comunicazione tra periferica e processore (a meno di qualche particolare dettato dalle specifiche della periferica e del protocollo utilizzato). Il dispositivo chiama il processore avvisandolo della interruzione solo all'inizio della comunicazione, quando cioè questo genera il segnale (corrisponde al primo evento nella lista), dopodiché esso non colloquia più con il processore e la comunicazione si svolge poiché comandata dal programma.

All'interno del processore, quando arriva il segnale di interruzione, quest'ultimo procede nel salvare il registro PC e il registro SR. A questo punto il processore si pone il problema di individuare quale periferica ha provocato l'interruzione e di risolverla in qualche modo.

Il processore fa quindi partire la procedura adeguata (la ISR, che come prima operazione effettua un salvataggio del contesto, questo è differente dalla sub-routine che quando lanciata effettua anch'essa il salvataggio ma può anche non salvare tutto il contesto). Ecco allora alcune domande tipiche:

- Come si individua la periferica che ha mandato il segnale? (e quindi anche la corretta procedura da lanciare?)
- Un interrupt deve sempre essere servito?
- Che succede se un interrupt giunge mentre un altro interrupt sta venendo eseguito?

Non tutte le interruzioni vanno servite, si usa infatti associare a ciascuna interruzione che arriva una priorità per stabilire quale tra l'istruzione attuale e quella prossima è più importante. Lo strumento che il processore usa per la gestione di più interruzioni è la maschera delle interruzioni. La maschera delle interruzioni è un vettore di bit che indica il livello di priorità dell'interruzione attualmente in esecuzione (al valore 0 è associata nessuna interruzione).Quindi:

- Ciascun Interrupt viene associato ad un livello di priorità;
- Tutti gli interrupt che giungono con priorità più bassa non vengono eseguiti;
- Nella maschera degli interrupt si memorizza il livello di priorità attuale;
- Spesso esiste un livello di priorità non mascherabile;

Quando un dispositivo provoca un segnale di interruzione e non viene supportato dal processore (perché in quel momento è impegnato a gestire una interruzione con un livello di priorità più alto) quest'ultimo resta in attesa fintanto che la sua richiesta non viene accolta (questo significa che il bit o il segnale di req che il dispositivo genera rimane attivato). Tuttavia in alcuni casi può capitare anche che la periferica, avendo generato un segnale di interruzione, decida in seguito di annullare la sua richiesta e quindi di non riproporre il proprio segnale al processore (generando ad esempio il tipico errore di timeout delle vecchie stampanti che dotati di un interrupt molto basso venivano quasi sempre scavalcati da altri processi, quando infatti passavano più o meno 100 o 300 secondi il segnale non veniva più riproposto alla CPU e ciò veniva comunicato all'utente con una finestra che recitava appunto l'errore di timeout).

Per risolvere il problema degli **interrupt innestati** (quelli cioè che avvengono quando il processore sta già gestendo una interruzione) sono possibili tre soluzioni:

- **Soluzione 1:** Non controllo la linea Interrupt fino a quando non ho terminato la procedura attuale;
- **Soluzione 2:** Memorizzo nello SR che sto eseguendo un Interrupt, la maschera degli interrupt mi dice quali possono avvenire (ovviamente quelli con priorità più alta di quella attuale);
- **Soluzione 3:** Controllo la linea degli interrupt solo sul fronte di salita;

In virtù di quanto detto accade che:

- Il dispositivo genera il segnale;
- Il processore interrompe il programma in esecuzione;
- **Gli interrupt di priorità più bassa vengono disabilitati;**
- I registri PC ed SR vengono salvati nello stack;
- Il dispositivo viene avvisato che l'interrupt è stato ricevuto (ack);
- Viene eseguita la procedura (ISR);
- Si ripristina il programma originale;

Cosa fare se ci sono più dispositivi?

Il processore deve in qualche modo identificare il dispositivo che effettivamente ha mandato il segnale. Anche in questo caso le soluzioni a questo problema sono varie ed impiegano un misto di hardware e di software, inoltre tutte le soluzioni dipendono fortemente sia dall'architettura del sistema che da quella del processore, queste le soluzioni tutto ora adottate:

- **Polling**: interrogo tutti i dispositivi basandomi su come sono collegati alla CPU, ciò comporta un dispendio di tempo dovendo infatti interrogare tutti i dispositivi fino a trovare quello che ha generato il segnale di interruzione (il dispositivo potrebbe essere il primo di una catena ma anche l'ultimo), è poi possibile ordinare i dispositivi nella catena secondo un criterio basato sull'importanza dei dispositivi;
- **Interrupt vettorizzato**: il dispositivo che genera una interruzione manda un identificativo come risposta al segnale ack (come un allegato al segnale di ack), tale identificativo è quindi unico per ogni dispositivo che è così univocamente identificato;
- **Interrupt autovettorizzato con raggruppamento**: i dispositivi vengono raggruppati e la procedura selezionata attraverso il vettore delle interruzioni interroga i dispositivi;

Nell'interrupt vettorizzato il dispositivo manda il segnale di interrupt assieme ad un codice (quello che serve alla sua identificazione). Il codice è un indice di un vettore delle interruzioni. Il contenuto del vettore delle interruzioni è l'indirizzo di memoria dove si trova la procedura che gestisce l'interruzione al codice fornito.

Il colloquio, tra periferica e processore, quando si adotta l'interrupt vettorizzato è del tipo:

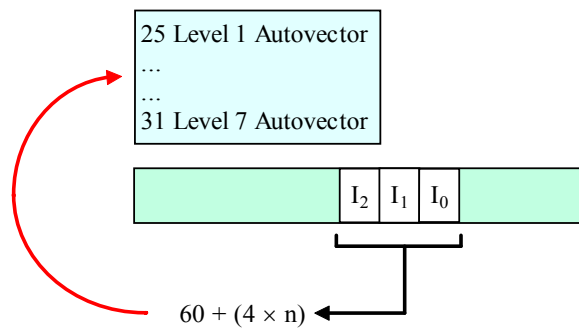
- La periferica manda un segnale di interrupt;
- Il processore risponde alla periferica con ack;
- **La periferica manda un codice di interruzione**;
- Il contenuto del vettore delle interruzioni è l'indirizzo di memoria dove si trova la procedura che gestisce l'interruzione al codice fornito e che il processore prende come riferimento;

Che cosa è il vettore delle interruzioni?

Identificare un dispositivo corrisponde anche a scegliere la corretta ISR da eseguire contenuta nel driver o programma, per distinguere le differenti ISR si utilizza il "Vettore delle Interruzioni". Il Vettore delle Interruzioni è un array contenente indirizzi di memoria i cui indirizzi corrispondono alla locazione di memoria dove si trova la ISR da eseguire.

Nel caso in cui si usa la tecnica dell'interrupt autovettorizzato accade si verificherà una sequenza di eventi come la seguente:

- La periferica manda il segnale di interrupt;
- In base alla priorità del segnale di interruzione si accede ad una locazione di memoria prefissata contenente l'indice del vettore delle interruzioni;
- Dal vettore delle interruzioni si ricava l'indirizzo di memoria della corretta ISR da eseguire



Simulazione di un'interfaccia seriale programmabile in ASIM

(Dalla guida ad ASIM con qualche commento aggiunto fatto a lezione)

Nell'ambito della progettazione di sistemi complessi particolare importanza ha la simulazione di dispositivi periferici di tipo seriale. Tali dispositivi permettono:

- di realizzare il collegamento di una periferica seriale (ad esempio una tastiera o un mouse) ad un calcolatore;
- di collegare due computer distanti attraverso una linea telefonica oppure più semplicemente in rete;
- di convertire un dato dal formato parallelo a quello seriale e viceversa;

Il componente preso di riferimento per questa simulazione è l'**Intel 8251A**. In figura 1 è mostrato il suo modello con i registri e le linee fisiche simulate. Per questo componente sono stati realizzati i registri e le posizioni dei bit nei registri aderenti al componente reale.

L'interfaccia seriale da simulare è un componente periferico programmato dalla CPU per realizzare comunicazioni seriali di dati. Essa accetta i caratteri dalla CPU in formato parallelo e poi li converte in un flusso continuo di dati seriali per trasmetterli. Simultaneamente, esso può ricevere un flusso di dati seriali e convertirli nel formato parallelo per essere letti dalla CPU. Il componente segnala alla CPU quando esso è pronto ad accettare un nuovo carattere da trasmettere e quando ha ricevuto un carattere che può essere letto dalla CPU stessa; quest'ultima può leggere lo stato completo dell'interfaccia in qualsiasi momento.

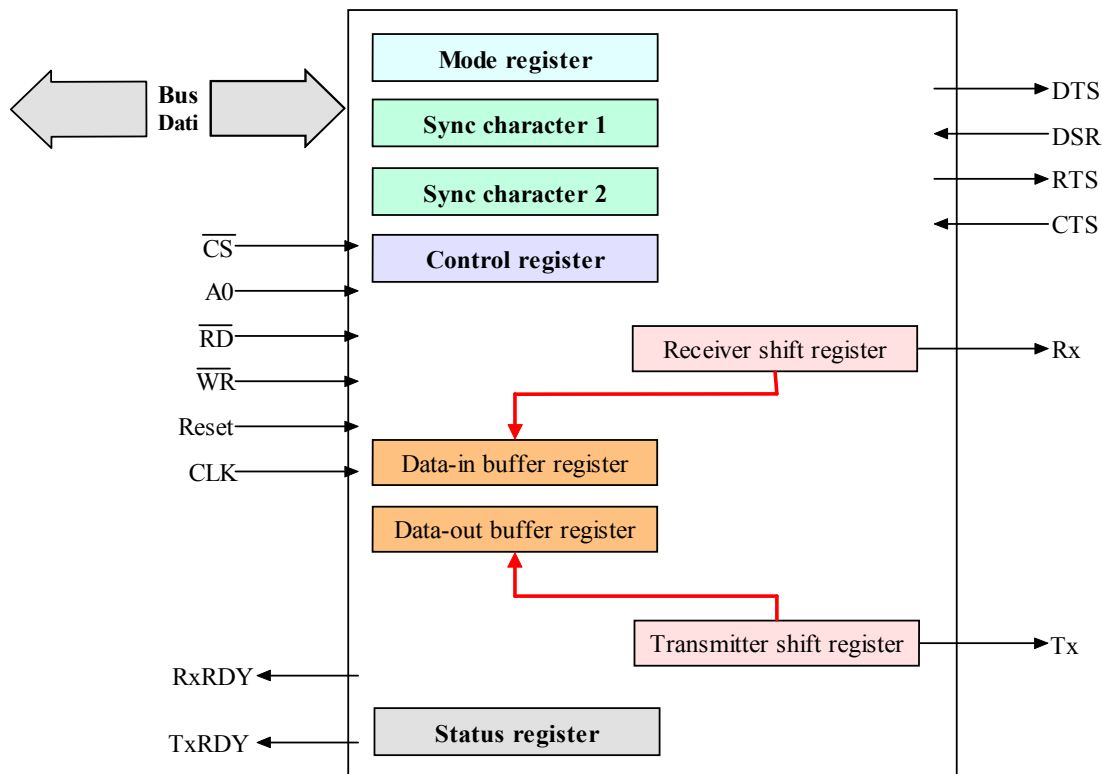


Figura 1

Come un comune dispositivo la USART (Intel 8251A) è dotato di registri dati, registri indirizzi e registri di controllo e stato, tutti fisicamente connessi al bus dati e quindi alle rispettive linee indirizzi, linee dati e linee di controllo.

In particolare la USART ha 4 registri di controllo, quelli in alto a sinistra e cioè: mode register, sync character 1, sync character 2 e control register.

Il dialogo che il dispositivo intrattiene con il processore è la parte che più ci interessa essendo questa realizzabile con alcune righe di programma, ciascuna delle quali comanda il dispositivo attraverso i segnali di controllo, la parte di macchina a sinistra è quella che si interfaccia con il processore.

Abbiamo i segnali: CS per selezionare il dispositivo (abilitazione), A0 per selezionare il registro interno (è bene notare che con un solo bit è possibile fare due soli riferimenti ad indirizzi, tuttavia lo stato di altri segnali ci permette comunque di selezionare i registri interni del dispositivo, RD/RW sono i segnali di lettura e scrittura, RxDY è il segnale di interruzione qualora si riceve un carattere, TxDY è il segnale di interruzione qualora si trasmette un carattere.

L'utilità di quest'ultimo segnale (TxDY) è di fondamentale importanza, infatti qualora si sta spedendo più di un carattere (un byte) il processore saprà in questo modo quando può procedere con il carattere successivo.

Altre caratteristiche importanti della USART sono:

- possibilità di diversi tipi di comunicazione come quella di tipo sincrono oppure asincrono (a breve saranno definite le modalità);
- trasmettitore e ricevitore full duplex con doppia bufferizzazione (shift register);
- trattamento di caratteri da 5 ad 8 bit;
- inserzione automatica dei caratteri di sincronismo;
- rilevazione di errori di parità, di overrun e di framing;

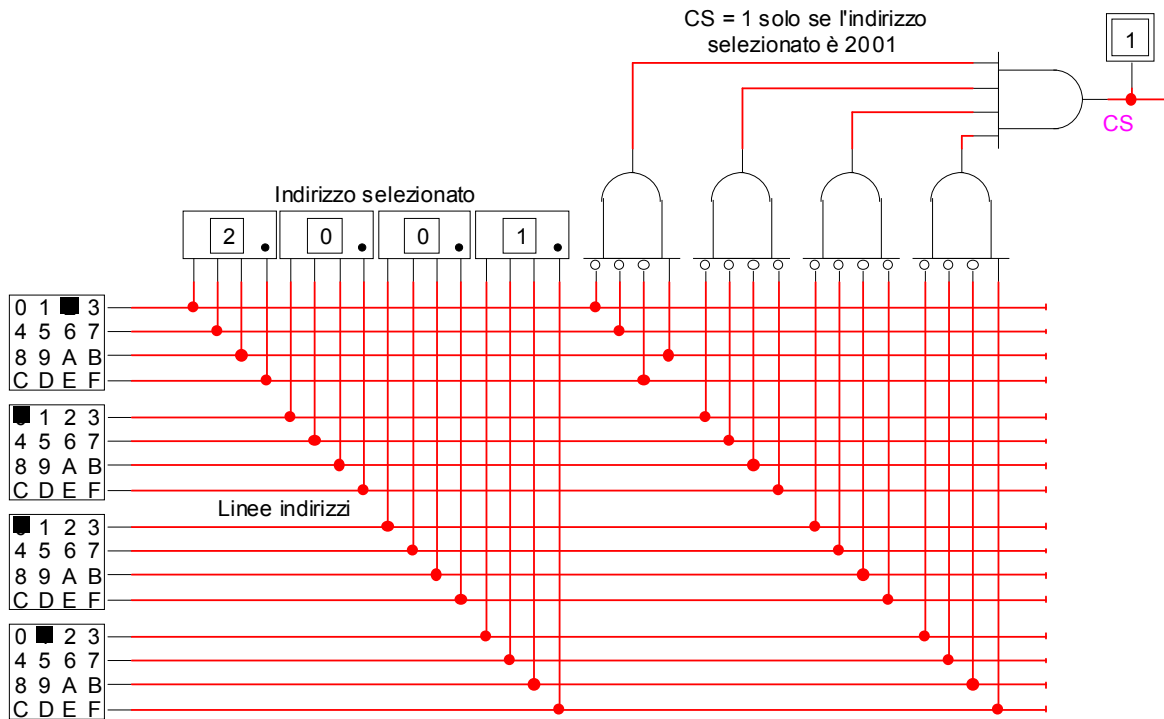
Configurazione

In questo paragrafo saranno descritti i criteri per inserire il componente da simulare in una configurazione. La descrizione verterà sui collegamenti da realizzare tra il processore e l'interfaccia seriale e tra quest'ultima e un device. Sarà infine mostrato come questi collegamenti sono creati nel nostro ambiente di simulazione (ASIM).

In generale due dispositivi sono collegabili quando hanno i segnali d'ingresso e uscita compatibili tra loro. Ciò significa che le uscite di un dispositivo devono essere dello stesso tipo delle entrate dell'altro e viceversa, inoltre deve essere possibile rispettare i protocolli di handshaking di entrambi i dispositivi.

Collegamento al processore

Le linee fisiche, che collegano attraverso il sistema bus l'interfaccia seriale al processore, sono mostrate in figura 2. Il processore scambia i dati con il componente attraverso il bus dati e seleziona lo stesso mettendo sul bus indirizzi uno dei due indirizzi ad esso associati. Poi, un decodificatore di indirizzi, collegato al bus indirizzi, quando rivela uno degli indirizzi associati all'8251A, seleziona il componente attraverso la linea CS (come da esempio).



I registri interni sono selezionati dal bit meno significativo del bus indirizzo quali A0; dai segnale di lettura-scrittura RD e WR; oltre che dallo stato interno.

Un segnale sulla linea Reset riporta il componente nello stato iniziale cancellando tutti i suoi registri. Questo segnale può essere spedito sia dal processore che da un dispositivo esterno.

RxRDY e TxRDY sono due linee d'interruzione che trasmettono al processore o all'eventuale P.I.C. una richiesta d' interruzione.

L'interruzione su **RxRDY** è inviata quando viene ricevuto un carattere in **Receiver shift register** ed è copiato in **Data-in buffer register**.

Su **TxRDY**, invece, l' interruzione viene inviata quando viene copiato il carattere dal **Data-out buffer register** in **Transmitter shift register** ed inizia la trasmissione.

Un esempio di collegamento dell' interfaccia seriale con un processore è mostrato in figura 2, dove per il Motorola 68000 sono presenti solo le linee coinvolte nel collegamento.

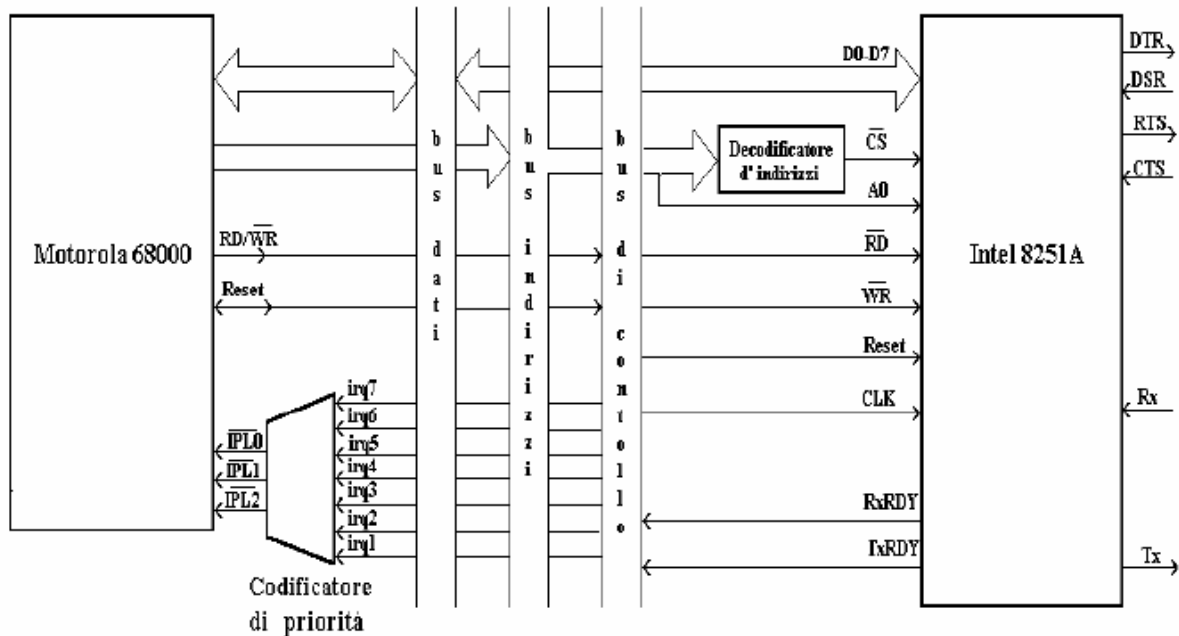


Figura 2

Collegamento ad un device

Sulla destra dello schema in figura 1 sono mostrate le linee fisiche che collegano l'interfaccia seriale ad un device. Le cinque linee poste più in alto vengono utilizzate per lo sviluppo del protocollo di handshaking ed il loro significato è fornito in tabella 1.

Tabella 1:

Segnali per lo sviluppo del protocollo di handshaking

Segnale	significato
DTR	L'interfaccia chiede al modem di connettersi alla linea
DSR	Il modem segnala all'interfaccia che si è connesso alla linea
RTS	L'interfaccia chiede al modem di trasmettere
CTS	Il modem invia in linea la portante e segnala all' interfaccia che è pronto a trasmettere

Le altre due linee, **TX** e **RX**, sono utilizzate rispettivamente per la trasmissione e la ricezione dei dati.

Un componente che volesse interfacciare con la periferica seriale Intel 8251A deve avere le seguenti proprietà o caratteristiche:

(1) Deve essere provvisto di linee compatibili alle seguenti 6 linee: **DTR**, **DSR**, **RTS**, **CTS**, **TX** ed **RX**, se si vuole effettuare una connessione piena con l'interfaccia seriale , oppure, se si vuole realizzare una connessione parziale, deve disporre solo di linee compatibili a **TX** ed **RX**;

(2) Deve rispettare il protocollo di handshaking;

(3) Nel caso di comunicazione asincrona la trasmissione deve avere il formato presentato in figura 3.

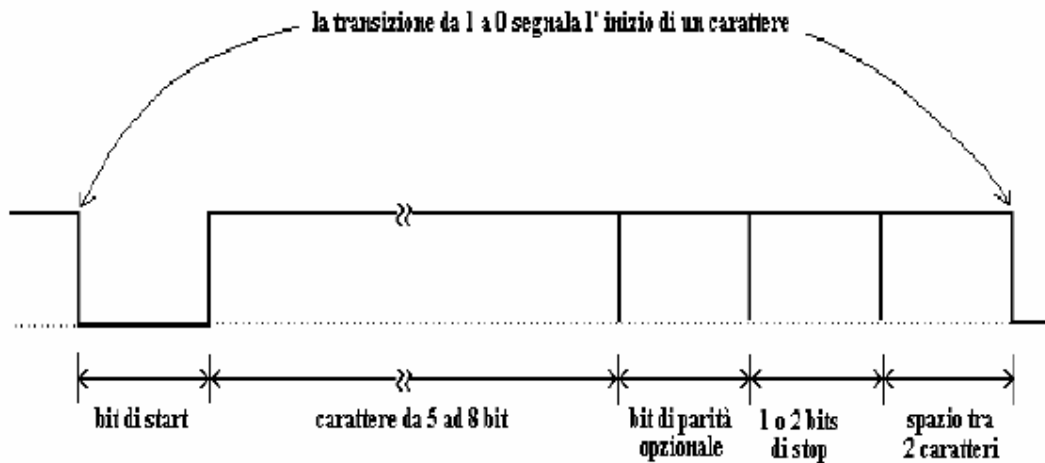


Figura 3

In figura 4 sono mostrate l'evoluzioni dei segnali di handshaking dell'interfaccia che **trasmette** i dati nel caso in cui dopo la trasmissione del messaggio non venga sconnesso il modem dalla linea:

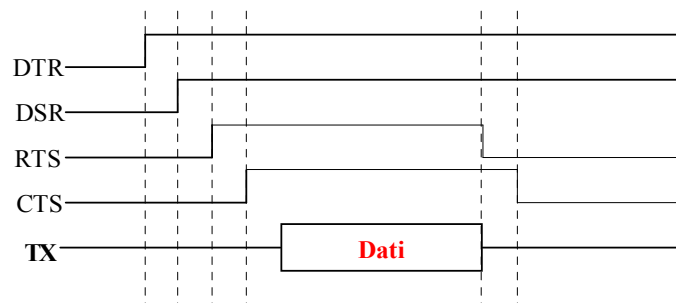


Figura 4

Con riferimento all'esempio precedente sono mostrate in figura 5 l'evoluzioni dei segnali di handshaking dell'interfaccia **ricevente**. La scala temporale è uguale a quella della figura 4:

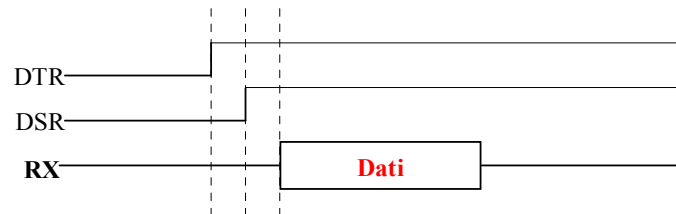


Figura 5

Un esempio di connessione di due interfacce seriali è rappresentato in figura 6:

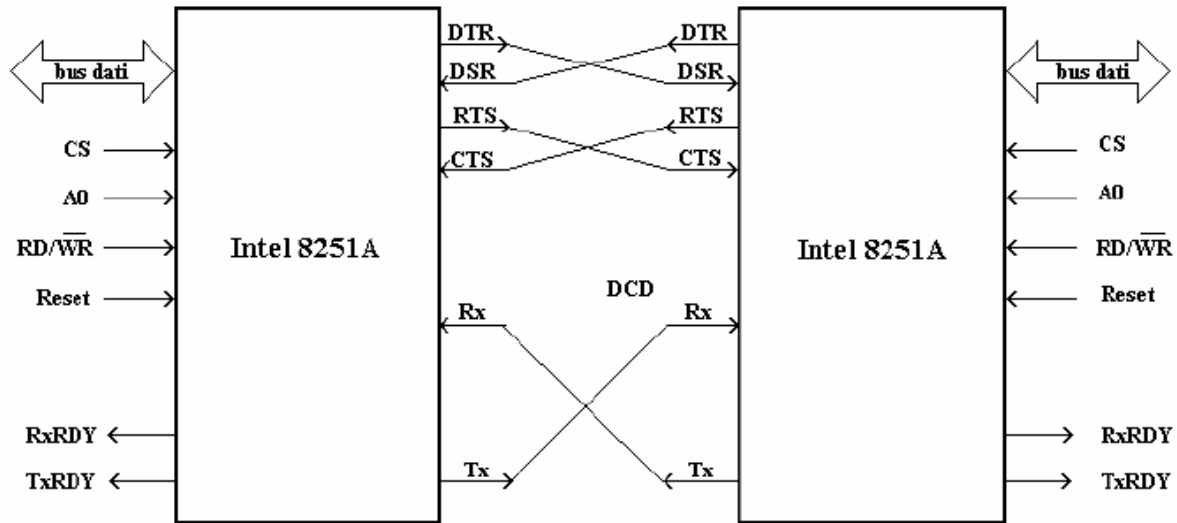


Figura 6

Oltre a questo tipo di connessione che solitamente si accompagna con l'aggettivo "**piena**" (connessione piena) è possibile simulare anche una connessione "**parziale**", la quale coinvolge solo le linee **RX** e **TX** come viene mostrato in figura 7:

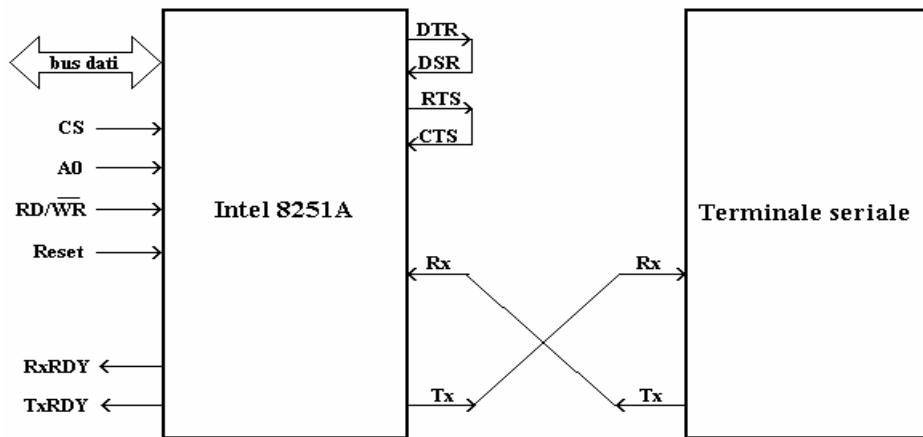


Figura 7

Collegamento software

Per realizzare, nel nostro ambiente di simulazione (ASIM), le connessioni tra l'interfaccia seriale e gli altri componenti della configurazione, dobbiamo specificare attentamente i parametri richiesti dal simulatore editando una configurazione in questo modo:

- 1) selezionare il menù **Edit** dalla finestra principale di ASIM;
- 2) scegliere quindi il comando **Add Device**;
- 3) inserire i parametri nella finestra che appare (Vedi Fig.8);
- 4) scegliere il pulsante **OK** per inserire il dispositivo nella configurazione.

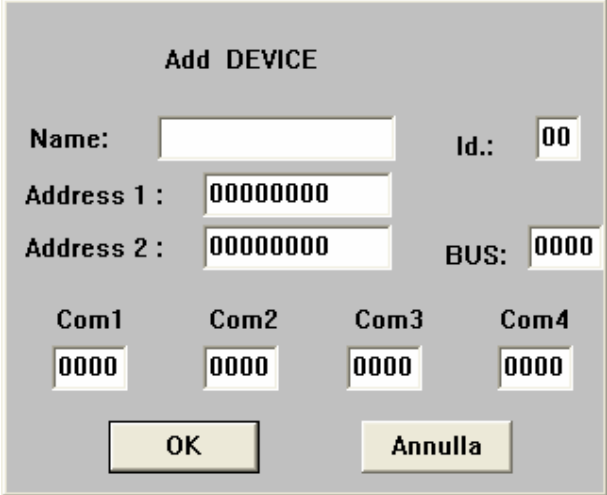


Figura 8

I parametri presenti nella finestra **Add DEVICE** ci permettono di gestire la connessione dell'interfaccia seriale con gli altri componenti della configurazione. Segue una descrizione dei singoli parametri.

- **Name:** è utilizzato per specificare il tipo di componente da inserire, nel nostro caso deve essere **"I8251USART"**;
- **Id.:** è l'abbreviazione di identify (Identificativo) e deve essere un numero compreso tra 01 ed FF (FF in decimale è 255) e viene utilizzato dal programma per riferirsi a questo dispositivo;
- **Address 1:** rappresenta l'indirizzo più basso per accedere al componente, in questo caso esso deve essere un numero pari;
- **Address 2:** definisce l'indirizzo più alto per indirizzare il componente, esso deve essere *uguale ad Address 1 + 1*;

Questi ultimi due parametri permettono di definire per quali indirizzi il decodificatore d'indirizzi attiva la linea CS (la linea CS se attivata permette l'abilitazione della periferica). Infatti, se l'indirizzo sul bus indirizzo è pari ad uno dei due, il componente viene selezionato, cioè l'operazione di lettura o scrittura è eseguita su un suo registro.

- **Com1:** determina l'identificatore del bus a cui è connesso l'interfaccia seriale. Quindi, scegliendo per Com1 l'identificatore di un componente MMU/BUS si connettono le

linee dell'interfaccia seriale: bus dati, A0, RD,WR e Reset al bus suddetto;

- **Com2:** definisce l'identificatore del gestore delle interruzioni, cioè il componente (sia esso una CPU o un P.I.C. come ad esempio: **I8259PIC**) a cui trasmettere le interruzioni;
- **Com3:** viene utilizzato per specificare l'interruzione trasmessa al gestore delle interruzioni, quando viene ricevuto un carattere in Receiver shift register ed è copiato in Data-in buffer register, per essere letto dal processore. Delle quattro cifre esadecimali che definiscono Com3 la meno significativa individua la linea d'interruzione, la seconda definisce la priorità e le due più significative specificano il "vector number" da trasmettere al processore che gestisce l'interruzione. Se le interruzioni sono gestite da un P.I.C., queste due cifre non devono essere specificate.;
- **Com4:** specifica l'interruzione inviata al gestore delle interruzioni, quando viene copiato il carattere dal Data-out buffer register in Transmitter shift register ed inizia la sua trasmissione. Le quattro cifre esadecimali hanno lo stesso significato di quelle di Com3;

E' chiaro, quindi, che nel nostro simulatore, l'insieme dei parametri **Com2**, **Com3** e **Com4**, permettono di gestire la connessione delle linee d'interruzioni **RxRDY** e **TxRDY**, con il processore o il P.I.C.

- **Com5:** se presente in altri simulatori (come ad esempio Circuit Maker 2000) viene impiegato per specificare il device a cui l'interfaccia seriale è connessa ed il tipo di connessione. Precisamente, le 2 cifre meno significative specificano l' Identificatore del Device a cui è connesso l'interfaccia seriale. La terza cifra può assumere il valore '0' per realizzare un collegamento pieno o '1' per uno parziale.;

Quindi, **Com5** consente di simulare la connessione delle **linee di handshaking**, di **Tx** e **Rx**, con un componente connettibile all' interfaccia seriale, cioè, che soddisfa le proprietà 1 e 2 descritte nella precedente pagina.

Una volta inserito il dispositivo nella nostra configurazione, scegliamo il comando **Create All** dal menù **Edit** e saranno prodotte le finestre associate ai vari Chip presenti nella configurazione editata. E' poi possibile, per agevolare la visione di tutte le finestre inerenti ai dispositivi, scegliere il comando **Tile** dal **menu Window**. La finestra associata all'interfaccia seriale è quella in figura 9.

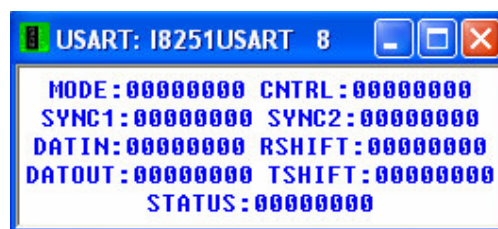


Figura 9

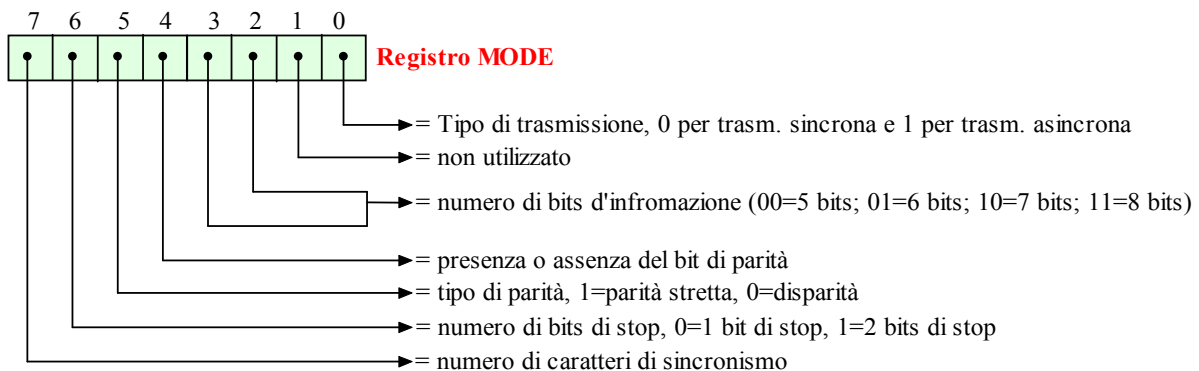
Modello di programmazione

I registri programmabili dell'interfaccia seriale sono quelli presenti in figura 9. Per accedere ad uno di questi 7 registri ad 8 bit (si sta cioè escludendo RSHIFT e TSHIFT che non sono programmabili) occorre che l'indirizzo sia uguale a Address 1 o Address 2.

La selezione dei registri in questo dispositivo avviene attraverso il bit meno significativo dell'indirizzo, in base al tipo di accesso e allo stato del sistema. Il registro che contiene le informazioni relative al genere di trasmissione che vogliamo realizzare è chiamato **MODE**. A Tale registro si può avere accesso solo in scrittura, all'indirizzo dispari e quando l'interfaccia è stata appena resettata. Il significato dei bits nel registro **MODE** sono illustrati nella tabella 2.

Tabella 2: significato dei bits nel registro MODE

bit	significato
0	determina il tipo di trasmissione , è fissato a 0 per ottenere una trasmissione sincrona e ad 1 per quella asincrona
1	non utilizzato
2/3	indicano il numero di bits d'informazione per ogni carattere trasmesso: 00 per 5 bits ,01 per 6, 10 per 7 ed 11 per 8
4	il valore 1 di questo bit abilita la presenza del bit di parità nel carattere trasmesso
5	il tipo di parità è pari se il bit ha valore 1 altrimenti è dispari
6	determina in una trasmissione asincrona il numero di bits di stop per ogni carattere trasmesso: 0 per un solo bit di stop e 1 per 2
7	in una trasmissione sincrona definisce il numero di caratteri di sincronismo che devono essere trasmessi all'inizio della trasmissione: è posto a 1 per un carattere o a 0 per due caratteri



Se con il valore inserito in **MODE** abbiamo deciso di realizzare una trasmissione sincrona (è allora necessario stabilire i caratteri di inizio e fine trasmissione), la prossima scrittura all'indirizzo dispari ci permetterà di inserire il primo carattere di sincronismo nel registro **SYNC1**, invece , qualora abbiamo deciso

che i caratteri di sincronismo devono essere due, dobbiamo effettuare una ulteriore scrittura all'indirizzo dispari per inserire il secondo carattere di sincronismo in **SYNC2**.

Dopo la scrittura nel registro **MODE**, se trasmettiamo in modo asincrono, o dopo l'inserimento del/i carattere/i di sincronismo se invece trasmettiamo in modo sincrono, ogni ulteriore accesso in scrittura all'indirizzo dispari viene eseguito nel registro **CNTRL**, il quale controlla il funzionamento dell'interfaccia seriale.

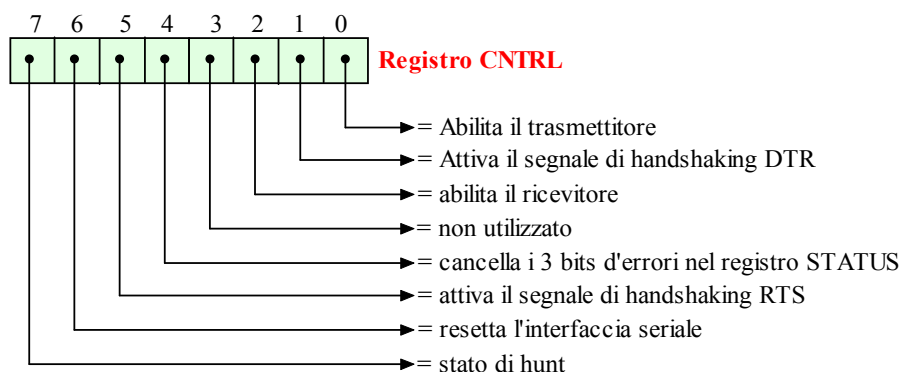
Il significato dei bits nel registro **CNTRL** sono illustrati nella tabella 3. Ovviamente, se resettiamo il componente, la sequenza di scrittura nei vari registri citati si ripete iniziando dal registro **MODE**.

Nel registro **DATIN** viene copiato il carattere che è stato ricevuto in **RSHIFT** per essere letto dal processore. **DATIN** è accessibile solo in lettura all'indirizzo pari. Un accesso in lettura in esso azzerà il bit 1 di **STATUS**.

Nel registro **DATOUT** viene scritto dal processore il carattere che deve essere copiato in **TSHIFT** per essere trasmesso. **DATOUT** è accessibile solo in scrittura all'indirizzo pari; un tale tipo di accesso azzerà il bit 0 di **STATUS**.

Tabella 3: significato dei bits nel registro CNTRL

bit	azione svolta se il bit è posto ad 1
0	abilita il trasmettitore
1	attiva il segnale di handshaking DTR
2	abilita il ricevitore
3	non utilizzato
4	cancella i 3 bits d' errori nel registro STATUS
5	attiva il segnale di handshaking RTS
6	resetta l' interfaccia seriale
7	conduce il ricevitore nello stato "hunt", in questo stato il ricevitore cerca i/il caratteri/e di sincronismo memorizzati in SYNC1 e SYNC2



I registri **RSHIFT** e **TSHIFT** sono degli shift register non accessibili, ma vengono utilizzati dall'interfaccia per effettuare rispettivamente la trasformazione del formato serie/parallelo in ricezione e parallelo/serie in trasmissione.

Nel registro **STATUS** sono contenute informazioni sull'interfaccia che possono essere utilizzate dal programma che è in esecuzione; esso è accessibile in lettura all'indirizzo pari.

I significati dei bits nel registro STATUS sono illustrati nella tabella 4.

Si riscontra un errore di parità, quando il numero di "1" nei bits d'informazione più il bit di parità non corrisponde al tipo di parità prevista.

Tabella 4: significato dei bits nel registro STATUS

<i>bit</i>	<i>informazione indicata quando assume il valore 1</i>
0	è stato copiato il carattere da DATOUT in TSHIFT ed inizia la trasmissione, questo bit si azzerà quando il processore scrive un nuovo carattere in DATOUT
1	è stato ricevuto un carattere in RSHIFT ed è stato copiato in DATIN , questo bit si azzerà quando il processore legge il dato da DATIN
2	in una trasmissione sincrona il trasmettitore non ha nessun carattere da trasmettere
3	è stato rilevato un errore di parità
4	è stato rilevato un errore di overrun
5	è stato rilevato un errore di framing
6	è stato rilevato il o i caratteri di sincronismo previsti
7	è stato attivato il segnale di handshaking DSR

Un **errore di overrun** si verifica quando viene ricevuto un nuovo dato in DATIN, prima che il precedente sia stato letto dal processore, oppure quando il processore inserisce il prossimo carattere in DATOUT, prima che venga completamente trasmesso il dato precedente.

Un **errore di framing** si riscontra quando in una trasmissione asincrona il ricevitore si aspetta di rilevare un bit di stop e, invece, rileva uno zero.

Un quadro riassuntivo sugli indirizzamenti dei vari registri è mostrato in tabella 5.

Tabella 5: indirizzamento dei registri dell' interfaccia seriale

<i>Indirizzo</i>	<i>Tipo di accesso</i>	<i>Registro</i>
Pari	R	DATIN
Pari	W	DATOUT
Dispari	R	STATUS
Dispari	W	* MODE, CNTRL, SYNC1 e SYNC2

* il registro che viene selezionato dipende dallo stato del sistema.

Quindi, riassumendo, in base allo stato del dispositivo e al solo valore del segnale A0 è possibile selezionare uno degli 8 registri del dispositivo, al registro DATAIN accedo solo in lettura per cui R\W = 0 (R\W = 0 read, R\W = 1 write) ed A0 = 0; viceversa, in DATAOUT accedo solo in scrittura e lo faccio con R\W = 1 ed A0 = 1; al registro di stato si accede in lettura per cui R\W = 0 ma stavolta A0 = 1, infine con R\W = 1 ed A0 = 1 accedo in base allo stato ad uno dei registri MODE, SYNC 1, SYNC 2 e CNTRL.

La scrittura in MODE fatta all'indirizzo dispari mi permette di accedere successivamente ai registri SYNC 1 e SYNC 2 a seconda che si scelga o meno una trasmissione sincrona o asincrona, da ora in poi ogni accesso all'indirizzo sarà riferito al registro CNTRL dalla quale si comanda il dispositivo.

Programmazione

Prima di iniziare una comunicazione, appena dopo aver resettato l'interfaccia, dobbiamo scrivere un byte all'indirizzo dispari, che ci permette di accedere al registro MODE, per determinare il tipo di comunicazione che vogliamo effettuare (sincrona o asincrona) e il formato del carattere (numero di bits d'informazione, bit di parità, tipo di parità, numero di bits di stop per comunicazione asincrona e numero di caratteri di sincronismo per comunicazione sincrona). Se la comunicazione è sincrona dobbiamo scrivere, sempre all'indirizzo dispari, i caratteri di sincronismo previsti in SYNC1 ed eventualmente in SYNC2.

I prossimi accessi in scrittura all'indirizzo dispari ci permettono di accedere in CNTRL e, quindi, di abilitare il trasmettitore e/o il ricevitore, di attivare le linee di handshaking RTS e DTR, di cancellare i bits d'errore nel registro STATUS. Inoltre se l'interfaccia, in una comunicazione sincrona, è stata abilitata a ricevere, deve essere fissato ad 1 il bit 7 di CNTRL per iniziare la ricerca dei caratteri di sincronismo.

Se la comunicazione, invece, è asincrona, dopo aver caricato MODE, eseguendo una scrittura all'indirizzo dispari si accede direttamente a CNTRL. Essa ci permette di effettuare le stesse azioni descritte per la comunicazione sincrona tranne l'inserimento nel modo "hunt".

In entrambi i casi da questo momento in poi, tutti gli accessi in scrittura all'indirizzo dispari selezionano il registro CNTRL. In ogni istante è, comunque, possibile resettare il componente ponendo ad 1 il bit 6 di tale registro.

Esempi di sequenze di inizializzazione dell'interfaccia seriale sono mostrati nelle figure 10 e 11. In entrambi gli esempi si presuppone che A0 sia stato precedentemente caricato con l'indirizzo più basso associato all'interfaccia seriale:

move.b	#\$5d,1(A0)
move.b	#\$37,1(A0)

Figura 10

In figura 10 la prima istruzione, scrivendo in **MODE 01011101**:

move.b #\$5d,1(A0)

- 0101110**1**: imposta la trasmissione asincrona per la periferica seriale;
- 0101**11**01: fissa il numero di bits d'informazione ad 8;
- 010**1**1101: abilita la presenza del bit di parità dispari;
- 0**1**011101: fissa ad 1 il numero di bits di stop;

La seconda istruzione scrivendo nel registro **CNTRL 110111**:

move.b #\$37,1(A0)

- 110111: cancella i bits d'errore nel registro **STATUS**;
- 110111: abilita il trasmettitore ed il ricevitore;
- 110111: attiva i segnali di handshaking **DTR** ed **RTS**;

Altro esempio:

move.b	#\$84,1(A0)
move.b	#\$16,1(A0)
move.b	#\$b7,1(A0)

Figura 11

La prima istruzione in figura 11, scrivendo in **MODE 10000100**:

move.b \$84,1(A0) :

- 10000100: imposta la trasmissione di tipo sincrona;
- 10000100: fissa il numero di bits d'informazione a 6;
- 10000100: disabilita la presenza del bit di parità;
- 10000100: fissa ad 1 il numero dei caratteri di sincronismo;

La seconda istruzione **move.b #\$16,1(A0)** memorizza il carattere di sincronismo nel registro **SYNC1**, il carattere scelto è pari a 16 esadecimale. L'ultima scrittura avviene nel registro **CNTRL (10110111)**, essa:

- 10110111: cancella i bits d' errore nel registro **STATUS**;
- 10110111: abilita il trasmettitore ed il ricevitore;
- 10110111: abilita la ricerca dei caratteri di sincronismo;
- 10110111: attiva i segnali di handshaking **DTR** ed **RTS**;

Questo componente permette di realizzare operazioni di I/O sia in modo programmato che attraverso interruzioni.

In entrambi i casi, se desideriamo leggere i caratteri ricevuti, dobbiamo ogni volta controllare che i bit 3, 4 e 5 (questo solo per comunicazione asincrona) del registro **STATUS** siano pari a zero, cioè che non vi sia stato nessun errore nella trasmissione del carattere.

Nell'eseguire operazioni di I/O in modo programmato deve essere continuamente esaminato il valore di uno dei due bit meno significativi di **STATUS** (Vedi tabella 4 per il significato dei bit meno significativi).

In particolare, se vogliamo trasmettere un carattere, prima di inserirlo nel registro buffer **DATOUT**, dobbiamo verificare che il bit 0 sia pari ad 1, assicurandoci, in questo modo, che il carattere precedentemente inserito in **DATOUT** sia stato già trasferito in **TSHIFT**, evitando così un errore di overrun.

Invece, quando intendiamo leggere un carattere dal registro buffer **DATIN**, dobbiamo verificare che il bit 1 sia pari ad 1, questo significa che il carattere ricevuto in **RSHIFT** è stato trasferito

in DATIN. Qualora il carattere non venga letto ed un nuovo carattere viene ricevuto, esso viene trasferito in DATAIN cancellando il carattere precedente e, quindi, causando un errore di overrun. In alternativa le operazioni di I/O possono essere realizzate con l'ausilio delle interruzioni specificate nei parametri Com3 e Com4.

La routine associata alla prima interruzione deve assicurarsi di eventuali errori nella ricezione e quindi leggere il carattere da DATIN, mentre, quella associata all'interruzione specificata in Com4 deve provvedere a scrivere in DATOUT il prossimo carattere da trasmettere.

Il comportamento

In questo paragrafo verrà descritto il comportamento del dispositivo nei vari modi di funzionamento facendo implicito riferimento che le interruzioni vengono inviate al gestore delle interruzioni specificato in Com2 e che le linee di handshaking e le linee di ricezione e trasmissione dati sono collegate al device specificato in Com5.

Dato che la frequenza del segnale di clock applicato all'interfaccia seriale è almeno 30 volte inferiore a quella applicata al processore, bisogna ridurre la frequenza di clock simulata, applicata al modulo che simula l'interfaccia seriale.

Questo può essere realizzato con le seguenti operazioni:

- 1) attivare il menù **Device** dalla finestra principale;
- 2) scegliere il comando **Speed**;
- 3) inserire un valore maggiore di uno nella finestra che appare;

Maggiore sarà il valore inserito più bassa sarà la frequenza del segnale di clock applicata all'interfaccia. Il valore inserito indica dopo quanti colpi di clock, applicati al processore, viene mandato uno all'interfaccia.

Ovviamente, i colpi di clock cui faremo riferimento successivamente saranno quelli applicati all'interfaccia seriale e non al processore.

Trasmissione asincrona

Il trasmettitore è pronto a trasmettere solo dopo la sua abilitazione e dopo aver ricevuto i segnali DSR e CTS; tuttavia, fino a quando non viene inserito un carattere in DATOUT, sarà trasmesso il valore "1" per ogni colpo di clock.

Quando scriviamo un carattere in DATOUT, viene azzerato il bit 0 di STATUS, poi al prossimo impulso di clock:

- il valore in **DATOUT** viene copiato in **TSHIFT**;
- viene fissato ad 1 il bit 0 di **STATUS**;
- viene inviata un'interruzione sulla linea specificata in **Com4**;
- infine viene trasmesso un bit '0', detto bit di start;

Successivamente per ogni impulso di clock viene trasmesso il bit 0 di TSHIFT e contemporaneamente il contenuto di questo registro viene 'shiftato' di una posizione verso destra. Questo

comportamento continua fin quando non vengono trasmessi tutti i bits d'informazione previsti in MODE. Se il carattere contiene più bits di quelli da noi previsti, allora i bits in eccesso non saranno presi in considerazione.

Il prossimo bit ad essere trasmesso, se esso è stato richiesto in MODE, è il bit di parità, con il giusto tipo di parità, altrimenti viene trasmesso un bit '1' citato come bit di stop. Se in MODE sono stati fissati due bits di stop, al prossimo colpo di clock verrà trasmesso un'altro '1' riportandoci nello stato iniziale.

Se nel frattempo non è stato inserito un nuovo carattere in DATOUT, sarà trasmesso un '1' per ogni successivo colpo di clock.

Trasmissione sincrona

Dopo aver impostato la trasmissione come di tipo sincrona , dopo aver abilitato il trasmettitore e ricevuto i segnali DSR e CTS, il trasmettitore è pronto a trasmettere, ma fin quando non viene inserito un carattere in DATOUT, verrà posto ad 1 il bit 3 di STATUS e saranno trasmessi in continuazione i caratteri di sincronismo.

Precisamente il prossimo impulso di clock produce le seguenti azioni:

- copia del valore di **SYNC1** in **TSHIFT**;
- trasmissione del bit 0 di **TSHIFT**;
- shift verso destra di una posizione del valore in **TSHIFT**;

Successivamente per, ogni impulso di clock, vengono ripetute le ultime due operazioni fino alla completa trasmissione degli otto bit del primo carattere di sincronismo. Se deve essere trasmesso anche il secondo carattere di sincronismo, tutte le operazioni sono ripetute, con la differenza che in TSHIFT viene caricato SYNC2 invece che SYNC1.

Se, nel frattempo, non è stato caricato in DATOUT nessun carattere, viene posto ad 1 il bit 3 di STATUS e vengono trasmessi di nuovo i caratteri di sincronismo previsti. In generale questo avviene ogni volta che il trasmettitore viene a trovarsi nella condizione di *underrun*.

Quando scriviamo un carattere in DATOUT viene azzerato il bit 0 di STATUS e al prossimo impulso di clock viene:

- copiato il valore di **DATOUT** in **TSHIFT**;
- posto ad 1 il bit 0 di **STATUS**;
- azzerato il bit 3 di **STATUS**;
- inviata un' interruzione sulla linea specificata in **COM4**;
- trasmesso il bit 0 di **TSHIFT**;
- shiftato verso destra di una posizione il valore in **TSHIFT**;

Le due ultime operazioni vengono ripetute ad ogni impulso di clock fin quando non vengono trasmessi tutti i bits d' informazione previsti in MODE. Anche in questo caso, se il carattere contiene più bits di quelli da noi previsti, i bits in eccesso non saranno presi in considerazione.

Il prossimo bit ad essere trasmesso, se esso è stato richiesto in MODE, è il bit di parità, con il giusto tipo di parità.

Se, nel frattempo, non è stato inserito un nuovo carattere in DATOUT, il trasmettitore viene a trovarsi nella condizione di *underrun* e quindi verrà posto ad 1 il bit 3 di STATUS e verranno trasmessi di nuovo i caratteri di sincronismo previsti.

Ricezione asincrona

Quando il ricevitore è stato abilitato ed è stato attivato il segnale DSR, esso è pronto a ricevere.

Se viene ricevuto un dato sulla linea RX, esso è inserito sempre nel bit più significativo di RSHIFT, dopo che quest'ultimo è stato shiftato di una posizione verso destra.

Il ricevitore inizia a considerare i bits in arrivo come bits d'informazione solo quando riceve il bit di start.

Una volta ricevuto l'ultimo bit d'informazione del carattere trasmesso, il valore in RSHIFT viene shiftato di un numero di posizioni verso destra pari a otto meno il numero di bits d'informazioni trasmesso, in modo da spostare il carattere al limite destro di RSHIFT.

A questo punto viene copiato il dato di RSHIFT in DATIN e se il bit 1 di STATUS è uno, cioè se il carattere ricevuto prima non è stato letto dal processore, viene segnalato un errore di *overrun* ponendo il bit 4 di STATUS ad uno.

Il prossimo bit ricevuto se presente è il bit di parità, se viene scoperto un errore di parità, esso viene segnalato ponendo ad uno il bit 3 di STATUS.

Successivamente il ricevitore si aspetta uno o due bits di stop, se, invece, viene ricevuto uno zero, esso segnala un errore di *framing* ponendo ad uno il bit 5 di STATUS.

Quando viene ricevuto l'ultimo bit di stop viene posto ad uno il bit 1 di STATUS ed è inviata un' interruzione del tipo specificata in COM3.

Quando poi il processore preleva il dato da DATIN il bit 1 di STATUS viene azzerato.

Ricezione sincrona

Anche in questo caso il ricevitore è pronto a ricevere, se esso è stato abilitato ed è stato attivato il segnale DSR.

Se viene ricevuto un dato, esso è inserito sempre nel bit più significativo di RSHIFT dopo che quest'ultimo è stato shiftato di una posizione verso destra.

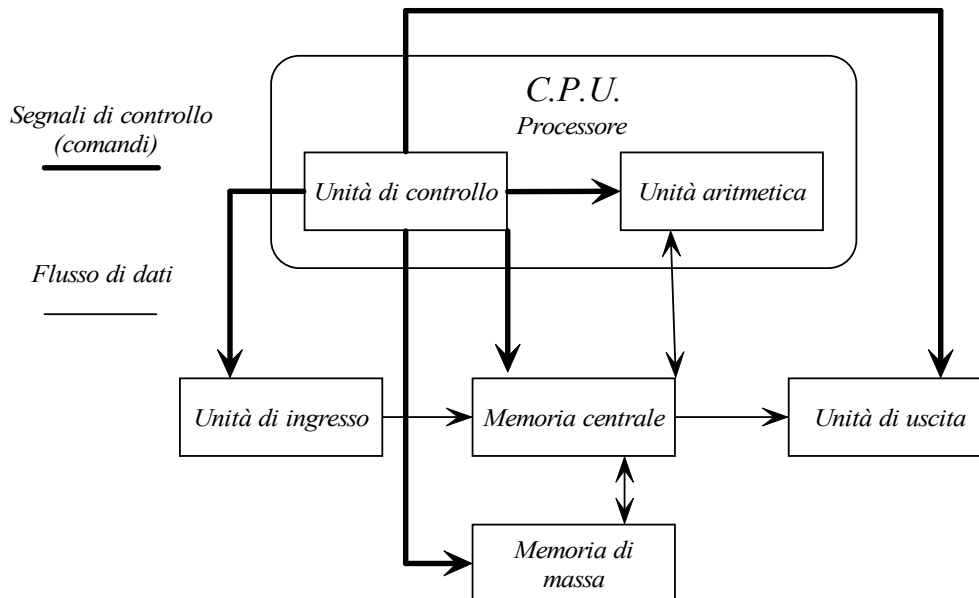
Le differenze con la ricezione asincrona sono che non esistono né bit di start, né bit di stop e che, se il ricevitore è nello stato *hunt*, esso ricerca non i bits d'informazione ma i caratteri di sincronismo. Quando riconosce i caratteri di sincronismo memorizzati all' inizio in SYNC1 ed eventualmente in SYNC2, esso azzerà il bit7 di CNTRL, il quale fa terminare la ricerca, e pone ad uno il bit 6 di STATUS.

Una volta ricevuto tutti i bits d'informazione e l'eventuale bit di parità e dopo aver segnalato i possibili errori di *overrun* e di *parità*, in modo identico a quanto visto per il ricevitore asincrono, viene posto ad uno il bit 1 di STATUS ed è inviata un' interruzione del tipo specificata in COM3. Anche per la ricezione sincrona, quando il dato è prelevato dal processore da DATIN, viene azzerato il bit 1 di STATUS.

Lezione N°15: "Le memorie"

La memoria svolge un ruolo determinante all'interno di un calcolatore al punto da condizionarne le prestazioni. I dati che un programma elabora in fase di esecuzione vengono tipicamente mantenuti nella memoria principale di un calcolatore e sono restituiti al processore solo quando richiesti da un programma in corso.

Osservando lo schema ideale di un calcolatore, quello di Von Neumann qui sotto riportato:



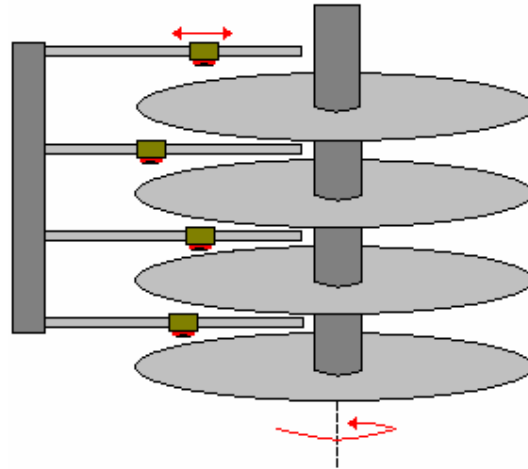
verrebbe da pensare ad un'unica memoria di sistema, uguale per tutti i componenti del calcolatore, tuttavia la velocità di esecuzione dei programmi dipende molto dalla velocità con cui le istruzioni e i dati vengono trasferiti fra la CPU e la memoria principale.

Ecco allora che si decide di realizzare diversi tipi di memoria, alcuni di questi molto veloci altri invece non necessariamente richiedono un'elevata velocità e sono quindi realizzati con diverse tecnologie. La memoria centrale di un calcolatore a cui si fa riferimento nello schema di Von Neumann è quella classica che va sotto il nome di memoria RAM (Random Access Memory, ossia memoria ad accesso casuale)

La caratteristica RAM non è vera per tutti i tipi di memoria: essa consente l'accesso al dato memorizzato alla quale corrisponde un indirizzo per l'identificazione, inoltre l'accesso è detto casuale (random) poiché indipendente dalla collocazione del dato. Difatti, la disponibilità di un dato situato in memoria è all'incirca la stessa per tutti gli altri dati.

Ben diversa è invece la situazione per un hard disk che è pur sempre inquadrato nella categoria delle memorie (dette tipicamente memorie di massa) ma necessita di un tempo di caricamento che cambia da dato a dato. Come si nota dalla figura, la struttura di

un hard disk prevede la disposizione di più piatti magnetici uno sopra l'altro e diverse testine per la lettura e la scrittura:



Per individuare informazioni su un disco è necessario conoscere su quale piatto esso è stato scritto, non solo, ma anche su quale delle due superfici del piatto, su quale traccia e su quali settori. I tempi di accesso per i dischi di tipo magnetico dipendono da:

$$\text{Tempo di accesso} = \text{Tempo medio di ricerca} + \text{Latenza media}$$

Per tempo medio di ricerca intendiamo quell'intervallo di tempo necessario allo spostamento radiale delle testine verso il settore da leggere. Per tempo di latenza media s'intende invece la rotazione che il piatto deve compiere fino al raggiungimento del settore voluto. Nel corso di queste note ci occuperemo essenzialmente delle memorie RAM, dispositivi fondamentali per le prestazioni dell'sistema.

Alcune intuizioni logiche portano a pensare che una memoria debba essere contemporaneamente grande, veloce e ovviamente poco costosa. Tuttavia all'aumentare dei parametri che ne caratterizzano le prestazioni corrisponde un inevitabile innalzamento del costo. Per evitare di costruire memorie dall'elevato costo sono state sviluppate diverse strutture che migliorano la velocità e la capacità della memoria mantenendo ragionevole il costo. In base a quanto detto prima, i parametri che caratterizzano una memoria sono:

- **Dimensione:** è il numero di byte che la memoria è in grado di memorizzare, spesso volte questa informazione si accompagna ad altre che ne specificano la struttura. Ad esempio se è riportata la voce 64K x 4 bit si vuole intendere che la memoria si compone di 65536 celle ciascuna di 4 bit, per cui abbiamo che una parola è di 4 bit;
- **Velocità:** questo parametro è indicativo della rapidità con la quale la memoria effettua i prelievi dei dati e li rende disponibili al processore. Un primo parametro legato alla velocità della memoria è il *tempo di accesso alla memoria* ed è costituito dall'istante di tempo che trascorre tra

l'attivazione del segnale Read (oppure Write) e il segnale MFC che indica appunto il completamento della lettura del dato (oppure della scrittura). Un altro parametro legato alla velocità è invece il *tempo di ciclo della memoria*, ed è il tempo minimo che trascorre tra due operazioni successive di accesso alla memoria. Il tempo di ciclo è generalmente più alto del tempo di accesso in memoria.

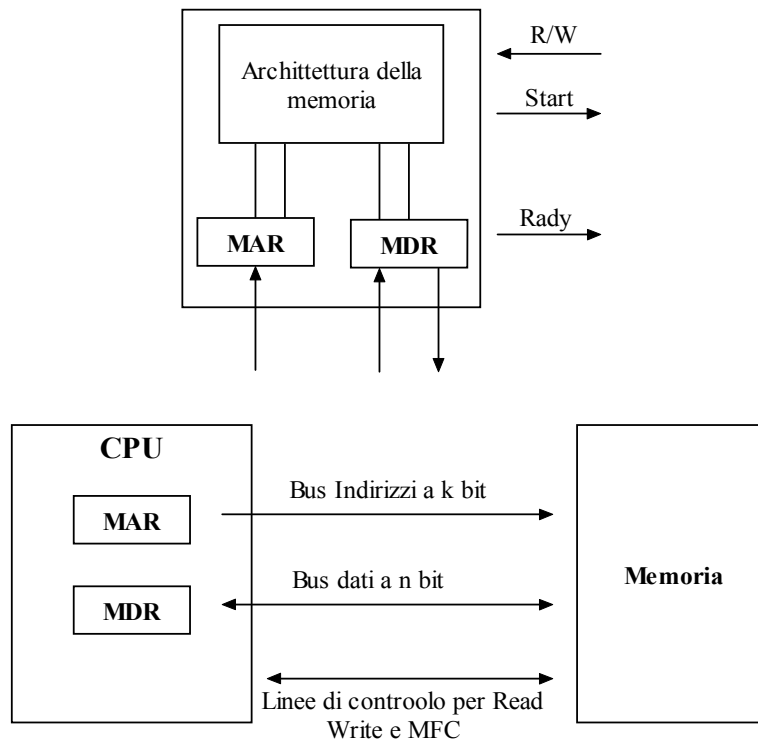
- **Altri parametri:** non è possibile prescindere dal costo della realizzazione, per questo motivo appartiene a questa classe il costo che il dispositivo di memoria richiede, in alcuni casi la realizzazione risulta essere impossibile proprio per gli elevati costi che il dispositivo di memoria può richiedere, per cui le scelte, come già detto in precedenza, sono influenzate da questo fattore ma hanno dato anche un contributo in modo positivo alla ricerca di nuove tecnologie più innovative quali: memoria cache, memoria virtuale e interlacciamento della memoria. Anche la potenza dissipata è un parametro da tenere sotto controllo;

Ogni calcolatore può utilizzare una memoria massima di dimensione pari a due elevato il numero di bit che è in grado di indirizzare. Un calcolatore con indirizzi di 16 bit può allora indirizzare $2^{16}=64K$ locazioni di memoria. La memoria può essere organizzata nella modalità byte-addressable, l'indirizzamento avviene cioè byte per byte, ed è inoltre possibile seguire lo schema big-endian o quello little-endian, a seconda che l'assegnazione dell'indirizzo di memoria meno significativo venga corrisposto alla cella di memoria più a sinistra o più a destra del dispositivo. Il processore colloquia con la memoria attraverso l'utilizzo di due registri situati all'interno della CPU e che sono:

- Registro **MAR**, **Memory Address Register**;
- Registro **MDR**, **Memory Data Register**;

I registri MAR e MDR costituiscono allora l'interfaccia del dispositivo di memoria. Il registro MDR è il registro contenente dati della memoria. Il registro MAR è il registro della CPU contenente gli indirizzi della memoria.

Entrambi sono collegati alla memoria principale da appositi bus (control bus e Add bus), il bus comprende le linee di controllo per la lettura (Read), la scrittura (Write) e il segnale di completamento della funzione di memoria detto MFC (Memory Function Completed).



Per gestire la memoria occorre che il processore sia in grado di comunicare attraverso l'interfaccia generando i segnali che ne permettono il corretto funzionamento. In realtà, il controllo della memoria vera e propria è codificata in hardware per cui i segnali non sono controllati via software ma vengono delegati, da parte del processore, ad alcuni circuiti.

I segnali che servono alla comunicazione tra memoria e CPU sono:

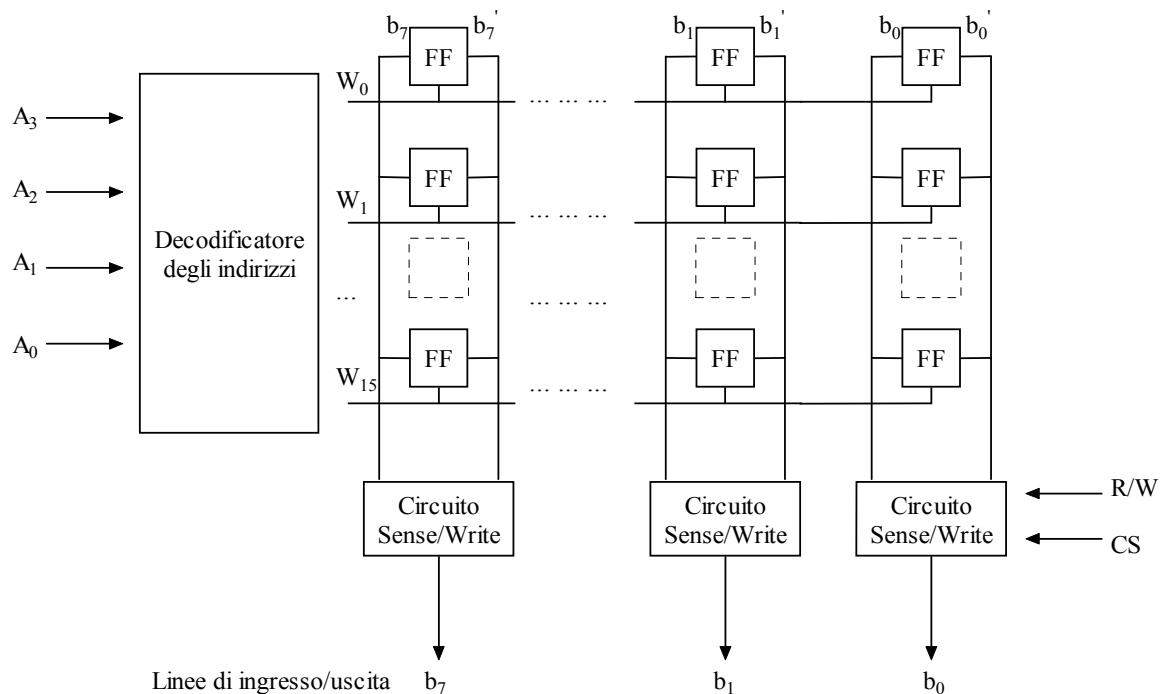
- **R/W** (segnali di Read e Write), è il segnale che controlla le operazioni di lettura e scrittura ed è generalmente comandato con un unico segnale, il valore alto indica un'operazione di lettura viceversa per quello basso;
- **Start**, è il segnale che individua l'inizio delle operazioni di lettura o scrittura, in realtà i segnali che compaiono sono solo R/W, questo e il successivo segnale servono solo per comprendere l'inizio e la fine del ciclo;
- **Ready**, è il segnale significativo di un ciclo di operazioni conclusosi con esito positivo;

E' abbastanza logico collegare il lavoro che svolge una memoria e quello svolto da un flip-flop, entrambi hanno in comune le operazioni di lettura e scrittura dati, ovviamente il flip-flop si limita alla lettura e scrittura dati di un singolo bit, diversamente per le memorie che operano invece su più di un bit. Il flip-flop costituisce di fatto l'elemento portante di tutte le memorie che sono infatti realizzate da un certo numero di flip-flop per quanti sono i bit massimi che la memoria deve ricordare.

Se ad esempio abbiamo una memoria di 32Mbyte, abbiamo anche $32 \times 10^6 \times 2^8$ differenti bit, l'unico modo di trovare quelli giusti è allora quello di etichettare ciascun bit associandolo ad un numero

codificato che permette la sua individuazione in maniera univoca e senza ambiguità. La macchina che realizza questo sistema è il decodificatore, poiché i fili che ci occorrono sono molti (ne abbiamo infatti uno per ciascuno flip-flop) si deve allora realizzare una composizione di decodificatori (è estremamente complesso realizzare un decodificatore che abbia in ingresso molti fili per la selezione del dato), quindi i dati che si inseriranno nel registro MAR saranno poi elaborati dal decodificatore che ci consentirà di individuare i bit che si vogliono leggere o scrivere (dipende dalla modalità di accesso).

Le celle di memoria vengono organizzate secondo una struttura ad array, ogni cella di memoria può memorizzare un bit di informazione:



Tutte le celle di memoria e quindi tutti i flip-flop di una riga sono collegati ad una linea comune che determina la lunghezza di parola della memoria e che è detta appunto linea di parola. Questa linea è collegata al decodificatore degli indirizzi.

Le celle di memoria appartenenti alla stessa colonna (quindi tutti i flip-flop di una stessa colonna) sono invece collegati ad un circuito Sense/Write (rileva/scrivi) mediante due linee di bit. In una operazione di lettura questi circuiti leggono i bit memorizzati nelle celle di memoria, durante un operazione di scrittura i circuiti Sense/Write ricevono come ingresso un dato e scrivono quest'ultimo all'interno della cella di memoria.

Nella figura precedente, che mostra 16 parole (16 è il numero di righe) di 8 bit (8 bit è il numero di colonne) è possibile memorizzare 128 bit (l'equivalente del prodotto 16×8), la struttura mostrata prima è nota anche come "organizzazione 16×8 ".

Questa struttura richiede, per la comunicazione con l'esterno, 14 pin, alla quale si devono aggiungere altri 2 pin per l'alimentazione del dispositivo, per cui in totale abbiamo 16 pin, di cui 8 sono i fili che vanno da b_7 a b_0 e che corrispondono alle linee di ingresso e di uscita, 4 sono i fili che vanno da A_3 a A_0 e che sono in ingresso al decodificatore degli indirizzi, e 2 pin sono associati ai segnali R/W e CS che seleziona un determinato chip.

Altra struttura nota è quella 128×8 che è quindi dotata di 1024 celle di memoria e che adopera oltre ad un decodificatore a 5 bit, due multiplexer che prendono in ingresso i diversi fili in uscita dai circuiti Sense/Write.

Se guardiamo il circuito della figura che mostra la struttura 16×8 , notiamo che molti fili sono stati uniti, questo causerebbe una condizione di indeterminazione poiché la logica che abbiamo usato fin ora era basata su due valori logici, in realtà per realizzare una serie di dispositivi viene utilizzato il meccanismo della **logica three state**, viene cioè aggiunto un terzo stato detto neutro. Tutto ciò avviene poiché si vogliono realizzare dei circuiti che hanno dal punto di vista tecnologico un'alta impedenza in uscita, per cui se si trovano in una certa condizione sono in grado di non influenzare il filo a cui sono connessi. Questa logica è tipicamente applicata alla realizzazione dei BUS.

Quando le dimensioni della memoria aumentano (gli esempi di strutture 16×8 oppure 128×8 sono utili a comprendere lo schema che le memoria possono adottare) è necessario gestire tutto il sistema in modo più efficace.

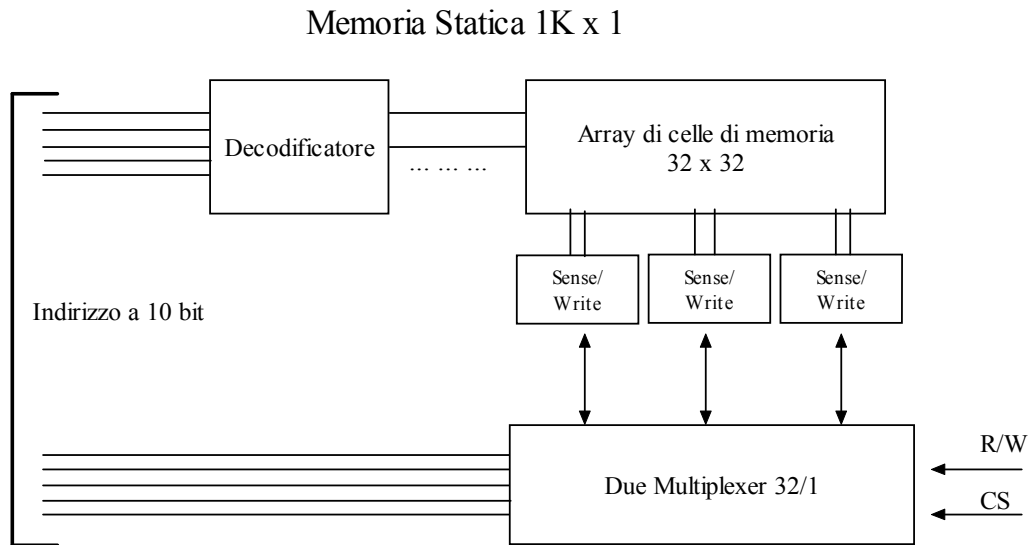
La realizzazione delle memorie segue differenti tecniche che possiamo suddividere in due famiglie:

- memorie statiche (SRAM);
- memorie dinamiche (DRAM);

Le memorie statiche SRAM sono le più performanti ed hanno la caratteristica di essere le più veloci, ma sono molto piccole dal punto di vista capacitivo. Sono dette memorie statiche tutte quelle memorie capaci di mantenere il loro stato durante tutto il periodo in cui sono alimentate. Affinché la memoria mantenga il suo stato è necessario alimentare il dispositivo in continuazione, interrompendo l'alimentazione si perde il contenuto della memoria.

Le memorie SRAM sono veloci ma anche più costose a causa dell'elevato numero di transistor necessari per realizzare la cella.

Schema di una memoria statica:



Consideriamo un array di celle di memoria di 32×32 , il che significa che abbiamo 32 dispositivi e 32 fili in ingresso, il decodificatore dell'indirizzo stabilirà qual è la riga giusta, in ingresso avrà bisogno di $\log_2 32 = 5$ bit, con 5 valori in ingresso si estraggono 2^5 possibili uscite che devono ovviamente essere tutte 0 tranne una che corrisponde alla riga da attivare.

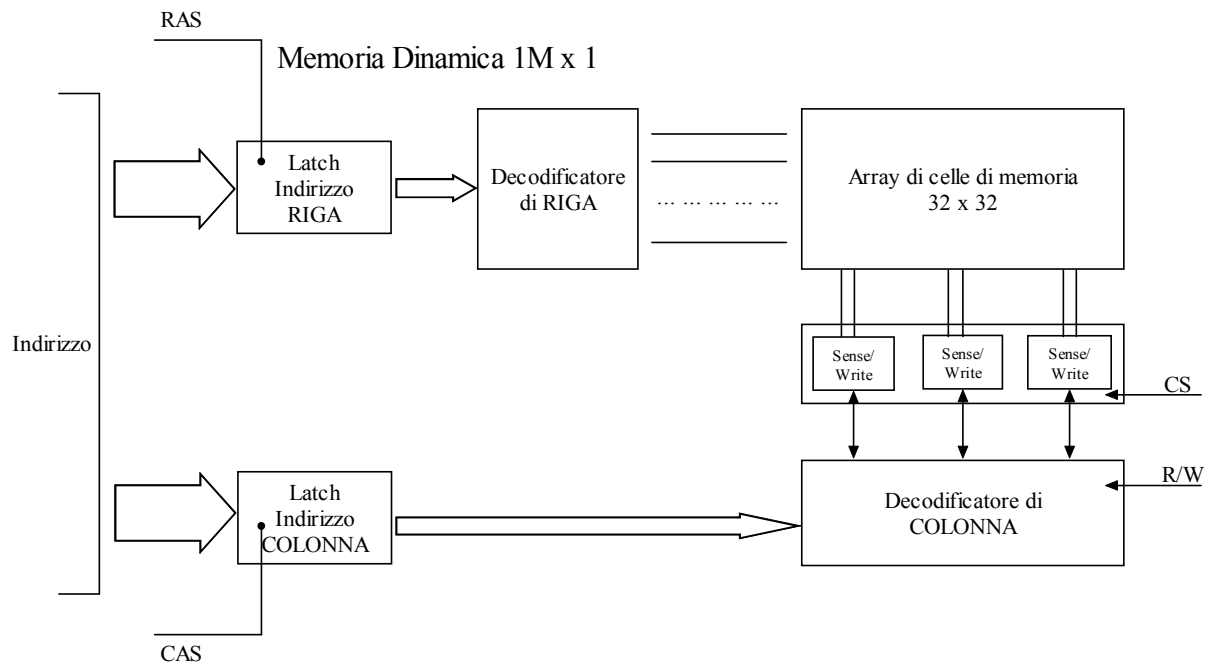
Il risultato sarà una serie di 32 bit memorizzati in questi circuiti, il multiplexer (in realtà c'è ne sono 2) ha in ingresso $\log_2 32 = 5$ bit che corrispondono all'ingresso indirizzi, 32 bit che corrispondono all'ingresso dati. Dai 5 ingressi di indirizzo ne estrae uno solo che è la colonna dell'array ed individuando con la riga indicata dal decodificatore il contenuto di una delle 1024 celle differenti che compongono il vettore.

Questa costituisce la soluzione che di fatto viene adottata e trasferisce al dispositivo un'elevata velocità poiché si sono impiegati tutti i fili che si avevano a disposizione, senza cioè ricorrere ad una semplificazione del dispositivo.

Le memorie dinamiche o DRAM costituiscono l'alternativa alle SRAM, esse sono meno costose ed utilizzano celle più semplici da realizzare dal punto di vista elettronico, tuttavia le celle possono mantenere il dato memorizzato solo per un tempo indefinito.

L'informazione presente in una memoria dinamica è memorizzata sotto forma di carica di un condensatore, per questo motivo l'informazione dura solo pochi milli-secondi. Una situazione del genere sarebbe inaccettabile poiché di scarsa utilità, tuttavia si realizzano alcuni dispositivi che periodicamente "rinfrescano" la carica dei condensatori mantenendo vivi i dati memorizzati.

Schema di una memoria dinamica:



Consideriamo sempre un array di celle 32 x 32, l'indirizzo viene suddiviso in due blocchi, viene fornito un indirizzo di riga ed uno di colonna, abilitati da due segnali differenti il RAS (Row address strobe) e il CAS (Columns address strobe).

Durante un operazione di lettura o di scrittura, viene prima applicato l'indirizzo di riga fornendo quindi il segnale RAS e ottenendo la riga, in seguito si fornisce il segnale CAS che abilita il decodificatore di colonna e seleziona uno solo dei 32 bit di uscita.

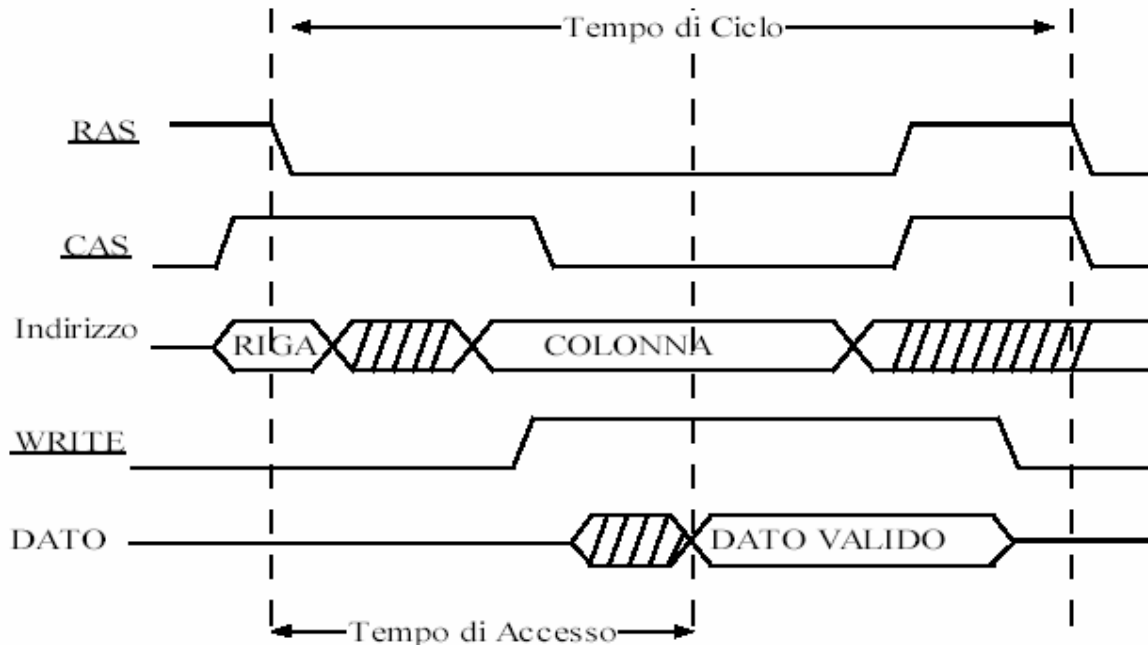
Usando questo metodo non occorrono più due decodificatori differenti poiché è possibile usarne uno solo abilitandolo prima per un segnale e poi per l'altro. Tuttavia il progetto è stato sviluppato in questo modo poiché quando diamo il segnale di RAS tutta la riga è portata nei dispositivi Sense/Write che sono abilitati a fornire l'uscita leggendo inoltre tutti i dati che vengono quindi nuovamente "rinfrescati".

Quando si applica un indirizzo di riga, tutte le celle della riga corrispondente vengono lette e rinfrescate durante le operazioni di lettura e scrittura. Per garantire che il contenuto delle DRAM sia mantenuto valido si deve allora accedere periodicamente ad ogni riga di celle, tipicamente l'accesso avviene dopo un intervallo di tempo che va dai 2 ai 16ms.

I circuiti di rinfresco, detti refresh circuit, svolgono questa funzione in modo indipendente e invisibile all'utente che non s'accorge dei continui accessi in memoria. Un aspetto importante da considerare è il fatto che l'indirizzo dei riga è caricato nel latch una sola volta (per ogni accesso ad una riga) mentre l'indirizzo di colonna, anch'esso caricato nel latch, non è sempre lo stesso ma cambia ad ogni ciclo successivo fino a scorrere tutta la riga.

Le SRAM sono usate maggiormente qualora il requisito fondamentale della macchina debba essere l'alta velocità di funzionamento, mentre le DRAM trovano facilmente impiego e costituiscono la scelta predominante per realizzare la memoria principale di un calcolatore.

Il ciclo di funzionamento di questi dispositivi è sviluppato in questo modo:



NOTA: i segnali RAS, CAS e Write sono negati.

Nell'interfaccia avevamo inserito altri due segnali, quello di start e quello di ready, che in questo caso non consideriamo.

In realtà abbiamo cercato di considerare un meccanismo del tutto generale in cui i segnali sono gestiti in modo differente a seconda del dispositivo. In questo caso consideriamo i segnali RAS e CAS anziché quelli di start e ready. Il dispositivo opera sui fronti di discesa dei segnali. Osservando il diagramma si nota infatti che l'operazione di Write assunta come esempio ha inizio solo quando il segnale RAS compie una discesa sul proprio fronte.

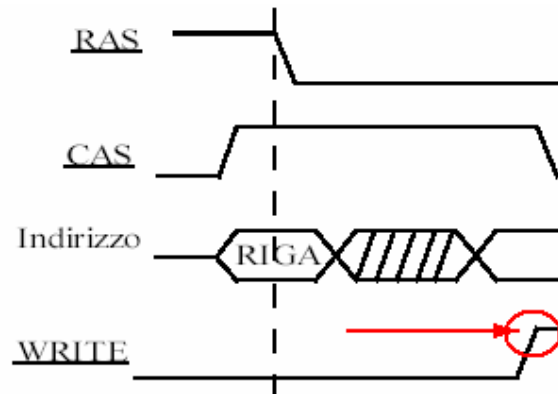
Quindi: il segnale RAS (Rows address strobe) compie una variazione sul fronte di discesa, le operazioni di lettura o scrittura possono cominciare a patto che il dato sia disponibile sul bus. Al fronte di discesa del RAS la memoria:

- Carica nel latch l'indirizzo di riga;
- l'indirizzo viene fornito al decodificatore di riga;
- il decodificatore di riga seleziona l'array costituito dalle celle di memoria richieste (vengono selezionati i 32 bit);

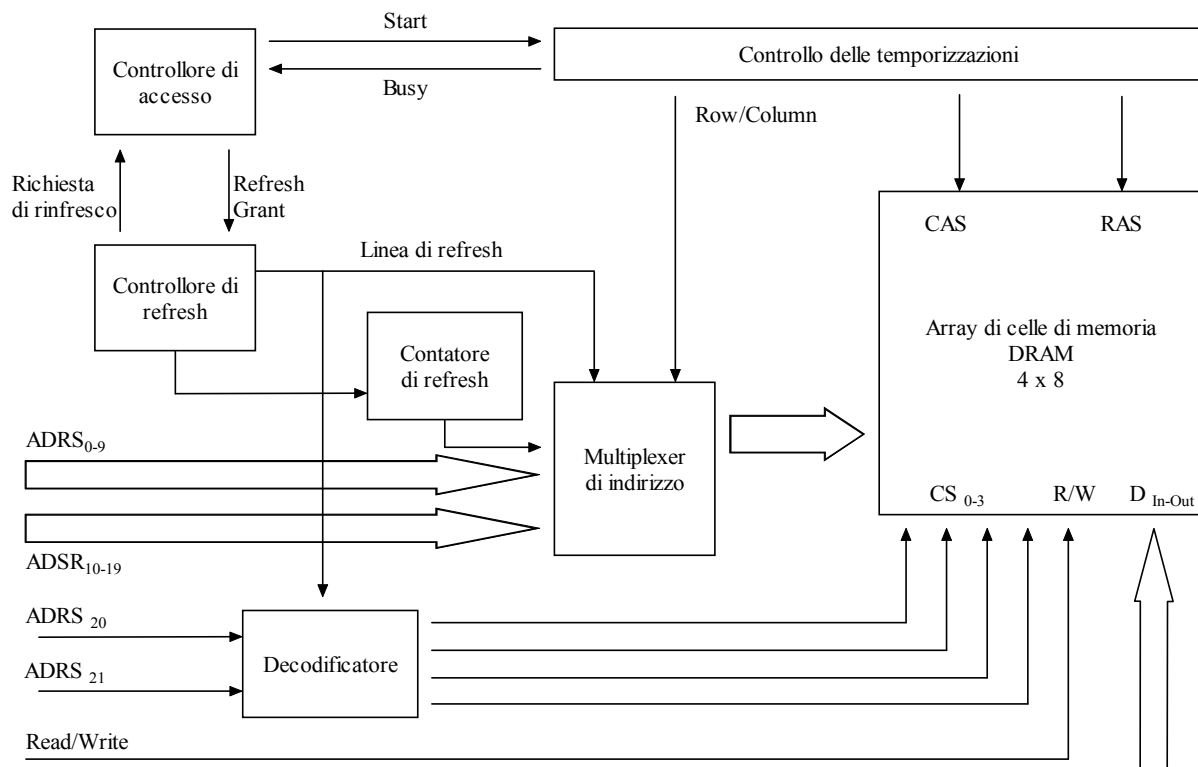
Le operazioni fin qui fatte si sono svolte senza tener conto di quale operazione si dovesse poi fare successivamente (sia essa di Write o di Read, lettura o scrittura), infatti al dispositivo di

memoria l'intenzione di un eventuale lettura o scrittura viene comunicata solo dopo il segnali di RAS e prima che avvenga il fronte di discesa del segnale di CAS.

Al fronte di discesa del segnale CAS (Columns address strobe) il dispositivo è in grado di individuare il giusto bit. Osservando nuovamente il ciclo operativo delle DRAM notiamo che ai fronti di discesa dei segnali RAS e poco prima che avvenga anche quello del segnale CAS segue la richiesta di Read. Il dato sarà allora letto sul fronte di discesa del segnale CAS:



Il tempo di accesso inizia con la variazione sul fronte di discesa del segnale RAS e termina non appena il dato è disponibile. Quando il tempo di accesso si conclude il dato può essere preso e memorizzata dal processore, per individuare il dato il dispositivo ha bisogno di un nuovo fronte nel quale i segnali di RAS e di CAS si alzano.



Dal libro Hamacker, Introduzione all'architettura dei calcolatori:

Si consideri ora l'operazione di rinfresco della memoria. Il blocco di controllo di Refresh genera periodicamente le richieste di rinfresco, facendo sì che il blocco di controllo dell'accesso inizi un ciclo di memoria in modo normale.

Il blocco di controllo dell'accesso indica al blocco di controllo di Refresh che può procedere con una operazione di rinfresco attivando la linea Refresh Grant.

Il blocco di controllo dell'accesso fa da arbitro tra le richieste di accesso alla memoria e di rinfresco. Se le richieste arrivano contemporaneamente, la priorità viene accordata alla richiesta di rinfresco. Onde evitare che il contenuto delle celle possa andar perso.

Non appena il blocco di controllo di Refresh riceve il segnale Refresh Grant attiva la linea di Refresh. Questo fa sì che il multiplexer dell'indirizzo selezioni il contatore di Refresh come sorgente per l'indirizzo di riga invece delle linee degli indirizzi esterne $ADRS_{19-10}$. Quindi nel momento in cui il segnale RAS è attivato, il contenuto del contatore viene caricato nei latch dell'indirizzo di riga di tutti i chip della memoria. Durante tale periodo, la linea R/W del bus di memoria può indicare un'operazione di scrittura, perciò è necessario garantire che questo non provochi inavvertitamente il caricamento di nuovi dati nelle celle di memoria che in quell'istante vengono rinfrescate.

Ciò può essere fatto in diversi modi. E' possibile per esempio far sì che il blocco di decodifica disabiliti tutte le linee CS per impedire che i chip di memoria rispondano alla linea R/W. Il resto del ciclo di rinfresco è simile a un ciclo normale.

Alla fine, il blocco di controllo di Refresh incrementa il contatore di Refresh in preparazione al ciclo di rinfresco successivo.

Lo scopo principale del circuito di rinfresco è quello di mantenere l'integrità delle informazioni memorizzate. Teoricamente, la sua presenza dovrebbe essere invisibile al resto del sistema di calcolo: cioè le altre parti del sistema, come ad esempio la CPU non dovrebbero essere influenzate dall'operazione di rinfresco. In effetti, però, la CPU e il circuito di rinfresco sono in competizione per l'accesso alla memoria. Al circuito di rinfresco spetta una maggiore priorità rispetto alla CPU per garantire che non vengano perse informazioni. Quindi la risposta della memoria ad una richiesta proveniente dalla CPU, o dal dispositivo di accesso diretto alla memoria (DMA), può essere ritardata nel caso sia in atto un'operazione di rinfresco.

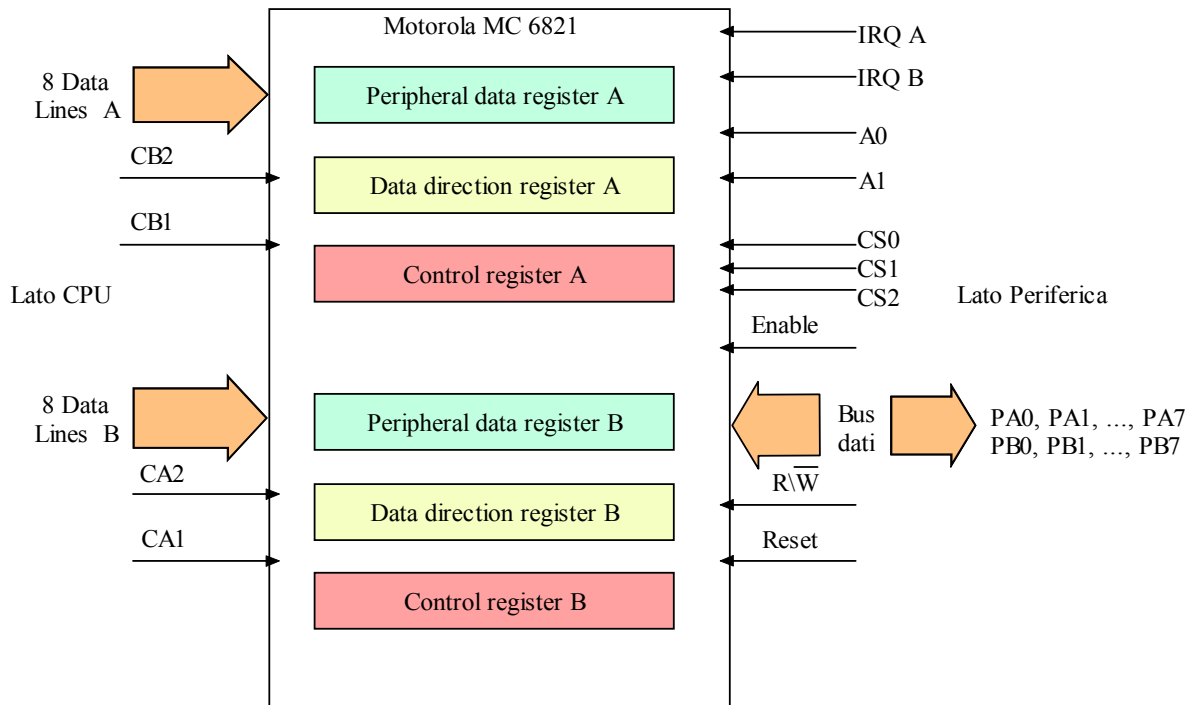
Il rinfresco provoca un degrado delle prestazioni delle memorie DRAM, quindi è necessario minimizzare il tempo complessivo dedicato a tale operazione. Se l'array di memoria è costituito da chip di $1M \times 4$. Ogni chip contiene un array di celle organizzato in $1024 \times 1024 \times 4$ il che significa che ci sono 1024 righe con 4096 bit per riga!!! Ci vogliono 130ns per rinfrescare una riga, e ogni riga deve essere rinfrescata ogni 16ms. Quindi il tempo necessario per rinfrescare tutte le righe del chip è $133\mu s$ (l'equivalente di 1024 righe \times 130nS tempo per rinfrescare una riga).

Poiché tutti i chip vengono rinfrescati simultaneamente, il costo dell'operazione di rinfresco per l'intera memoria è di $130/16000 = 0.0081$. Di conseguenza meno dell'1% dei cicli di memoria disponibili.

Lezione N°16: "La comunicazione parallela"

In realtà un'interfaccia di tipo parallelo ha molte semplificazioni rispetto ad una di tipo seriale, dal punto di vista strettamente realizzativo. Ciò non significa che da un punto di vista della configurazione il dispositivo sia più facile da gestire, al contrario esso risulta più complicato.

Il dispositivo, così come quello seriale, può essere visto da due lati, il lato che va verso il processore e che permette il dialogo con il dispositivo, e il lato che interfaccia i dispositivi.



La prima cosa che subito si nota è che questo dispositivo è pressoché duplicato, abbiamo all'interno del dispositivo due interfacce di tipo parallelo che sono perfettamente duplicate e che possono essere utilizzate in modo indipendente l'una dall'altra. Sono disponibili per la comunicazione sei registri, tutte le linee della comunicazione parallela possono essere utilizzate sia in ingresso che in uscita. I segnali che si hanno a disposizione sono:

- Data bus D0, D1, ..., D7 sono le linee di comunicazione utilizzate sia in ingresso e in uscita, ma non possono essere usate contemporaneamente in entrambe le direzioni, volendo effettuare una comunicazione seriale dobbiamo allora dedicare 4 di questi bit ai segnali DTR, DSR, RTS e CTS;
- Segnali di selezione del chipset (CS0, CS1 e CS2);
- Segnale di lettura\scrittura;
- Segnale enable, abilitazione;
- 2 Data bus periferiche ad 8 bit (PA0, PA1, ..., PA7, PB0, PB1, ..., PB7);
- 2 linee di interruzione esterne (CA1, CA2);
- 2 linee di controllo periferico (CA2, CAB2), programmate per funzionare come In\Out;
- 6 Registri interni (2 segnali RS0, RS1 per indirizzarli);

Le linee PA0,PA1,...,PA7,PB0,PB1,...,PB7 possono essere sia di ingresso che di uscita, la direzione dei dati si programma settando i bit nei registri direzione dei dati, chiamati anche DDR, ad esempio se DDRA = 11111111, allora tutte le linee di A sono state settate come linee di uscita.

Le linee di comunicazione D0,D1,...,D7 consentono il trasferimento dei dati dalla PIA al processore, queste linee sono di tipo three state, come la logica usata nelle memorie RAM. La direzione dei dati dipende dal segnale R\W.

I segnali delle interruzioni a disposizione sono due, IRQA ed IRQB e sono associati rispettivamente alla porta A ed alla porta B, queste linee sono di tipo Open drain.

Quando si verifica una interruzione ci sarà una routine SW che legge e controlla tutti i registri di controllo ed in particolare i bit 6 ed il bit 7 che permettono di gestire le interruzioni, ed eventualmente di mascherarle. Quando la CPU legge i dati dai registri, cancella automaticamente il segnale di richiesta di interruzione (disabilita cioè i bit interessati all'interruzione).

I segnali CA1,CB1, CA2, CB2 sono segnali che vanno dal dispositivo verso l'altro dispositivo. CA1 e CB1, i primi due segnali, funzionano solamente da ingressi della PIA e vengono usati per inoltrare una richiesta di interruzione.

Quando il dispositivo esterno vuole fare una comunicazione con la PIA, emette un segnale su CA1 e questo viene tradotto come interruzione per il processore, dall'esterno arriva quindi il segnale CA1 e il dispositivo emette il segnale IRQA in modo tale da svegliare o richiamare l'attenzione del processore, il dispositivo inizia quindi a mandare i dati all'interfaccia parallela.

CA2 e CB2 sono linee di controllo verso le periferiche sia esse d'ingresso che d'uscita, le funzioni svolte dalle linee sono programmate mediante 3 bit del registro di controllo (CRA), i bit interessati sono CRA1,CRA2 e CRA3. Il registro CRA si presenta in questo modo:

	Bit 7	Bit 6	Bit 5,4,3	Bit 2	Bit 1,0
CRA	IRQA 1	IRQA 2	CA2	DDRA	CA1

I due registri di controllo sono perfettamente analoghi, il bit 6 e il bit 7 sono i segnali per gestire le interruzioni che il dispositivo emette. I bit 3,4 e 5 servono per controllare il bit CA2 e quindi i tipi di segnali che devono essere emessi verso l'ingresso e verso l'uscita, il bit 1,0 controllano il bit CA1 e il bit 2 permette di avere l'accesso al registro DDRA per indicare quale deve essere la funzione di comunicazione.

CRA0 e CRA1 determinano il modo in cui trattare un'interruzione:

- CRA0 determina se la richiesta di interruzione deve essere o meno inoltrata al processore attraverso la linea IRQA (è come una maschera);
- CRA1 determina quale fronte dell'interruzione è riconoscibile dall'ingresso CA1;

CRA2 serve per selezionare il registro dati o il registro di direzione, andando a segnare su CRA2 il valore "1" si indica la volontà di accedere al registro dati, viceversa se in esso indichiamo "0".

CRA2, CRA4 e CRA5 individuano uno dei possibili modi di funzionamento del bit CA2, che è l'unico segnale a disposizione per il controllo vero e proprio verso l'esterno:

- CRA5 stabilisce se la line CA2 funziona da ingresso di interruzione o da uscita:
- Se CRA5=0 allora CA2 è programmato per funzionare come linea di ingresso e funziona come CA1 per cui:
 - CRA4 determina il tipo di fronte di interruzione riconoscibile sull'ingresso CA2;
 - CRA3 viene utilizzato per mascherare le richieste di interrupt provenienti da CA2;
- Se CRA5=1 allora CA2 è programmato per funzionare come linea di uscita, in questo caso i bit:
 - CRA4 e CRA3 sono utilizzato per stabilire uno dei seguenti modi di funzionamento:
 - 100 metodo Handshake;
 - 101 metodo impulsivo;
 - 11x metodo dipendente da CRA3;

Ad esempio, se CRA5=1, CRA=1, all'uscita di CA2 si presenta un livello logico coincidente con quello presente nel bit CRA3.

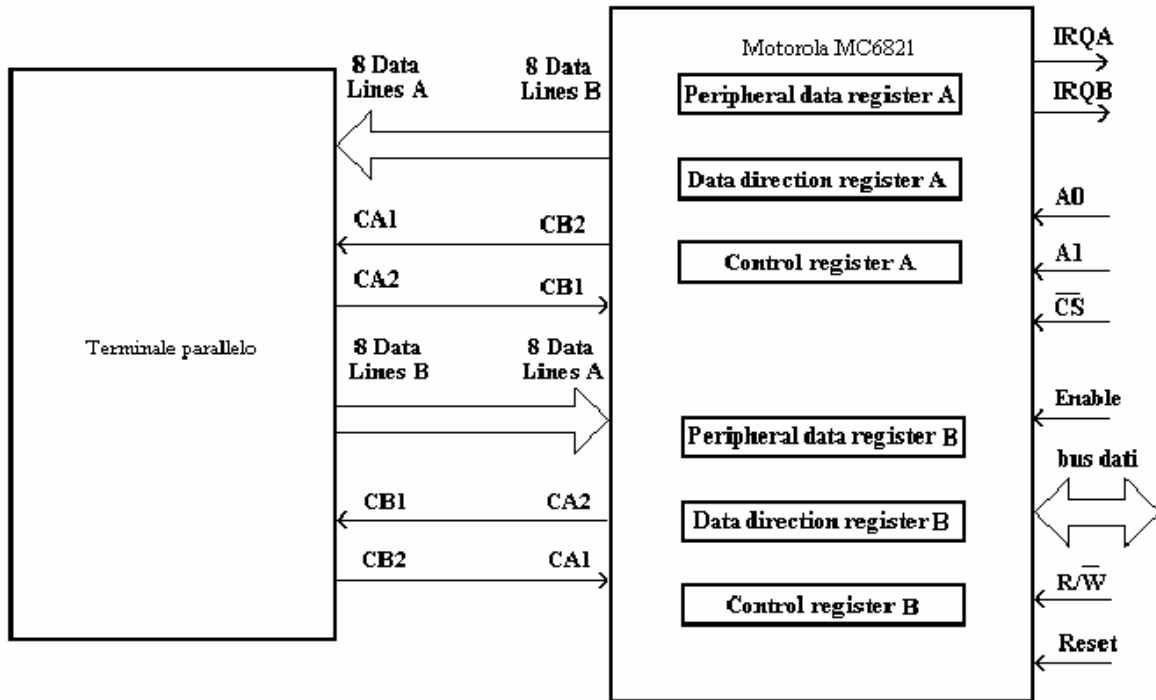
Ad esempio, consideriamo il caso in cui una periferica esterna funzionante da sorgente di dati sia connessa alla PIA. La periferica vuole inviare un dato alla PIA sul lato A (settato da ingresso), in tal caso:

- la periferica mette i dati sul Bus-dati;
- attraverso la linea CA1 invia alla PIA il segnale "Dato pronto per la lettura", sulla transizione di CA1 anche CA2 si alza;
- internamente alla PIA il flag IRQA 1 (bit 7CRA) si alza e viene inviata al processore una richiesta di interruzione che causerà lo svolgimento di un apposita interruzione.

La routine del processore prevede tra l'altro una operazione di lettura dati.

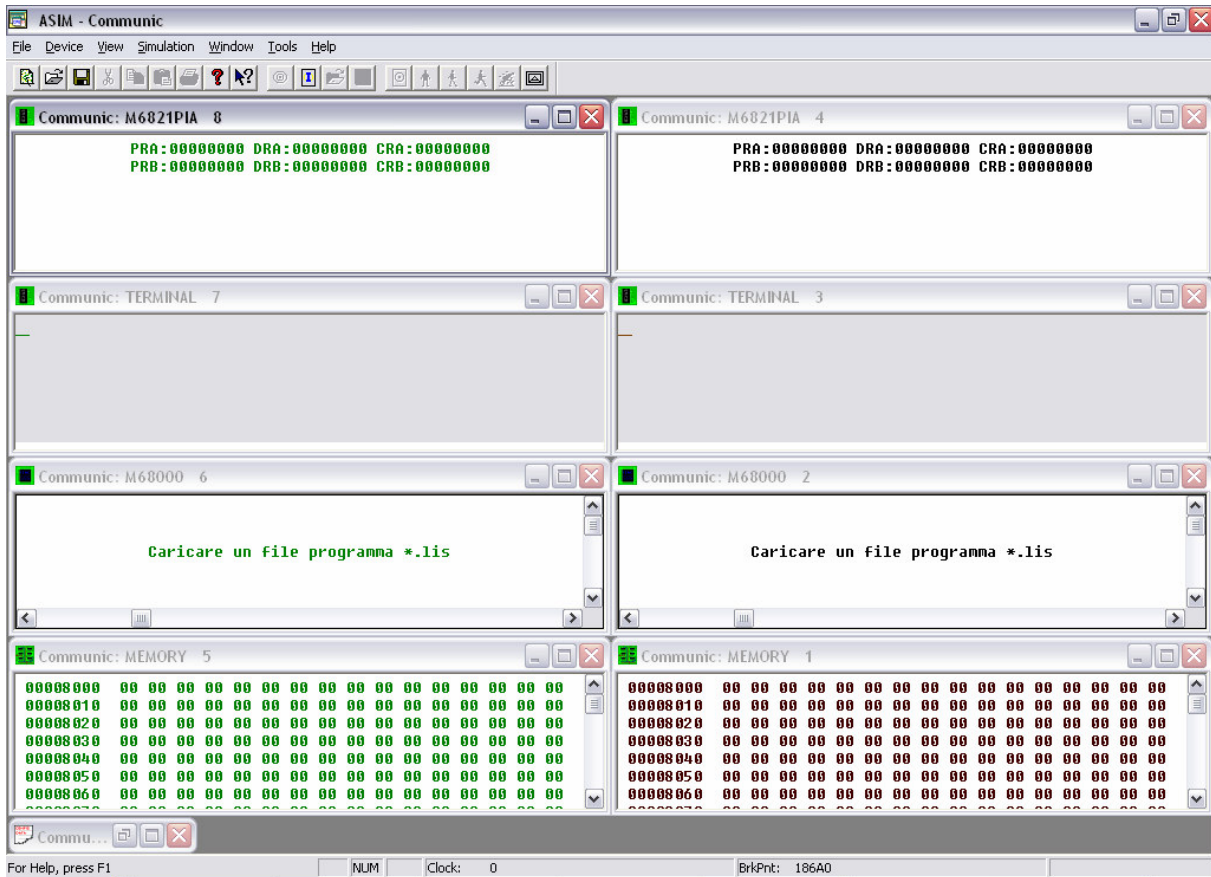
Ultimata la lettura, il flag di interruzione verrà abbassato e anche il segnale CA2 tornerà al valore logico basso, tale transizione costituisce una segnalazione "operazione di lettura terminata" per l'unità periferica esterna.

Nella simulazione con ASIM lo scambio dei dati tra il terminale e l'interfaccia parallela è regolato da un protocollo di handshaking sviluppato attraverso le linee di handshaking C!1, CA2, CB1 e CB2. Le proprietà che deve posseder l'interfaccia parallela da connettere al terminale sono quelle di possedere linee compatibili a quelle del terminale e rispettare il suo protocollo di handshaking. Lo schema di collegamento che si realizza è il seguente:



Address 1: 00002000. Address 2: 00002001.
 Com1: 0006. Com2: 0001. Com3: 0002. Com4: 0000.
CHIP Name: M6821PIA
 Type: Device. Identif: 08. BUS: 0005.
 Address 1: 00002004. Address 2: 00002007.
 Com1: 0006. Com2: 0003. Com3: 0004. Com4: 0204.

Al comando di Creat All la schermata di ASIM presenta i dispositivi in questo modo:



Lezione N°17: Alcune note sull'interfacciamento di microprocessori ed esempi di driver

In un programma che prevede operazioni di input/output, nel momento in cui ha inizio la comunicazione con il dispositivo (in alcuni casi è il dispositivo che avverte il processore oppure viceversa) il programma in corso entra in un ciclo di attesa nel quale verifica ripetutamente lo stato del dispositivo. Durante questo tale periodo il processore non esegue alcuna operazione utile. Il processore potrebbe allora sfruttare quel tempo perso per svolgere altre funzioni ad esso assegnate, ma perché ciò sia realizzabile è necessario favorire un meccanismo per far sì che il dispositivo avverta il processore quando è pronto. Questo segnale è detto interrupt.

La soluzione che comunemente è adottata consiste nel permettere al "supervisore" di prendere controllo del processore al termine di ciascun ciclo. Questo avviene esclusivamente nel caso in cui si verificano eventi eccezionali, di solito sincroni con l'esecuzione del programma correntemente in corso. In assenza di tali eventi l'elaborazione, come già citato nel corso di queste note, procede nella maniera consueta.

La fase di interrupt viene eseguita nel caso in cui il segnale INT è alto (è il dispositivo che avverte il processore). Questo evento è sintomatico del fatto che alcuni eventi sono irrisolti e necessitano di una soluzione al più presto. L'interruzione rappresenta il servizio che provvede a gestire questi eventi: ciascuna delle interruzioni richiede al sistema specifiche azioni elaborative (conteggio del tempo, segnalare un errore, rendere disponibile una risorsa...). Al termine del servizio il programma interrotto viene ripreso, se tale ripresa è compatibile con la stessa interruzione.

Durante questa fase del ciclo del processore, comunque, non viene eseguito un programma. Se INT è alto si innesca il sistema per saltare alla particolare ISR che verrà eseguita (come tutti i programmi) nel normale ciclo del processore. Per eseguire un programma (software), infatti, sarebbe necessario trovarsi all'interno del ciclo principale e muoversi tra le fasi di fetch ed execute. Ciò che avviene nella fase di interrupt consiste invece in una serie di meccanismi hardware che preparano il processore a gestire l'interruzione.

Una procedura di servizio dell'interrupt può essere vista come una normale procedura (sottoprogramma). Una differenza importante è che un sottoprogramma esegue una funzione richiesta dal programma chiamante, mentre in generale una procedura di servizio degli interrupt può non avere nulla in comune con il programma che è in esecuzione al momento della ricezione del segnale di interrupt. Una interruzione può essere causata da necessità di natura diversa, per esempio esistono:

- Interruzioni periodiche, ad esempio ogni 10ms, per permettere al sistema operativo di calcolare il tempo speso da un applicazione e aggiornare eventualmente il timer per quell'evento;
- Interruzioni di I/O, una periferica di I/O che informa su un suo particolare stato (ad esempio il dispositivo indica alla

- CPU che è pronto a ricevere dati) emette questo segnale per sincronizzarsi con il processore;
- Interruzioni per errori del programma correntemente eseguito, ad esempio un programma può indicare al processore che nei suoi calcoli si è verificato un overflow oppure un'istruzione è inesistente o privilegiata, questo tipo di interruzioni si dicono anche traps;
 - Interruzioni per guasti al sistema rilevati da apposite sonde quali ad esempio un termometro per la misura della temperatura del processore oppure un amperometro che controlla l'idoneità (entro certi limiti) di una corrente per alcuni dispositivi di una piastra madre.
 - Interruzioni per riportare l'intero sistema in uno stato noto di reset;
 - Interruzioni programmate, dette anche software interrupt, generate da un programma che voglia accedere una risorsa condivisa (ad esempio una periferica di I/O) e per questo motivo chiede la mediazione del S.O. Queste sono le uniche interruzioni ad essere sincrone con il programma in esecuzione.

Queste le fasi del processo delle interruzioni:

... ..

Esecuzione normale

Servizio dell'interruzione

Salvataggio del contesto (hardware)

Identificazione del device o della causa di interruzione

Salto all'entry point della Interrupt Service Routine, ISR

Salvataggio del contesto (software)

Servizio dell'interruzione

Ripristino del contesto (software)

Ripristino del contesto (hardware)

Esecuzione normale

... ..

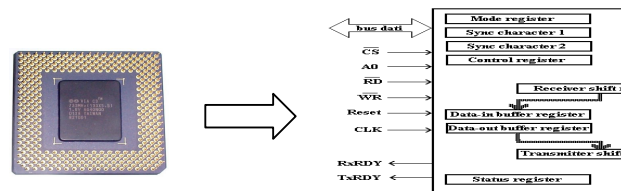
Una interruzione potrebbe eseguire un'elaborazione B completamente indipendente da quella A correntemente in corso sul processore, interrompendola.

La gestione delle interruzioni deve quindi anche provvedere a mettere A in condizioni di continuare successivamente senza accorgersi di nulla. Sorge la necessità di salvare (prima) e ripristinare (dopo) lo stato del programma che viene di volta in volta interrotto.

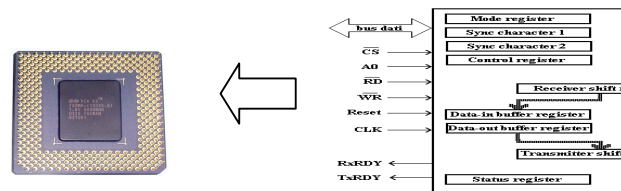
In questo modo A può continuare la sua elaborazione senza risentire in alcun modo del servizio dell'interruzione, a parte il ritardo di tempo. Le informazioni che devono essere salvate e ripristinate comprendono di solito i PC, i flag dei codici di condizione e il contenuto di qualsiasi registro che sia usato sia dal programma che dalla routine di gestione dell'interruzione. Un approccio alternativo consiste nel prevedere che sia il dispositivo stesso a fornire un proprio identificativo all'atto di una richiesta. L'identificativo serve inoltre a calcolare l'indirizzo della routine di interruzione che deve essere invocata, rendendo il meccanismo molto efficiente. Il numero di bits utilizzati pone il

limite massimo sul numero di diversi dispositivi che possono essere riconosciuti. Sono possibili diversi tipi di interfacciamento tra CPU e dispositivi (tra l'altro già citati nel corso di queste note):

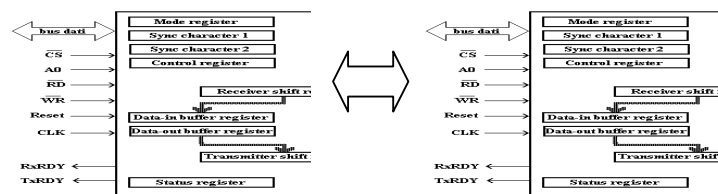
-Interfacciamento Processore/Dispositivo:



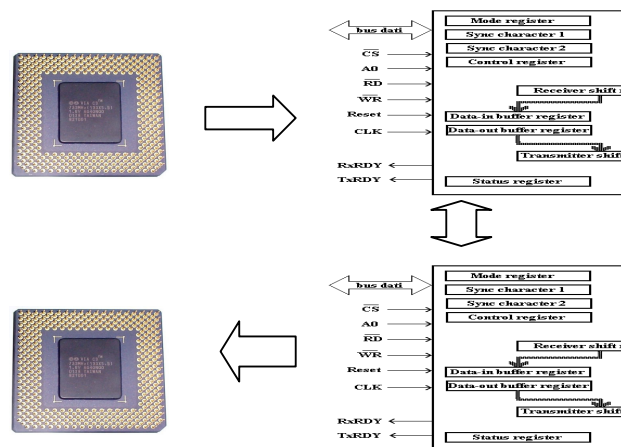
-Interfacciamento Dispositivo/Processore:



-Interfacciamento Dispositivo/Dispositivo:



-Interfacciamento tra sistemi:



Sono da considerare elementi fondamentali alla comunicazione:

Dal lato Periferica:

- Interfaccia della periferica
- Protocollo di Comunicazione

Dal lato Processore

- Meccanismo di Controllo
- Tipo di istruzioni di I/O

In questa sezione si cercherà di dare più attenzione alla creazione del driver. Il driver è il programma che gestisce il dialogo tra il processore e la periferica, esso implementa il protocollo di comunicazione e gestisce la comunicazione tra CPU e dispositivo.

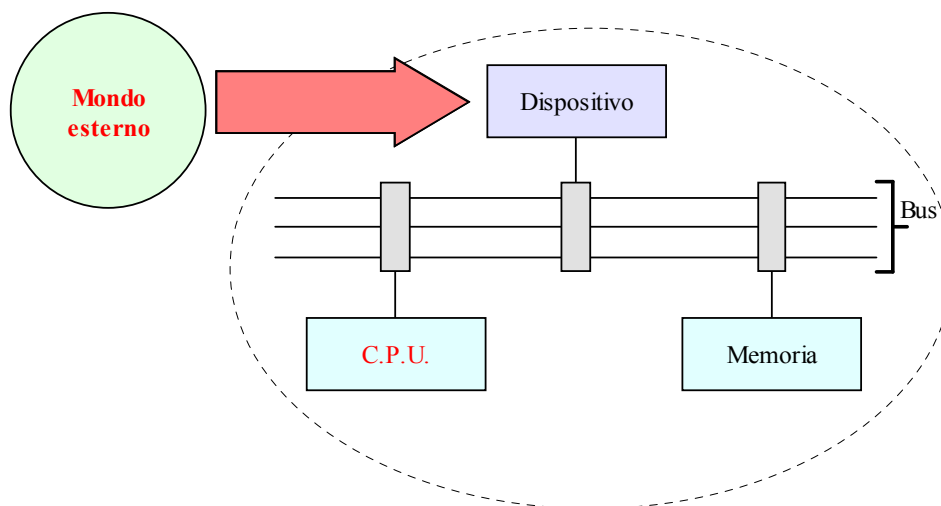
Esempio 1:

Il processore è collegato ad un dispositivo in grado di ricevere 8 byte dall'esterno e memorizzarli direttamente in memoria a partire dall'indirizzo \$9000. Il dispositivo dotato di un registro controllo e stato (RCS) all'indirizzo \$2000 ed un registro dato (RD) all'indirizzo \$2001 opera in questo modo:

- 1) Per essere attivato deve essere messo alto il bit 0 nel registro RCS;
- 2) Quando è attivato, appena riceve un segnale dall'esterno di richiesta di comunicazione emette un segnale di interruzione. Per iniziare la comunicazione è necessario che il processore metta ad alto il bit 1 del registro RCS;
- 3) Quando la trasmissione è completata il dispositivo emette una nuova interruzione;

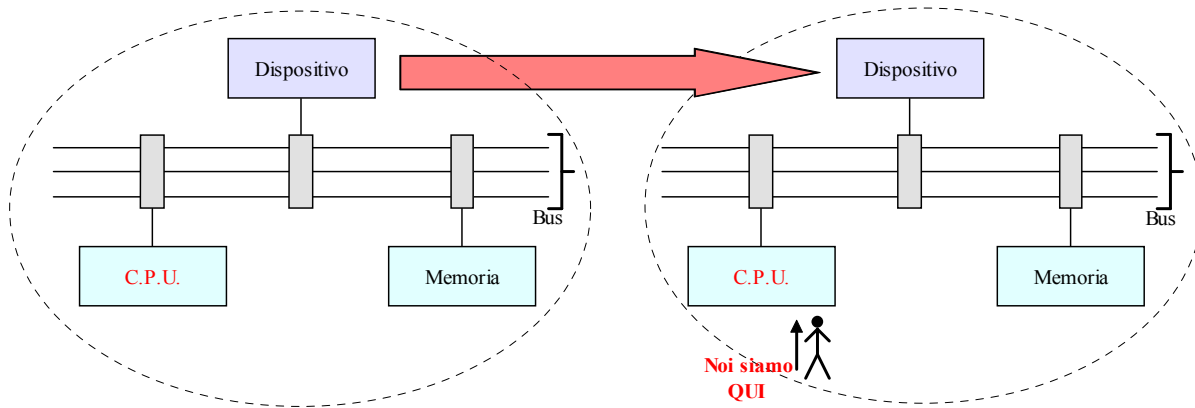
Scrivere un programma principale che attivi il dispositivo e le due ISR necessarie per avviare la comunicazione, leggere i dati e spostarli ad un differente indirizzo di memoria.

Volendo schematizzare la situazione si avrà questa configurazione:



Risulta quindi chiaro che è il dispositivo a dialogare con l'esterno ed il processore scambia informazioni con il solo dispositivo e la memoria tramite il bus.

Con il blocco "mondo esterno" si vuole tuttavia indicare tutti quei dispositivi, macchine o eventi esterni (quale ad esempio la pressione di un tasto), in grado di produrre sul dispositivo una modifica significativa del suo stato. Sono infatti possibili anche situazioni di questo tipo:

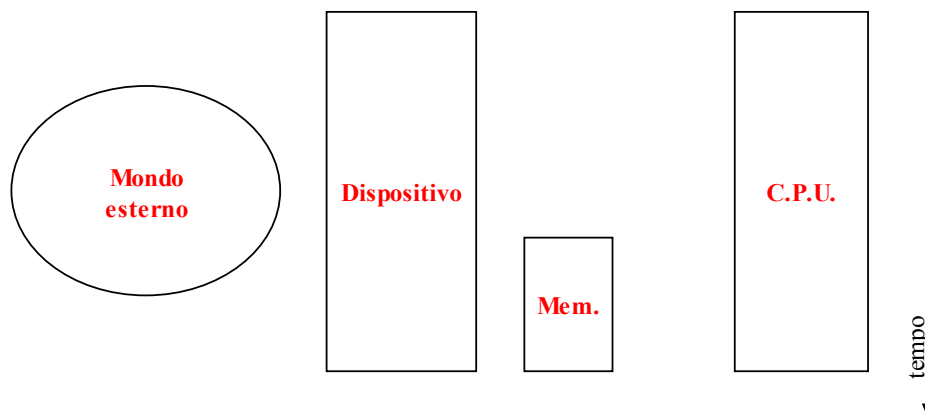


Il mondo esterno per un dispositivo può essere cioè anche un'altra macchina che intende interagire con quella già esistente.

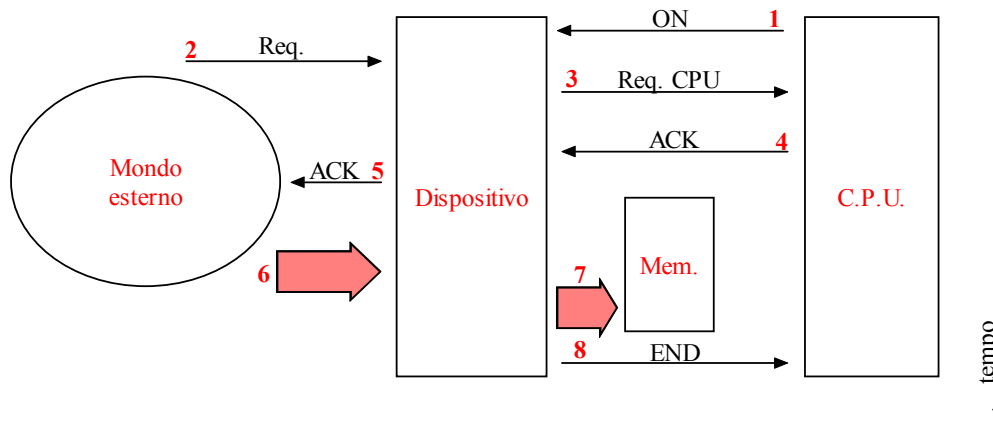
Nello sviluppo dei driver è bene precisare inoltre la posizione in cui il progettista deve andarsi a collocare per poter quindi comprendere le operazioni che gli competono (e tralasciare quindi tutte quelle appartenenti al dispositivo).

Un ulteriore grafico, che a breve mostreremo, aiuta invece a comprendere l'evoluzione dei segnali che nel tempo si possono presentare al dispositivo ed alla C.P.U.

Si tratta di un grafico in cui si è solito riportare due grandi blocchi, uno che rappresenta il dispositivo e l'altro la C.P.U. Più a sinistra collochiamo invece un ulteriore blocco che è significativo del "mondo esterno". In alcuni casi è inoltre necessario disegnare un altro blocco e collocarlo al centro, tra il blocco dispositivo e quello C.P.U. e che simboleggia la memoria, ad esempio:



I segnali utili al protocollo di comunicazione saranno disposti su questo grafico in ordine cronologico, tenendo conto cioè del fattore tempo (notare a destra del disegno la presenza di una retta temporale). Volendo realizzare un diagramma di questo tipo per l'esempio preso in considerazione si avrà che:



Questo diagramma riassume in modo veloce e grafico il protocollo usato dal dispositivo per comunicare con il processore, il diagramma va letto dall'alto verso il basso. Le uniche due frecce dirette dal dispositivo verso la CPU costituiscono di fatto le due interruzioni che il problema chiede di risolvere. Dal diagramma notiamo inoltre che il dispositivo, dopo aver avvisato la C.P.U. di una comunicazione dall'esterno (prima interruzione) procede nel suo protocollo fino alla ricezione degli 8 byte previsti dalle specifiche del problema. La ricezione di queste informazioni o dati avviene senza interrompere il processore per cui i dati vengono depositati dal dispositivo direttamente in memoria, questa è una caratteristica di alcuni dispositivi capaci di accedere in maniera autonoma in memoria e che appartengono quindi alla classe di dispositivi propriamente detti **DMA** (direct access memory). Un driver che implementa il protocollo richiesto dal problema è il seguente:

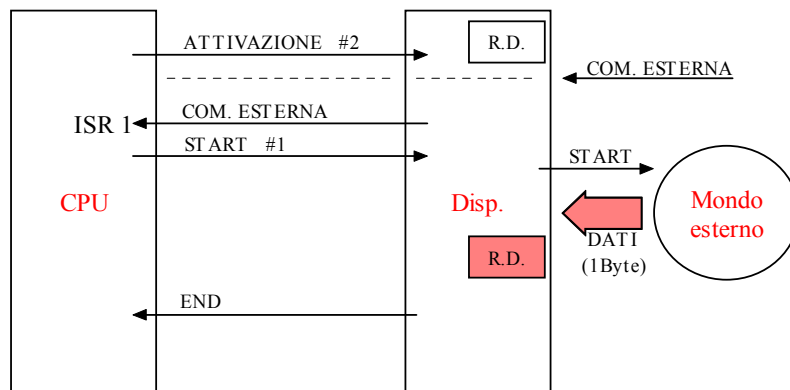
	ORG	\$8000		*
MEM1	EQU	\$9000		* MEM1 è l'area di memoria per i dati
MEM2	EQU	\$9500		* MEM2 è l'area di memoria per i dati
RCS	EQU	\$2000		* Il registro di controllo e stato è mappato all'indirizzo 2000
RD	EQU	\$2001		* Il registro dati è mappato all'indirizzo 2001
				*
BEGIN	LEA	RCS,A0		* Carico l'indirizzo del registro RCS in A0
	BSET	#0,(A0)		* Impostazione del bit num. 0 ad un valore alto
AZIONE	BRA	AZIONE		* Il programma continua a fare le sue istruzioni
				*
INT1	ORG	\$8500		* Prima interruzione, comunicazione dall'esterno
	MOVE.L	A0,-(A7)		* Salvo i registri usati
	LEA	RCS,A0		* Carico l'indirizzo del registro RCS in A0
	BSET	#1,(A0)		* Impostazione del bit num. 1 ad un valore alto
	MOVE.L	(A7)+,A0		* Ripristino i registri alla situazione pre-interruzione
	RTE			* Comando di ritorno dall'eccezione
				*
INT2	ORG	\$8600		* Seconda interruzione, i dati vengono spediti
	MOVEM.L	A0-A7/D0-D7,-(A7)		* Salvataggio dei registri usati
	LEA	RD,A0		* Carico l'indirizzo del registro dati in A0
	LEA	MEM1,A1		* Carico l'indirizzo dell'area di memoria-1 in A1
	LEA	MEM2,A2		* Carico l'indirizzo dell'area di memoria-2 in A2
	MOVE	#7,D0		* Definisco in D0 un variabile di conteggio
LOOP	MOVE	(A1)+,D1		* Etichetta di riferimento per il ciclo a conteggio
	MOVE	D1,(A2)+		* Sposto gli 8 byte da una locazione di memoria all'altra
	DBNE	D0,LOOP		* Decremento D0 e salto a LOOP se non uguale a zero
	MOVEM.L	(A7)+,A0-A7/D0-D7		* Ripristino dei registri alla situazione pre-interruzione
	RTE			* Comando di ritorno dall'eccezione
	END	BEGIN		* Fine del programma principale

Esempio 2:

Il processore è collegato ad un dispositivo in grado di ricevere 1 byte dall'esterno. Il dispositivo dotato di un registro di controllo e stato (RCS, indirizzo \$2000) ed un registro dato (RD, indirizzo \$2001), opera in questo modo:

- 1) Per essere attivato deve essere messo alto il bit 2 del registro RCS;
- 2) Quando è attivo appena riceve un segnale dall'esterno di richiesta di comunicazione emette un segnale di interruzione. Per iniziare la comunicazione è necessario che il processore metta ad alto il bit 1 del registro RCS;
- 3) Quando la trasmissione è completa il dispositivo ha il dato disponibile nel registro RD ed emette una nuova interruzione;

Scrivere un programma principale che attivi il dispositivo e le due ISR necessarie per avviare la comunicazione, leggere i dati e spostarli in memoria.



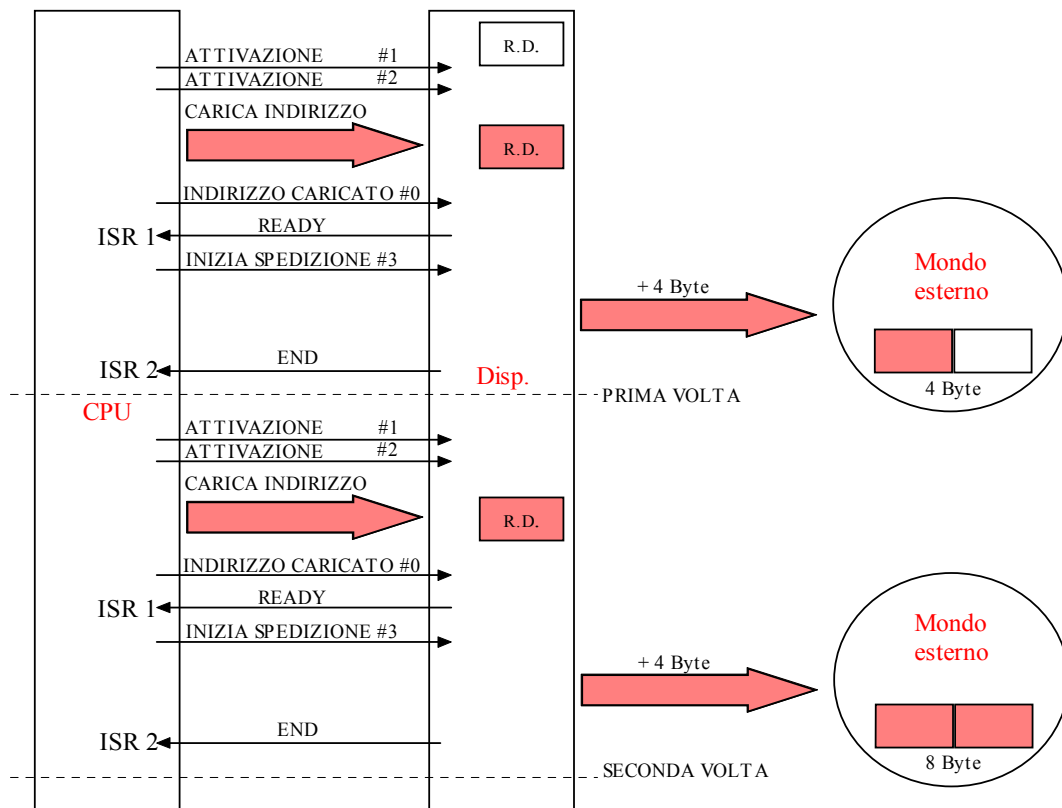
	ORG	\$8000	*
RCS	EQU	\$2000	*Il registro RCS è mappato in memoria all'indirizzo 2000
RD	EQU	\$2001	*Il registro RD è mappato in memoria all'indirizzo 2001
MEM	DS.B		*Riservo 1 byte della memoria per ospitare il dato
MAIN	MOVE	#RCS,A0	*Carico in A0 il registro RCS
	MOVE	#RD,A1	*Carico in A1 il registro RD
	LEA	MEM,A2	*Carico in A2 l'indirizzo dell'area di memoria prima riservata
	BSET	#2,(A0)	*Attivo il dispositivo
...			
CPU	BRA	CPU	*Il processore svolge altri programmi
...			
ISR1	ORG	#8500	*Prima interruzione
	MOVEM	A0,-(A7)	*Salvataggio dei registri che la ISR andrà a sporcare
	MOVE	#RCS,A0	*Carico in A0 il registro RCS
	BSET	#1,(A0)	*Impostazione del bit N°1 del registro RCS, segnale di START
	MOVEM	(A7)+,A0	*Ripristino lo stato precedente alla prima interruzione
	RTE		*Ritorna dall'eccezione
ISR2	ORG	#8600	*Seconda interruzione
	MOVEM	A0-A1/D0,-(A7)	*Salvataggio dei registri che la ISR andrà a sporcare
	MOVE	#RD,A0	*Carico in A0 il registro RD
	LEA	MEM,A1	*Carico in A1 l'indirizzo dell'area di memoria riservata al dato
	MOVE.B	(A0),D0	*Leggo il dato nel registro D0 del processore
	MOVE.B	D0,(A1)	*Sposto il dato nell'area di memoria ad esso riservato
	MOVEM	(A7)+,A0-A1/D0	*Ripristino lo stato precedente alla seconda interruzione
	RTE		*Ritorna dall'eccezione

Esempio 3:

Il processore è collegato ad un dispositivo in grado di spedire 4 byte all'esterno direttamente dalla memoria. Il dispositivo, dotato di un registro di controllo e stato (RCS, indirizzo \$2000) ed un registro dato (RD all'indirizzo \$2002), opera in questo modo:

- 1) Per essere attivato devono essere alzati i bit 1,2 del registro RCS;
- 2) Per cominciare la spedizione è necessario prima porre in RD l'indirizzo di memoria a partire dal quale si trovano i 4 byte da spedire, poi alzare il bit 0 del registro RCS;
- 3) Quando il dispositivo è pronto a spedire emette un segnale di interruzione al processore, perché la trasmissione abbia luogo è quindi necessario alzare il bit 3 del registro RCS;
- 4) Quando la trasmissione è completata il dispositivo emette una nuova interruzione;

Scrivere un programma principale che attivi il dispositivo e le due ISR necessarie per avviare la comunicazione, e spedire i dati due volte di seguito (spedendo così un totale di 8 byte).



```
RCS      EQU    $2000
RD        EQU    $2002
MEM       DS.B   8
CHK       DS.B   1
```

```
MAIN      MOVE #RCS,A0
           MOVE #RD,A1
           LEA MEM,A2
           LEA CHK,A3
           MOVE #0,(A3)
           BSET #1,(A0)
           BSET #2,(A0)
           MOVE A2,(A1)
```

*
*Il registro RCS è mappato in memoria all'indirizzo 2000
*Il registro RD è mappato in memoria all'indirizzo 2001
*Area di memoria dei dati
*Variabile di controllo

*Carico in A0 il registro RCS
*Carico in A1 il registro RD
*Carico in A2 l'indirizzo di memoria dei 4 byte
*Carico in A3 l'indirizzo della variabile di controllo
*Inizializzo a 0 la variabile di controllo
*Attivazione del dispositivo
*Attivazione del dispositivo
*Carico nel registro RD l'indirizzo dei 4 byte

	BSET #0,(A0)	*Alzo il bit 0 del registro RCS, pronto a spedire
	...	
CPU	BRA CPU	*Il processore svolge altri programmi
	...	
	ORG \$8500	*Prima interruzione
ISR1	MOVEM A0,-(A7)	*Salvataggio dei registri che la ISR andrà a sporcare
	MOVE #RD,A0	*Carico in A0 il registro RD
	BSET #3,(A0)	*Imposto ad alto il bit 3 del registro RD
	MOVEM (A7)+,A0	*Ripristino lo stato precedente alla interruzione
	RTE	*Ritorna dall'eccezione
	ORG \$8700	*Seconda interruzione
ISR2	MOVEM A0-A3,-(A7)	*Salvataggio dei registri che la ISR andrà a sporcare
	MOVE #RCS,A0	*Carico in A0 il registro RCS
	MOVE #RD,A1	*Carico in A1 il registro RD
	LEA MEM,A2	*Carico in A2 l'indirizzo di memoria dei dati
	LEA CHK,A3	*Carico in A3 l'indirizzo della variabile di controllo
	CMP #0,(A3)	*Confronto con 0 la variabile di controllo
	BNE FINE	*Se è uguale a 0 devo spedire altri 4 byte
	MOVE #1,(A3)	*Incremento di 1 la variabile di controllo
	MOVE A2,(A1)	*Carico nel registro RD un altro indirizzo
	BSET #0,(A0)	*Alzo il bit 0 del registro RCS, pronto a spedire
FINE	MOVEM (A7)+,A0-A3	*Ripristino lo stato precedente alla interruzione
	RTE	*Ritorna dall'eccezione

Alcuni Esercizi (Macchine sequenziali)

1. Progettare un riconoscitore di sequenza per la sequenza binaria 10-01 in un flusso continuo di bit;

Tab. di transizione:

STATI	INGRESSI	
	0	1
q ₀	q ₀	q ₁
q ₁	q ₂	q ₁
q ₂	q ₃	q ₄
q ₃	q ₅	q ₁
q ₄	q ₆	q ₁
q ₅	q ₀	q ₁ /1
q ₆	q ₀	q ₄ /1

Codifica degli stati:

q₀=000

q₁=001

q₂=010

q₃=011

q₄=100

q₅=101

q₆=110

Codifica della tabella (FF T):

STATI	A ₀	A ₁	A ₂	I	A ₀ '	A ₁ '	A ₂ '	U
q ₀	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	1	0
q ₁	0	0	1	0	0	1	1	0
	0	0	1	1	0	0	0	0
q ₂	0	1	0	0	0	0	1	0
	0	1	0	1	1	1	0	0
q ₃	0	1	1	0	1	1	0	0
	0	1	1	1	0	1	0	0
q ₄	1	0	0	0	0	1	0	0
	1	0	0	1	1	0	1	0
q ₅	1	0	1	0	1	0	1	0
	1	0	1	1	1	0	0	1
q ₆	1	1	0	0	1	1	0	0
	1	1	0	1	0	1	0	1
	1	1	1	0	-	-	-	0
	1	1	1	1	-	-	-	0

Mappe di Karnaugh:

Mappa di K. per A₀'

A ₀ A ₁ \ A ₂ I	00	01	11	10
00			1	
01		1		1
11			-	1
10		1	-	1

Mappa di K. per A₁'

A ₀ A ₁ \ A ₂ I	00	01	11	10
00			1	1
01		1	1	
11		1	-	
10	1	1	-	

A_0A_1		Mappa di K. per A_2'			
A_2I		00	01	11	10
0	0		1		
0	1	1			1
1	1			-	
1	0	1		-	1

A_0A_1		Mappa di K. per U			
A_2I		00	01	11	10
0	0				
0	1			1	
1	1				1
1	0				

Funzioni logiche:

$$A_0' = A_0A_1\bar{A}_2I + \bar{A}_0A_1\bar{A}_2I + A_0\bar{A}_1I + A_1A_2\bar{I} + A_0A_1$$

$$A_1' = A_0A_1 + A_0\bar{A}_2\bar{I} + A_1I + A_1A_2 + \bar{A}_0A_2\bar{I}$$

$$A_2' = \bar{A}_0A_1\bar{A}_2\bar{I} + \bar{A}_1\bar{A}_2I + \bar{A}_1A_2\bar{I}$$

$$U = A_0A_1\bar{A}_2I + A_0\bar{A}_1A_2I$$

2. Progettare un riconoscitore di sequenza per la sequenza binaria 01-11 in un flusso continuo di bit;

Tab. di transizione:

STATI	INGRESSI	
	0	1
q ₀	q ₁	q ₀
q ₁	q ₁	q ₂
q ₂	q ₃	q ₄
q ₃	q ₁	q ₅
q ₄	q ₁	q ₆
q ₅	q ₃	q ₀ /1
q ₆	q ₁	q ₀ /1

Codifica degli stati:

q₀=000

q₁=001

q₂=010

q₃=011

q₄=100

q₅=101

q₆=110

Codifica della tabella (FF T):

STATI	A ₀	A ₁	A ₂	I	A ₀ '	A ₁ '	A ₂ '	U
q ₀	0	0	0	0	0	0	1	0
	0	0	0	1	0	0	0	0
q ₁	0	0	1	0	0	0	0	0
	0	0	1	1	0	1	1	0
q ₂	0	1	0	0	0	0	1	0
	0	1	0	1	1	1	0	0
q ₃	0	1	1	0	0	1	0	0
	0	1	1	1	1	1	0	0
q ₄	1	0	0	0	1	0	1	0
	1	0	0	1	0	1	0	0
q ₅	1	0	1	0	1	1	0	0
	1	0	1	1	1	0	1	1
q ₆	1	1	0	0	1	1	1	0
	1	1	0	1	1	1	0	1
	1	1	1	0	-	-	-	0
	1	1	1	1	-	-	-	0

Mappe di Karnaugh:

Mappa di K. per A₀'

A ₀ A ₁ \ A ₂ I	00	01	11	10
00			1	1
01		1	1	
11		1	-	1
10			-	1

Mappa di K. per A₁'

A ₀ A ₁ \ A ₂ I	00	01	11	10
00			1	
01		1	1	1
11	1	1	-	
10		1	-	1

A ₀ A ₁		Mappa di K. per A ₂ '			
A ₂ I		00	01	11	10
0	0	1	1	1	1
0	1				
1	1	1		-	1
1	0			-	

A ₀ A ₁		Mappa di K. per U			
A ₂ I		00	01	11	10
0	0				
0	1			1	
1	1				1
1	0				

Funzioni logiche:

$$A_0' = A_0 A_1 + A_1 I + A_0 A_2 + A_0 \bar{I}$$

$$A_1' = A_0 A_1 + A_1 I + A_1 A_2 + A_0 \bar{A}_2 I + A_0 A_2 \bar{I} + \bar{A}_0 A_2 I$$

$$A_2' = \bar{A}_2 \bar{I} + \bar{A}_1 A_2 I$$

$$U = A_0 \bar{A}_1 A_2 I + A_0 A_1 \bar{A}_2 I$$

3. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna una rete in grado di riconoscere su di un ingresso di due bit X e Y la sequenza 00/01/10/11;

Tab. di transizione:

STATI	INGRESSI			
	00	01	11	10
q ₀	q ₁	q ₀	q ₀	q ₀
q ₁	q ₁	q ₂	q ₀	q ₀
q ₂	q ₁	q ₀	q ₃	q ₀
q ₃	q ₁	q ₀	q ₀	q ₀ /1

Codifica degli stati:

q₀=00

q₁=01

q₂=10

q₃=11

Codifica della tabella (FF T):

STATI	A ₀	A ₁	I ₁	I ₂	A ₀ '	A ₁ '	U
q ₀	0	0	0	0	0	1	0
	0	0	0	1	0	0	0
	0	0	1	0	0	0	0
	0	0	1	1	0	0	0
q ₁	0	1	0	0	0	0	0
	0	1	0	1	1	1	0
	0	1	1	0	0	1	0
	0	1	1	1	0	1	0
q ₂	1	0	0	0	1	1	0
	1	0	0	1	1	0	0
	1	0	1	0	0	1	0
	1	0	1	1	1	0	0
q ₃	1	1	0	0	1	0	0
	1	1	0	1	1	1	0
	1	1	1	0	1	1	0
	1	1	1	1	1	1	1

Mappe di Karnaugh:

A ₀ A ₁		Mappa di K. per A ₀ '			
A ₂ I		00	01	11	10
0	0			1	1
0	1		1	1	1
1	1			1	1
1	0			1	

A ₀ A ₁		Mappa di K. per A ₁ '			
A ₂ I		00	01	11	10
0	0	1			1
0	1		1	1	
1	1		1	1	
1	0		1	1	1

Funzioni logiche

$$A_0' = A_0 A_1 + A_0 \bar{I}_1 + A_0 I_2 + A_1 \bar{I}_1 I_2$$

$$A_1' = A_1 I_2 + A_1 I_1 + A_0 I_1 \bar{I}_2 + \bar{A}_1 \bar{I}_1 \bar{I}_2$$

$$U = A_0 A_1 I_1 I_2$$

4. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna una rete in grado di riconoscere su di un ingresso di due bit X e Y la sequenza 11/00/01/10;

Tab. di transizione:

STATI	INGRESSI			
	00	01	11	10
q ₀	q ₀	q ₀	q ₀	q ₁
q ₁	q ₂	q ₀	q ₀	q ₁
q ₂	q ₀	q ₃	q ₀	q ₁
q ₃	q ₀	q ₀	q ₀ /1	q ₁

Codifica degli stati:

q₀=00

q₁=01

q₂=10

q₃=11

Codifica della tabella (FF D):

STATI	A ₀	A ₁	I ₀	I ₁	A ₀ '	A ₁ '	U
q ₀	0	0	0	0	0	0	0
	0	0	0	1	0	0	0
	0	0	1	0	0	0	0
	0	0	1	1	0	1	0
q ₁	0	1	0	0	1	0	0
	0	1	0	1	0	0	0
	0	1	1	0	0	0	0
	0	1	1	1	0	1	0
q ₂	1	0	0	0	0	0	0
	1	0	0	1	1	1	0
	1	0	1	0	0	0	0
	1	0	1	1	0	1	0
q ₃	1	1	0	0	0	0	0
	1	1	0	1	0	0	0
	1	1	1	0	0	0	1
	1	1	1	1	0	1	0

Mappe di Karnaugh:

A ₀ A ₁		Mappa di K. per A ₀ '			
I ₀ I ₁		00	01	11	10
0	0		1		
0	1				1
1	0				
1	1				

A ₀ A ₁		Mappa di K. per A ₁ '			
I ₀ I ₁		00	01	11	10
0	0				
0	1				1
1	0	1	1	1	1
1	1				

Funzioni logiche:

$$A_0' = \bar{A}_0 A_1 \bar{I}_0 \bar{I}_1 + A_0 \bar{A}_1 \bar{I}_0 I_1$$

$$A_1' = I_0 I_1 + A_0 \bar{A}_1 I_1$$

$$U = A_0 A_1 I_0 \bar{I}_1$$

5. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna una macchina che ha in ingresso i caratteri X,Y e = ed emette in uscita il valore 1 quando viene riconosciuta la sequenza X=Y, la codifica dei caratteri è libera;

Tab. di transizione:

STATI	INGRESSI			
	00	01	11	10
q ₀	q ₁	q ₀	q ₀	q ₀
q ₁	q ₁	q ₀	q ₂	q ₀
q ₂	q ₁	q ₀ /1	q ₀	q ₀

Codifica degli stati:

q₀=00

q₁=01

q₂=10

- =11

Codifica della tabella (FF D):

STATI	A ₀	A ₁	I ₁	I ₂	A ₀ '	A ₁ '	U
q ₀	0	0	0	0	0	1	0
	0	0	0	1	0	0	0
	0	0	1	0	0	0	0
	0	0	1	1	0	0	0
q ₁	0	1	0	0	0	1	0
	0	1	0	1	0	0	0
	0	1	1	0	1	0	0
	0	1	1	1	0	0	0
q ₂	1	0	0	0	0	1	0
	1	0	0	1	0	0	1
	1	0	1	0	0	0	0
	1	0	1	1	0	0	0

Mappe di Karnough:

A ₀ A ₁ I ₀ I ₁		Mappa di K. per A ₀ '			
		00	01	11	10
0	0				
0	1				
1	0				
1	1		1		

A ₀ A ₁ I ₀ I ₁		Mappa di K. per A ₁ '			
		00	01	11	10
0	0	1	1		1
0	1				
1	0				
1	1				

Funzioni logiche:

$$A_0' = \bar{A}_0 A_1 I_0 \bar{I}_1$$

$$A_1' = \bar{A}_0 \bar{I}_0 \bar{I}_1 + \bar{A}_1 \bar{I}_0 \bar{I}_1$$

$$U = A_0 \bar{A}_1 \bar{I}_0 I_1$$

6. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna una rete in grado di riconoscere su di un ingresso di due bit X e Y la sequenza 11/01/11/11;

Tab. di transizione:

STATI	INGRESSI			
	00	01	11	10
q ₀	q ₀	q ₀	q ₀	q ₁
q ₁	q ₀	q ₂	q ₀	q ₁
q ₂	q ₀	q ₀	q ₀	q ₃
q ₃	q ₀	q ₂	q ₀	q ₁ /1

Codifica degli stati:

q₀=00

q₁=01

q₂=10

q₃=11

Codifica della tabella (FF D):

STATI	A ₀	A ₁	I ₀	I ₁	A ₀ '	A ₁ '	U
q ₀	0	0	0	0	0	0	0
	0	0	0	1	0	0	0
	0	0	1	0	0	0	0
	0	0	1	1	0	1	0
q ₁	0	1	0	0	0	0	0
	0	1	0	1	1	0	0
	0	1	1	0	0	0	0
	0	1	1	1	0	1	0
q ₂	1	0	0	0	0	0	0
	1	0	0	1	0	0	0
	1	0	1	0	0	0	0
	1	0	1	1	1	1	0
q ₃	1	1	0	0	0	0	0
	1	1	0	1	1	0	0
	1	1	1	0	0	0	0
	1	1	1	1	0	1	1

Mappe di Karnaugh:

A ₀ A ₁		Mappa di K. per A ₀ '			
I ₀ I ₁		00	01	11	10
0	0				
0	1		1	1	
1	0				1
1	1				

A ₀ A ₁		Mappa di K. per A ₁ '			
I ₀ I ₁		00	01	11	10
0	0				
0	1				
1	0	1	1	1	1
1	1				

Funzioni logiche:

$$A_0' = A_1 \bar{I}_0 I_1 + A_0 \bar{A}_1 I_0 I_1$$

$$A_2' = I_0 I_1$$

$$U = A_0 A_1 I_0 I_1$$

7. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna un contatore modulo 8 bidirezionale che effettui il conteggio a crescere per il bit $Up=1$ e quello a decrescere per il bit $Up=0$;

Grafo:

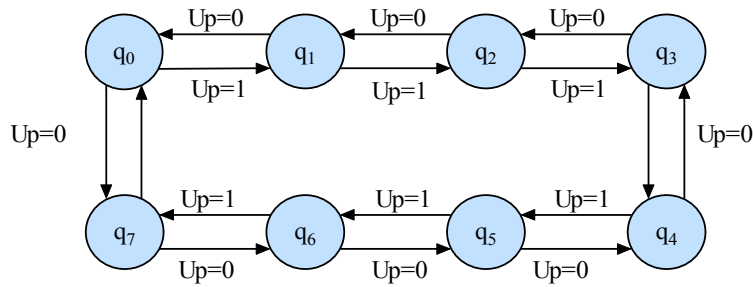


Tabella di transizione:

Stato	$Up=0$	$Up=1$
q_0	q_7	q_1
q_1	q_0	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_6
q_6	q_5	q_7
q_7	q_6	q_0

Codifica della tabella (FF T):

$A_0A_1A_2$	$A_0'A_1'A_2'$	DIV
0 0 0	1 1 1	1
0 0 1	0 0 1	0
0 1 0	0 1 1	0
0 1 1	0 0 1	0
1 0 0	1 1 1	0
1 0 1	0 0 1	0
1 1 0	0 1 1	0
1 1 1	0 0 1	0

Mappe di Karnaugh:

A_0A_1	Mappa di K. per A_0'			
A_2Up	00	01	11	10
0 0	1			1
0 1				
1 1		1	1	
1 0				

A_0A_1	Mappa di K. per A_1'			
A_2Up	00	01	11	10
0 0	1	1	1	1
0 1				
1 1	1	1	1	1
1 0				

Funzioni logiche:

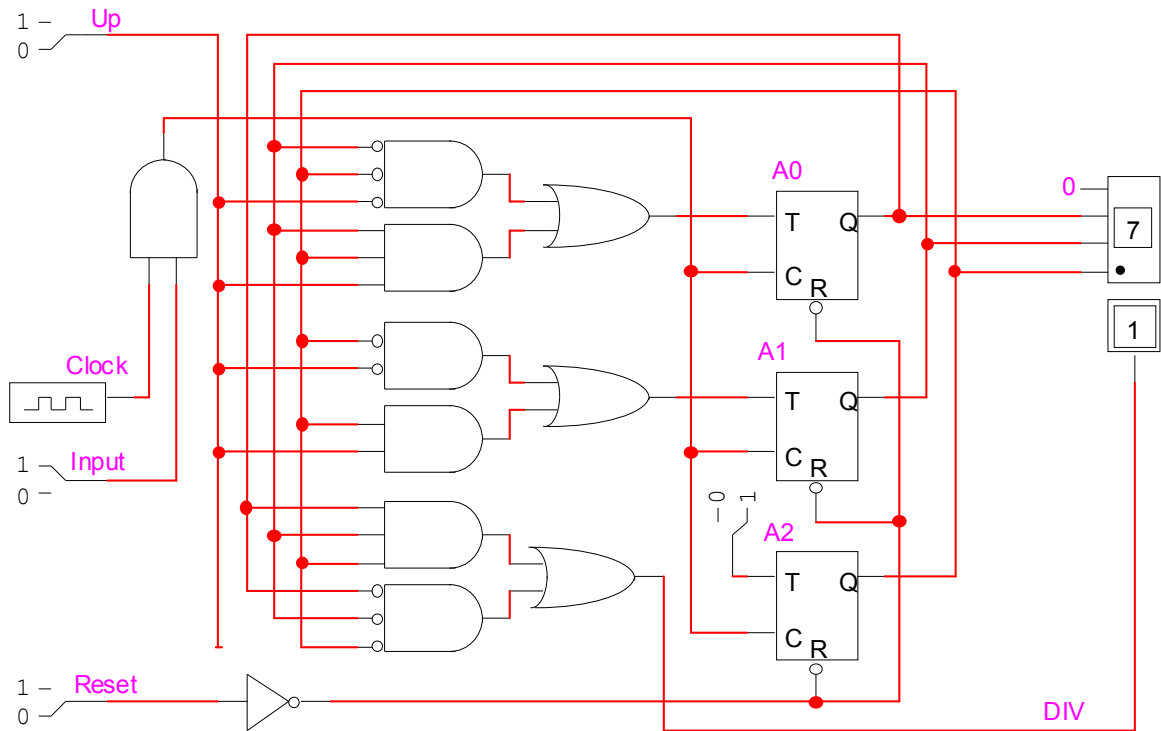
$$A_0' = \overline{A_1} \overline{A_2} \overline{Up} + A_1 A_2 Up$$

$$A_1' = \overline{A_2} \overline{Up} + A_2 Up$$

$$A_2' = 1$$

$$DIV = \overline{A_0} \overline{A_1} \overline{A_2} + A_0 A_1 A_2$$

Circuito:



Nota: A_2 è sempre alto per qualsiasi valore, come si può osservare anche dalla tabella.

8. Progettare secondo la teoria delle macchine sequenziali a sincronizzazione esterna un contatore modulo 8 ad incremento variabile che effettui il conteggio con un incremento +1 per $Up=0$ e +2 per $Up=1$;

Grafo:

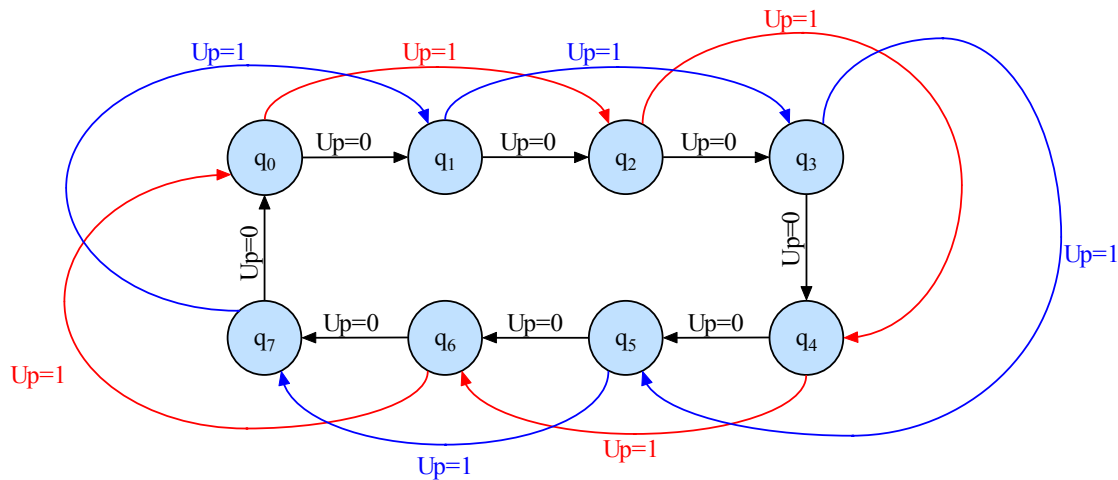


Tabella di transizione:

Stato	Up=0	Up=1
q ₀	q ₁	q ₂
q ₁	q ₂	q ₄
q ₂	q ₃	q ₆
q ₃	q ₄	q ₀
q ₄	q ₅	q ₇
q ₅	q ₆	q ₀
q ₆	q ₇	q ₁
q ₇	q ₀	q ₃

Codifica della tabella (FF T):

Per Up=0 (Incr. = +1)

A ₀ A ₁ A ₂	A ₀ 'A ₁ 'A ₂ '	DIV
0 0 0	0 0 1	0
0 0 1	0 1 1	0
0 1 0	0 0 1	0
0 1 1	1 1 1	0
1 0 0	0 0 1	0
1 0 1	0 1 1	0
1 1 0	0 0 1	0
1 1 1	1 1 1	1

Per Up=1 (Incr. = +2)

A ₀ A ₁ A ₂	A ₀ 'A ₁ 'A ₂ '	DIV
0 0 0	0 1 0	0
0 0 1	0 1 0	0
0 1 0	1 1 0	0
0 1 1	1 1 0	0
1 0 0	0 1 0	0
1 0 1	0 1 0	0
1 1 0	1 1 0	0
1 1 1	1 1 0	1

Mappe di Karnaugh:

A_0A_1		Mappa di K. per A_0'			
A_2Up		00	01	11	10
	0				
	0				
	1		1	1	
	1		1	1	
	0		1	1	

A_0A_1		Mappa di K. per A_1'			
A_2Up		00	01	11	10
	0				
	0				
	1	1	1	1	1
	1	1	1	1	1
	0	1	1	1	1

A_0A_1		Mappa di K. per A_2'			
A_2Up		00	01	11	10
	0	1	1	1	1
	0				
	1				
	1				
	0	1	1	1	1

Funzioni logiche:

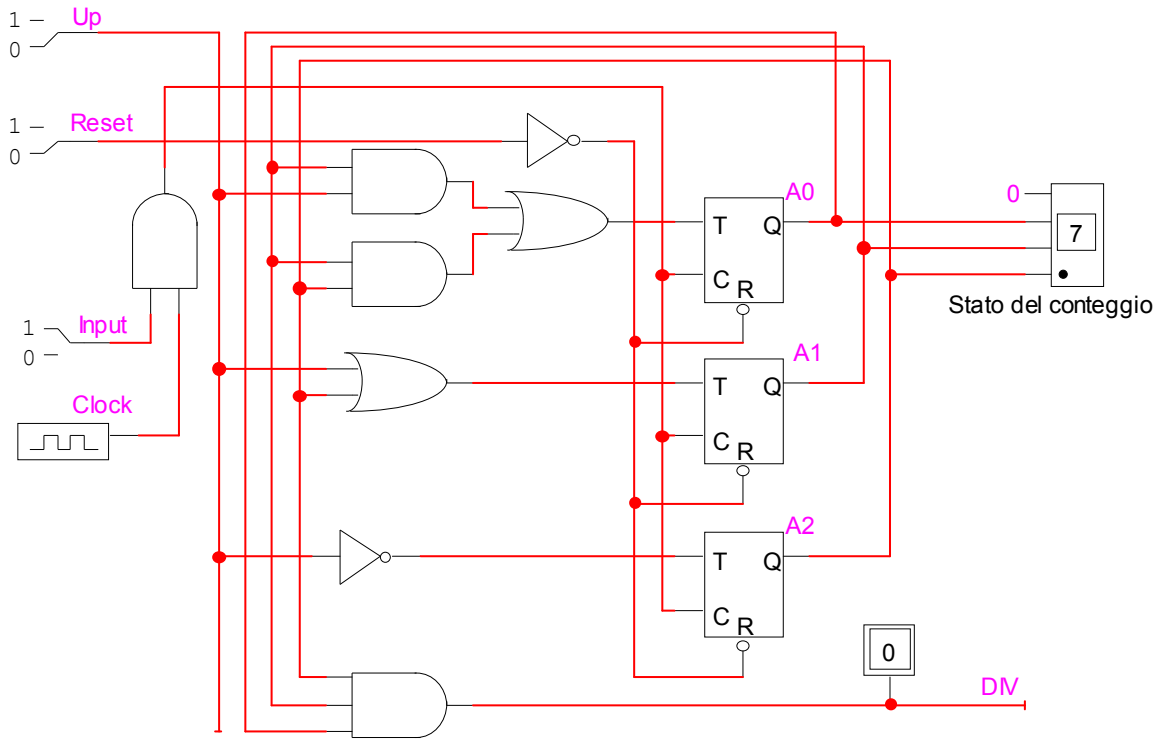
$$A_0' = A_1Up + A_2A_2$$

$$A_1' = Up + A_2$$

$$A_2' = \overline{Up}$$

$$DIV = A_0A_1A_2$$

Circuito:



APPENDICE

Il programma assembler per la USART:

```
*****
*OGGETTO: Lo scopo del programma è provare la configurazione USART.CFG      *
*costituita da due sistemi gemelli ciascuno con un processore M68000, una   *
*memoria ROM di 8k (Addr $0-1FFF), una RAM di 10k (Addr. $8000-$A7FF), un   *
*device seriale USART mappato a $2004, un device seriale di tipo TERMINAL   *
*mappato a $2000                                                            *
*ALTRO: Il presente listato va caricato in entrambi i sistemi. Assemblare  *
*da linea di comando e non con ASIMTOOL. Inizializzare il PC.              *
*Alla tastiera sono associati due tipi di interruzione:                   *
*-Interruzione su ENTER: linea 1, autovettore 25, mappata a $64, ISR a $8500 *
*-Interruzione su BUFFER FULL: linea 2, autovettore 26, mappata a $68, ISR a *
*$8600                                                                      *
*Al dispositivo USART sono associati due livelli di interruzione:           *
*-Interruzione su RxRDY: linea 3, autovettore 27, mappata a $6C,ISR a $8700 *
*-Interruzione su TxRDY: linea 4, autovettore 28, mappata a $70, ISR a $8800 *
*****
```

```

      ORG          $8200          Indirizzo di partenza del main program.
usart EQU          $2004          Interfaccia seriale.
ter   EQU          $2000          Terminale video.
```

```
START MOVEA.W      #usart,A0      Inizializza l'interfaccia seriale.
```

```
*****
*      PRIMO ACCESSO IN SCRITTURA ALLA SERIALE => REGISTRO MODE          *
*      INDIRIZZO DISPARI                                                  *
*      MODE |0 |1 |0 |1 |1 |0 |1 |                                     *
*      |     | | | | | | | | |__Trasmissione Asincrona                 *
*      |     | | | | | | | | |__Non utilizzato                         *
*      |     | | | | | | | | |__8 bit per dato                         *
*      |     | | | | | | | | |__bit di parità                          *
*      |     | | | | | | | | |__tipo di parità dispari                 *
*      |     | | | | | | | | |__2 bit di stop                          *
*      |     | | | | | | | | |__#bit di sync in trasmissione asincrona *
*****
```

```

      MOVE.B        #$5D,1(A0)      trasmissione asincrona, 8 bit di informazione
                                   bit di parità dispari e 2 bit di stop,
```

```
*****
*      SECONDO ACCESSO IN SCRITTURA ALLA SERIALE => REGISTRO CNTRL      *
*      INDIRIZZO DISPARI                                                  *
*      CNTRL |0 |0 |1 |1 |0 |1 |1 |1 |                                   *
*      |     | | | | | | | | |__Abilita trasmettitore                 *
*      |     | | | | | | | | |__Attiva DTR                             *
*      |     | | | | | | | | |__Attiva ricevitore                     *
*      |     | | | | | | | | |__Non utilizzato                         *
*      |     | | | | | | | | |__Azzera bits di errore in STATUS        *
*      |     | | | | | | | | |__Attiva RTS                             *
*      |     | | | | | | | | |__Non resetta la periferica              *
*      |     | | | | | | | | |__non va in 'hunt'                       *
*****
```

```

      MOVE.B        #$37,1(A0)      abilita trasmettitore e ricevitore, cancella flags
                                   di errore e attiva i segnali di handshaking
      MOVEA.W       #ter,A0          Inizializza il terminale video,
```

```

*****
*INIZIALIZZAZIONE DEL TERMINALE:
*
*
*   CNTRL |0|0|1|1|1|1|1|1|
*           | | | | | | | | |__Abilita interruzioni su Buffer full
*           | | | | | | | | |__Abilita interruzioni su Enter
*           | | | | | | | | |__Pulisci schermo
*           | | | | | | | | |__Pulisci buffer tastiera
*           | | | | | | | | |__Abilita tastiera
*           | | | | | | | | |__Abilita echo
*           | | | | | | | | |__Stato di buffer full
*           | | | | | | | | |__Stato di Enter inviato
*****

        MOVE.B    #$3f,1(A0)    abilita tastiera, eco, interruzioni enter e buffer
                                full, cancella video e pulisce buffer di tastiera.
        ANDI      #$F8FF,SR      Azzerata l'interrupt mask.

main    NOP
        JMP      main           Processore è in attesa di qualche interruzione.

*****
* ISR DELL'INTERRUZIONE DI LIVELLO 1 ( pressione tasto enter )
*****

        ORG      $8500          Interrupt vector.
        MOVE.L    A0,-(A7)      Salva nel supervisor stack pointer i registri
        MOVE.L    A1,-(A7)      usati dalla ISR.
        MOVE.L    D0,-(A7)

        MOVEA.W   #ter,A0       A0 e A1 puntano ai dispositivi TERMINAL e
        MOVEA.W   #usart,A1     USART rispettivamente.

*****
*   ACCESSO IN LETTURA ALL'INDIRIZZO DISPARI
*   DELLA SERIALE          =>   REGISTRO STATUS
*
*   STATUS |1|0|0|0|0|0|0|0|
*           | | | | | | | | |__b0 diviene alto quando DATA OUT
*           | | | | | | | | |__viene copiato in TSHIFT REG., torna
*           | | | | | | | | |__basso quando il processore copia
*           | | | | | | | | |__un nuovo carattere in DATA OUT
*           | | | | | | | | |__b1 diviene alto quando RSHIFT REG.
*           | | | | | | | | |__viene copiato in DATA IN, torna
*           | | | | | | | | |__basso in seguito a lettura da DATAIN
*           | | | | | | | | |__Underrun
*           | | | | | | | | |__Errore di parità
*           | | | | | | | | |__Errore di overrun
*           | | | | | | | | |__Errore di framing
*           | | | | | | | | |__Rilevati bit di sincronismo
*           | | | | | | | | |__DSR attivo
*****

wait11    MOVE.B    1(A1),D0      Controlla se è attivato il segnale DSR
                                della
                                ANDI.B    #$80,D0
                                BEQ      wait11      USART ed in caso affermativo trasmette,
                                                altrimenti attende.

wait12    MOVE.B    1(A1),D0      Se l'interfaccia seriale è pronta a
                                trasmettere
                                ANDI.B    #$01,D0
                                BEQ      wait12      continua altrimenti attende.

        MOVE.B    (A0),D0        Preleva uno alla volta i caratteri presenti nel
        MOVE.B    D0,(A1)        buffer di tastiera e li trasmette all'interfaccia

```

```

        CMPI.B      #13,D0      seriale finché non riconosce il tasto enter che
        BNE        wait12      ha causato l'interruzione.

end    MOVE.B      #$3f,1(A0)  Riabilita la tastiera, cancella il video
                                         e pulisce il buffer di tastiera.
        MOVE.L      (A7)+,D0    Ripristino dei registri
        MOVE.L      (A7)+,A1    precedentemente salvati.
        MOVE.L      (A7)+,A0
        RTE                      Ritorno alla routine interrotta.

*****
* ISR DELL'INTERRUZIONE DI LIVELLO 2 ( buffer full ) *
*****

        ORG        $8600      Interrupt vector.

        MOVE.L      A0,-(A7)    Salva nel supervisor stack pointer i registri
        MOVE.L      A1,-(A7)    usati dalla ISR.
        MOVE.L      D0,-(A7)
        MOVE.L      D1,-(A7)

        MOVEA.W     #ter,A0     A0 e A1 puntano ai dispositivi TERMINAL e
        MOVEA.W     #usart,A1   USART rispettivamente.

wait21  MOVE.B      1(A1),D0     Controlla se è attivato il segnale DSR della
        ANDI.B      #$80,D0     USART ed in caso affermativo trasmette,
        BEQ        wait21      altrimenti attende.

wait22  MOVE.B      #255,D0     Inizializza contatore caratteri nel buffer.
        MOVE.B      1(A1),D1    Se l'interfaccia seriale è pronta a
                                         trasmettere
        ANDI.B      #$01,D1     continua altrimenti attende.
        BEQ        wait22

        MOVE.B      (A0),D1     Preleva uno alla volta i 256 caratteri
                                         presenti
        MOVE.B      D1,(A1)     nel buffer di tastiera e li trasmette
                                         all'interfaccia
        DBEQ        D0,wait22   seriale.

        MOVE.B      #13,(A1)
        MOVE.B      #$3f,1(A0)  Riabilita la tastiera, cancellare il video
                                         e pulisce il buffer di tastiera.
        MOVE.L      (A7)+,D1    Ripristino dei registri
        MOVE.L      (A7)+,D0    precedentemente salvati.
        MOVE.L      (A7)+,A1
        MOVE.L      (A7)+,A0
        RTE                      Ritorno alla routine interrotta.

*****
* ISR DELL'INTERRUZIONE DI LIVELLO 3 ( la seriale ha ricevuto un carattere ) *
*****

        ORG        $8700      Interrupt vector.
        MOVE.L      A0,-(A7)    Salva nel supervisor stack pointer i registri
        MOVE.L      A1,-(A7)    usati dalla ISR.
        MOVE.L      D0,-(A7)

        MOVEA.W     #ter,A0     A0 e A1 puntano ai dispositivi TERMINAL e
        MOVEA.W     #usart,A1   USART.

        MOVE.B      1(A1),D0     Controlla se si è verificato qualche errore
                                         di
        ANDI.B      #$38,D0     trasmissione ed in caso negativo salta alla
        BEQ        stampa      routine di stampa, altrimenti visualizza #
                                         sul

```

```

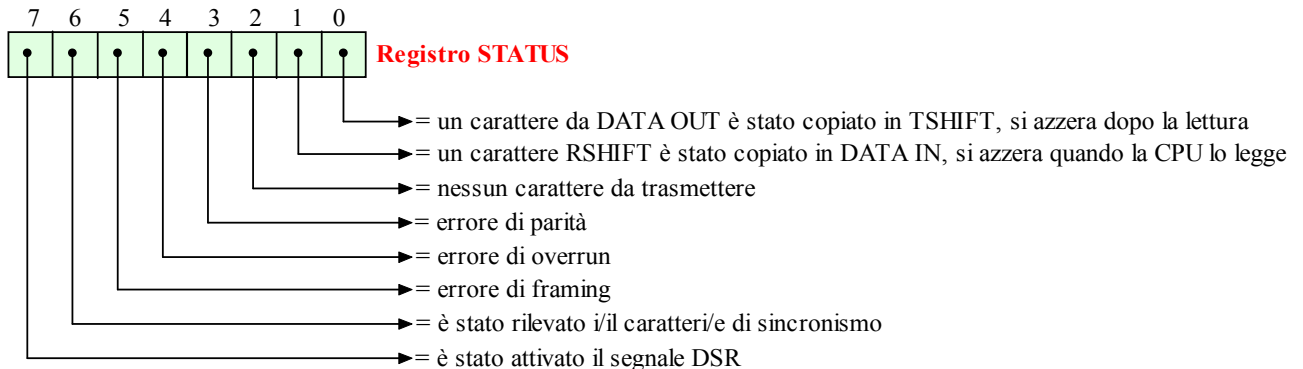
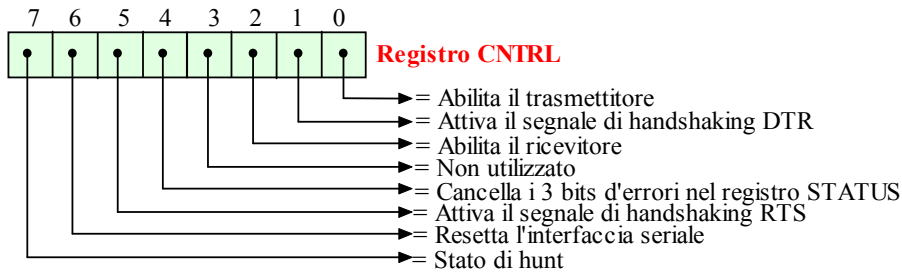
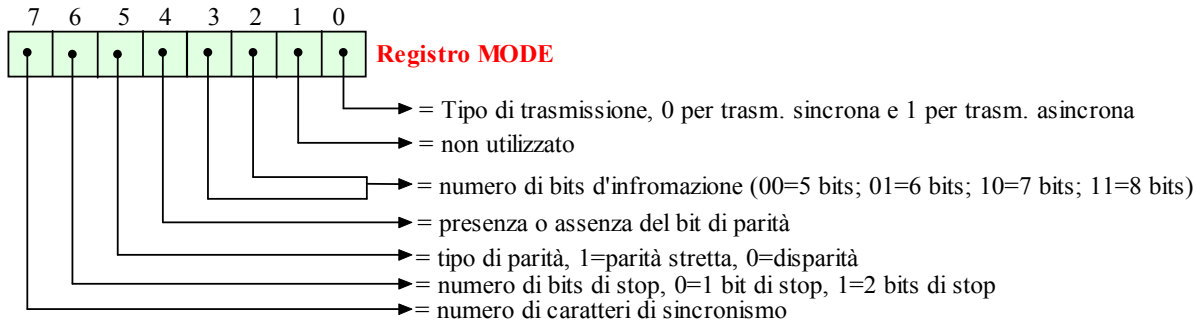
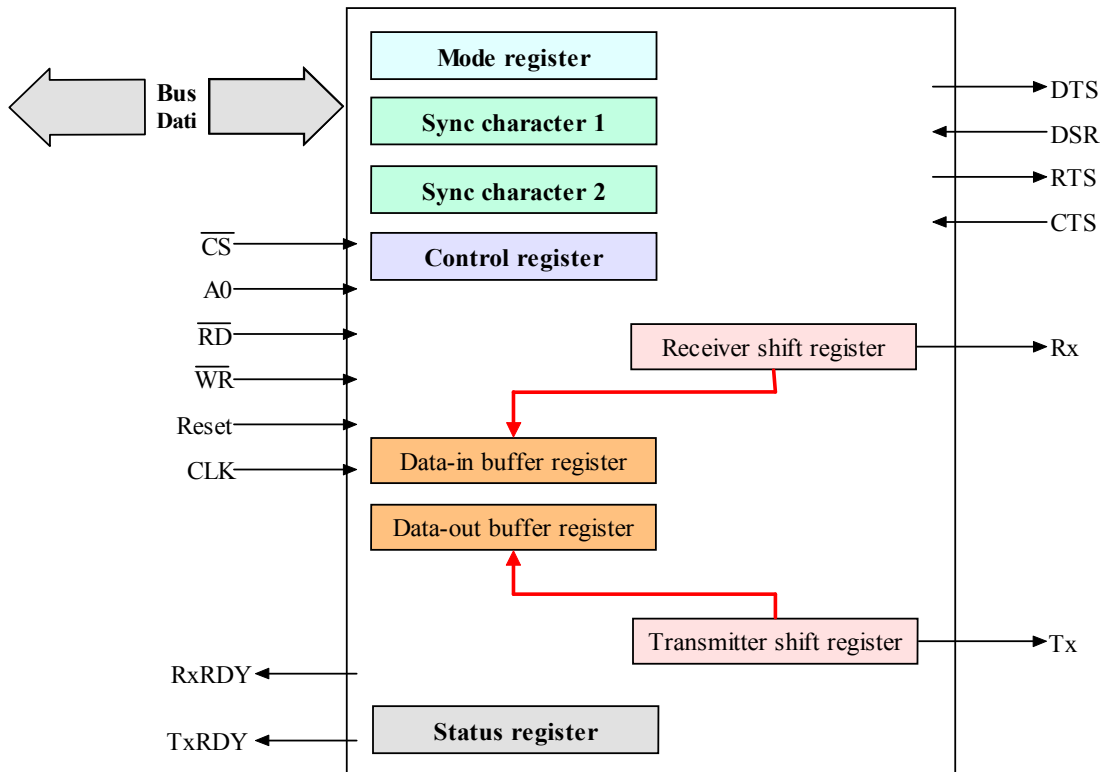
        MOVE.B      #$23,(A0)      terminale e cancella errori nel registro di
                                     stato
        MOVE.B      #$37,1(A1)    dell'interfaccia seriale.

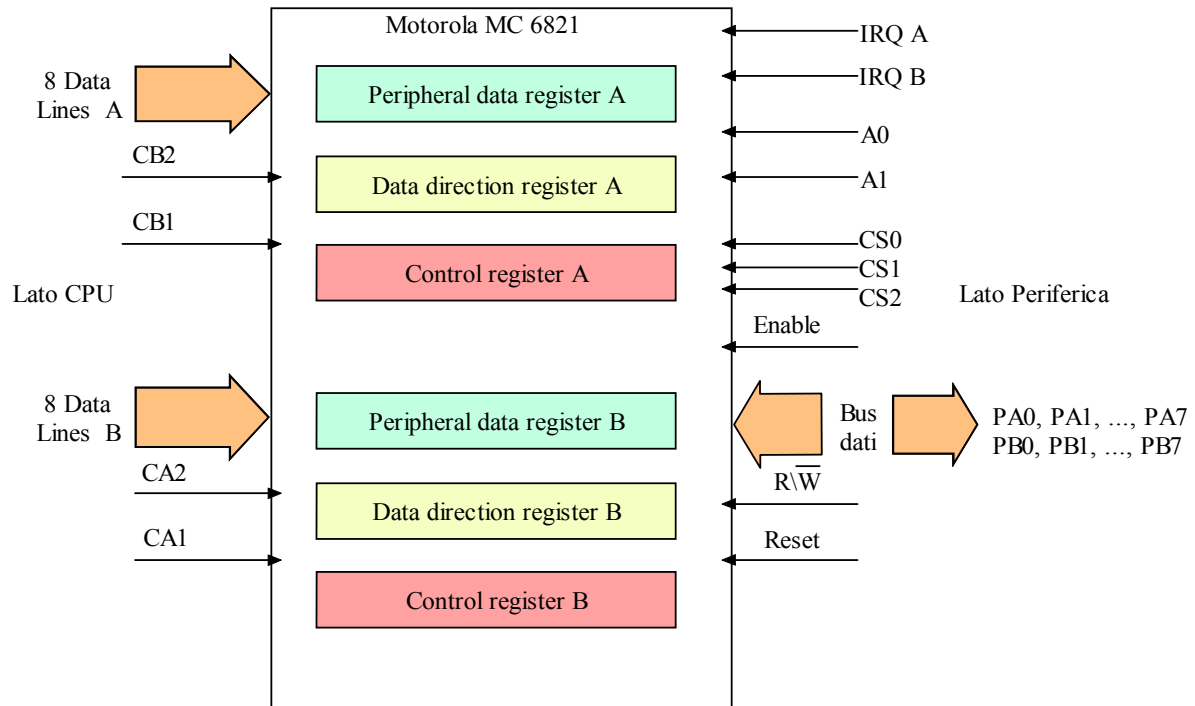
stampa  MOVE.B      (A1),(A0)      Stampa sul video il carattere ricevuto.
        MOVE.L      (A7)+,D0      Context switch : ripristino dei registri
        MOVE.L      (A7)+,A1      precedentemente salvati.
        MOVE.L      (A7)+,A0
        RTE                      Ritorno alla routine interrotta.

*****
* ISR DELL'INTERRUZIONE DI LIVELLO 4 ( la seriale ha trasmesso un carattere ) *
*****

        ORG          $8800          Interrupt vector.
        RTE          Ritorno alla routine interrotta.
END START

```





Indice degli argomenti:

LEZIONE N°1: "L'algebra di Boole"	PAG.1
-Proprietà e teoremi dell'algebra di Boole	PAG.5
-Dimostrazione di un teorema	PAG.6
-Teorema del Consensus	PAG.7
-Forma analitica per somme di prodotti	PAG.7
-Forma analitica per prodotti di somma	PAG.8
-Le mappe di Karnough	PAG.9
-Tabella delle porte logiche	PAG.13
-Le mappe di Karnough per funzioni di più quattro variabili	PAG.14
LEZIONE N°2: "La progettazione di macchine combinatorie"	PAG.15
-La rete di parità	PAG.17
-Multiplexer e multiplexer indirizzabile	PAG.19
-Decoder	PAG.21
-Full adder	PAG.22
LEZIONE N°3: "Il tastierino numerico"	PAG.24
-Introduzione	PAG.24
-Ingressi in forma codificata	PAG.26
LEZIONE N°4: "Esercizi in classe"	PAG.28
-Esercizio N°1	PAG.28
-Esercizio N°2	PAG.29
LEZIONE N°5: "La tempificazione"	PAG.31
-Alee transitorie e di regime	PAG.33
-Alea multipla	PAG.33
-Alea per impulsi concomitanti	PAG.35
-Alea statica	PAG.35
-Alea dinamica	PAG.36
-Le sequenze di ingresso	PAG.36
LEZIONE N°6: "Macchine e reti sequenziali"	PAG.39
-Il modello di Mealy	PAG.39
-Il modello di Moore	PAG.39
-Schema ideale di macchina sequenziale	PAG.40
-L'elemento di ritardo	PAG.41
-Il concetto di stato stabile	PAG.41
-Definizione di macchina asincrona	PAG.41
-Definizione di macchina sincrona	PAG.42
-Il registro	PAG.42
-Il flip flop RS fondamentale	PAG.43
LEZIONE N°7: "Il flip flop D Latch, primi esempi di progettazione"	PAG.45
-Il flip flop D latch	PAG.45
-Schema ideale di rete autosincronizzata	PAG.47
-Schema ideale di rete a sincronizzazione esterna	PAG.48
-Le sequenze di ingresso impulsive	PAG.49
-Le sequenze di ingresso a livello	PAG.49
-Le sequenze a sincronizzazione esterna	PAG.50
-Le sequenze impulsive autosincronizzata	PAG.50
-Esempi di progettazione di macchine sequenziali	PAG.51
-Riconoscitore di sequenza 101 Ver.A	PAG.51
-Riconoscitore di sequenza 101 Ver.B	PAG.51
-Riconoscitore 8421 Ver.A	PAG.58
-Riconoscitore 8421 Ver.B	PAG.62
LEZIONE N°8: "La tempificazione edge trigger e master slave, il FF JK"	PAG.66
-Il flip flop JK	PAG.66
-Il flip flop D ETS	PAG.66
-Il flip flop D ETD	PAG.68
-Il flip flop T	PAG.72
-Il flip flop T ETS	PAG.72
-Il flip flop T ETD	PAG.73

-Il flip flop JK	PAG.73
-Il problema del pilotaggio di un flip flop (esempi)	PAG.74
-La tempificazione master slave	PAG.77
-Il flip flop D master slave	PAG.78
-Il flip flop T master slave	PAG.80
LEZIONE N°9: "I contatori"	PAG.82
-Classificazione di un contatore	PAG.83
-Esempio N°1, progetto di un contatore modulo 8 con FF D	PAG.84
-Esempio N°2, progetto di un contatore modulo 8 con FF T	PAG.87
-Approfondimento, progettazione di contatori con FF JK o FF T	PAG.89
-Esempio N°3, contatore modulo 8 bilaterale con FF D ETS	PAG.90
-Esempio N°4, contatore modulo 8 bilaterale con FF T ETS	PAG.93
-Il componente contatore il logic works	PAG.96
-Composizione di contatori	PAG.97
LEZIONE N°10: "La progettazione dei sistemi"	PAG.99
-Composizione di macchine combinatorie	PAG.99
-Composizione per semiselezione di decodificatori	PAG.100
-Composizione ad albero di decodificatori	PAG.100
-Composizione di multiplexer	PAG.101
-Composizione di macchine sequenziali	PAG.103
-Riconoscitore 8421 per composizione	PAG.104
LEZIONE N°11: "I registri a scorrimenti"	PAG.108
-La rete di parità sequenziale	PAG.110
LEZIONE N°12: "Il progetto di macchine per composizione"	PAG.115
-Esempio	PAG.115
LEZIONE N°13: "Dati in forma seriale e/o parallela"	PAG.120
-Addizionatore binario	PAG.123
LEZIONE N°14: "Il problema dell'I/O"	PAG.127
-L'interfaccia	PAG.128
-Protocollo sincrono e asincrono	PAG.130
-Meccanismi di I/O	PAG.132
-Le interruzioni	PAG.132
-Cosa fare se ci sono più dispositivi?	PAG.137
-Cosa è il vettore delle interruzioni?	PAG.138
-Simulazione di una interfaccia seriale in ASIM	PAG.139
-Configurazione	PAG.140
-Collegamento al processore	PAG.141
-Collegamento a un device	PAG.142
-Collegamento software	PAG.145
-Modello di programmazione	PAG.147
-Programmazione	PAG.150
-Il comportamento	PAG.152
-Trasmissione asincrona	PAG.152
-Trasmissione sincrona	PAG.153
-Ricezione asincrona	PAG.154
-Ricezione sincrona	PAG.154
LEZIONE N°15: "Le memorie"	PAG.155
LEZIONE N°16: "La comunicazione parallela"	PAG.167
LEZIONE N°17: "Alcune note sull'interfacciamento e sistemi di driver"	PAG.173
-Esempio 1	PAG.176
-Esempio 2	PAG.179
-Esempio 3	PAG.180
Alcuni esercizi (Macchine sequenziali)	PAG.182
-Esercizio 1	PAG.182
-Esercizio 2	PAG.184
-Esercizio 3	PAG.186

-Esercizio 4	PAG.187
-Esercizio 5	PAG.188
-Esercizio 6	PAG.189
-Esercizio 7	PAG.190
-Esercizio 8	PAG.192

NOTE	PAG.204
------	---------

NOTE:

Questi appunti sono stati scritti considerando le lezioni svolte e raccogliendo informazioni dai vari libri di testo consigliati per il corso di reti logiche ed interfacciamento di sistemi a microprocessori dell'anno accademico 2003/2004. Nell'anno accademico 2004/2005 alcuni argomenti sono stati approfonditi e trattati con più di qualche esempio. Gli argomenti trattati sono pertanto una sintesi di quelli svolti a lezione per cui è necessario un ulteriore confronto ed approfondimento sui libri di testo per avere la piena conoscenza del singolo argomento.

Luca Petrosino

E-Mail: *luca.petrosino@tiscali.it*