

# **Modulo Calcolatori Elettronici**

**Proff. Gian Luca Marcialis, Giulia Orrù, Lorenzo Putzu, Fabio Roli**

## **Corsi di Laurea in Ingegneria Biomedica Ingegneria Elettrica, Elettronica ed Informatica**

---

### **Contatti:**

[marcialis@unica.it](mailto:marcialis@unica.it), [giulia.orrù@unica.it](mailto:giulia.orrù@unica.it), [lorenzo.putzu@unica.it](mailto:lorenzo.putzu@unica.it)

## **Capitolo 2**

### **Reti Logiche**

**Fonti principali:** Tanenbaum, A., "Architettura dei computer: un approccio strutturato", Cap. 3.

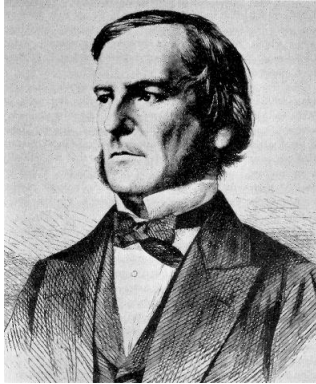
# Sommario

---

- Algebra Booleana
  - Definizioni ed elementi di base
  - Porte logiche
  - Funzioni booleane
- Reti combinatorie
  - Analisi e sintesi
  - Esempi di reti combinatorie
- Reti sequenziali sincrone
  - Elementi di memoria e sincronizzazione
  - Analisi e sintesi
  - Esempi di reti sequenziali sincrone

# Algebra booleana

---



- Nel 1854 **George Boole** propose i principi di base di questa algebra che è poi diventata l'algebra di riferimento per rappresentare e risolvere problemi di analisi e di progetto dei circuiti elettronici digitali.

# Algebra booleana

---

## Concetti base

- Si usano:
  - Variabili logiche il cui valore è 1 (true) o 0 (false).
  - Operatori che agiscono su tali variabili.
  - Tabelle di verità per definire gli operatori in funzione del loro “funzionamento” sugli operandi.
    - Elencano i valori assunti dall’operatore per tutte le possibili combinazioni dei valori degli operandi (variabili).

# Operatori logici di base

- $P \text{ AND } Q = P \bullet Q$ 
  - Il risultato è 1 se e solo se entrambi gli operandi valgono 1.
- $P \text{ OR } Q = P + Q$ 
  - Il risultato è 1 se almeno uno degli operandi vale 1.
- $\text{NOT } P = \overline{P} = P'$ 
  - Inverte ("complementa") il valore del suo operando.

## Tabella di verità per i tre operatori

P	Q	NOT P	P AND Q	P OR Q
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

# Algebra Booleana o della Commutazione

---

Sistema algebrico definito da:

1. Variabili binarie
2. Un operatore NOT (come definito prima)
3. Un operatore AND (come definito prima)
4. Un operatore OR (come definito prima)
5. Una relazione di equivalenza “=” fra espressioni booleane.  
Due espressioni sono equivalenti se e solo se assumono lo stesso valore per ogni valore delle variabili

# Espressioni Booleane

---

Espressione booleana: una combinazione di variabili booleane e costanti (0, 1) definita per mezzo di operatori logici

$$\text{Es. } E = x y + x z' + y = y (x + 1) + x z' = y + x z'$$

Un'espressione booleana  $E$  rappresenta quella funzione booleana  $F$  che vale 1 per tutte quelle assegnazioni delle variabili per le quali  $E=1$  (e viceversa  $E=0$  se e solo se  $F=0$ )

E' facile capire che esistono molte diverse espressioni possibili per una data funzione booleana. Come per le normali espressioni algebriche è possibile, usando le proprietà algebriche, passare da una espressione ad un'altra equivalente

# Espressioni Booleane

---

**Letterale:** ogni presenza di una variabile diretta o negata in una espressione

Es.  $a b + c d' + a'$

Contiene cinque letterali

**Livelli** di una espressione

Numero di operatori che agiscono in cascata sui letterali che compaiono in una espressione

Es.  $a b + c d$  È una espressione a due livelli

Nota: l'operatore NOT non viene solitamente conteggiato nel numero dei livelli



# **Relazioni notevoli dell'algebra booleana**

- Si noti la dualità tra gli operatori AND e OR

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

# Altri importanti operatori

---

- $P \text{ XOR } Q$ 
  - Detto anche OR esclusivo.
  - Vale 1 quando gli operandi sono diversi.
- $P \text{ NAND } Q = \text{NOT } (P \text{ AND } Q) = \overline{PQ}$ 
  - E' il complemento dell' AND.
- $P \text{ NOR } Q = \text{NOT } (P + Q) = \overline{P + Q}$ 
  - E' il complemento dell'OR.

P	Q	P XOR Q	P NAND Q	P NOR Q
0	0	0	1	1
0	1	1	1	0
1	0	1	1	0
1	1	0	0	0

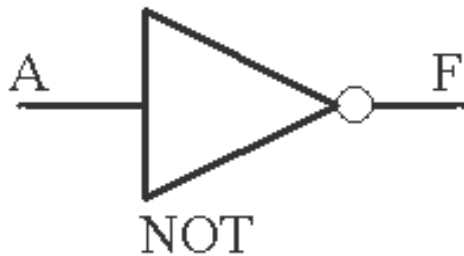
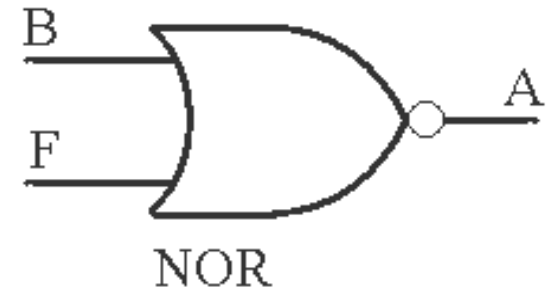
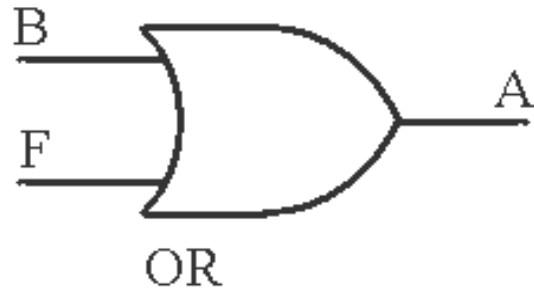
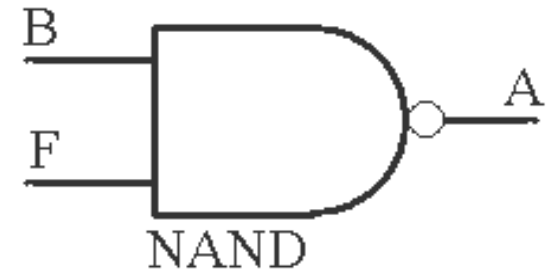
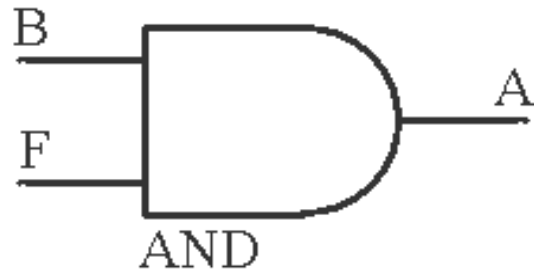
# Porte logiche

---

- Sono circuiti elettronici che producono come “segnale” di uscita il risultato di un’operazione booleana sui suoi ingressi. Possiamo quindi considerarle l’implementazione circuitale degli operatori logici booleani
- Ingressi ed uscite delle porte logiche sono segnali elettrici digitali. In questo corso non ci occuperemo della realizzazione circuitale. Considereremo pertanto ingressi ed uscite solo dal punto di vista “logico”
- Ogni porta è definita come simbolo grafico, notazione algebrica, o con la tabella di verità.
  - Secondo lo standard IEEE.
- Gli ingressi alla porta possono essere 1, 2 o più.

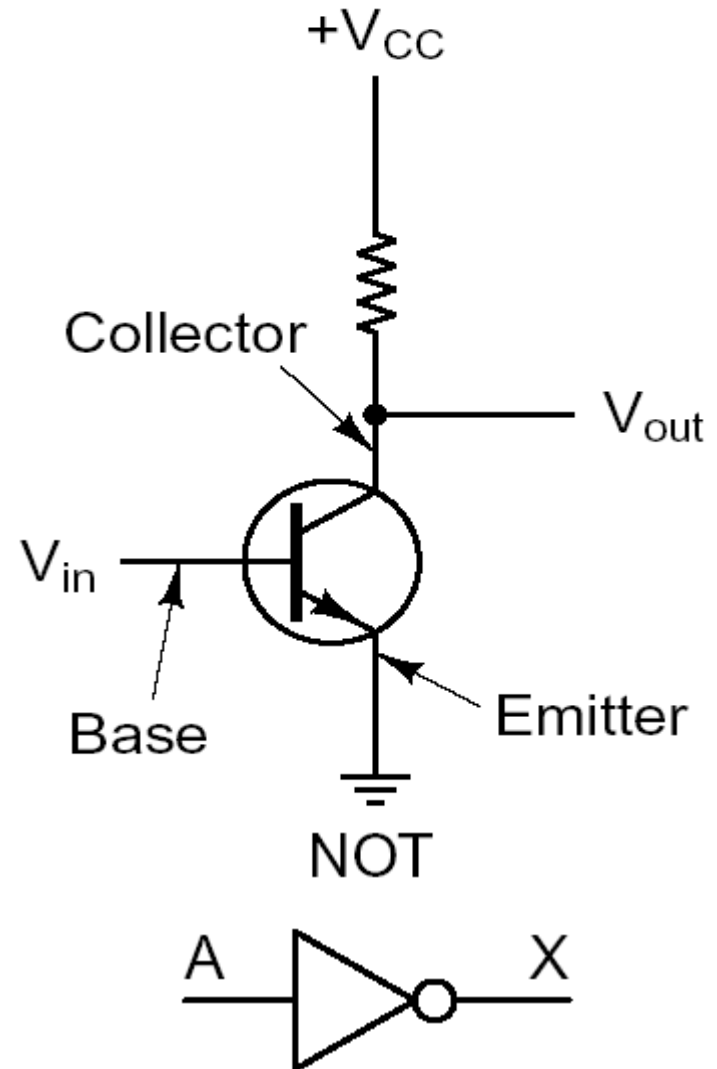
# Simboli grafici di alcune porte logiche

---



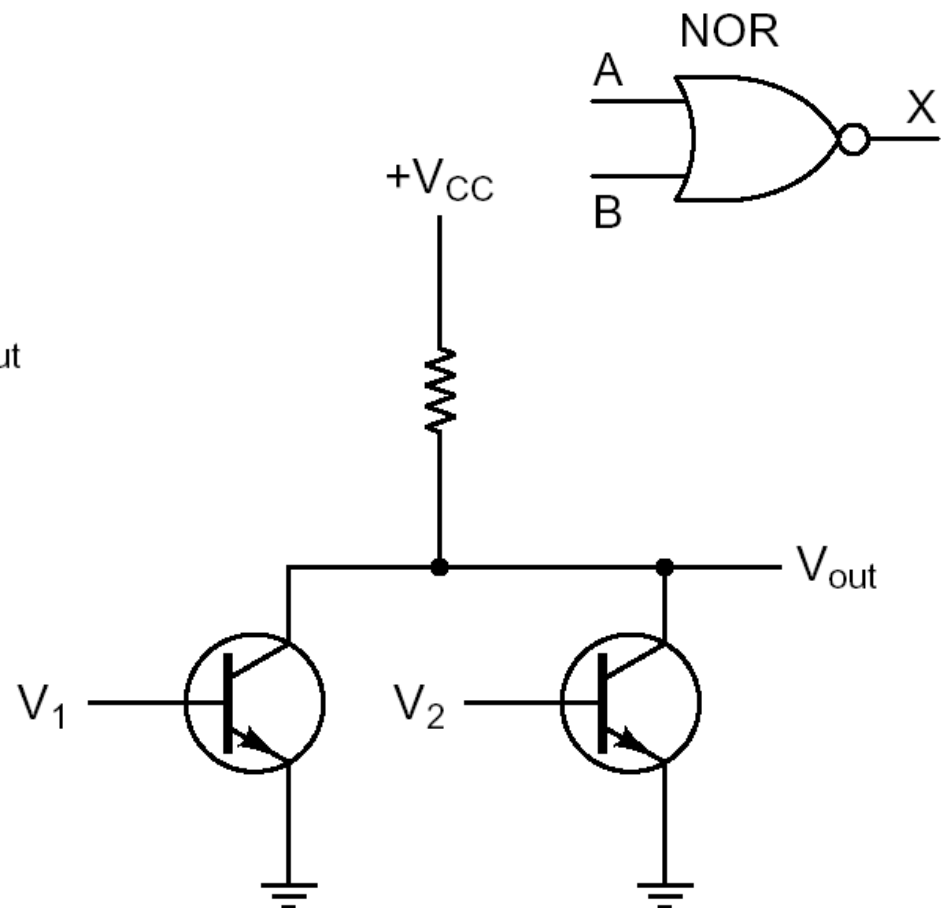
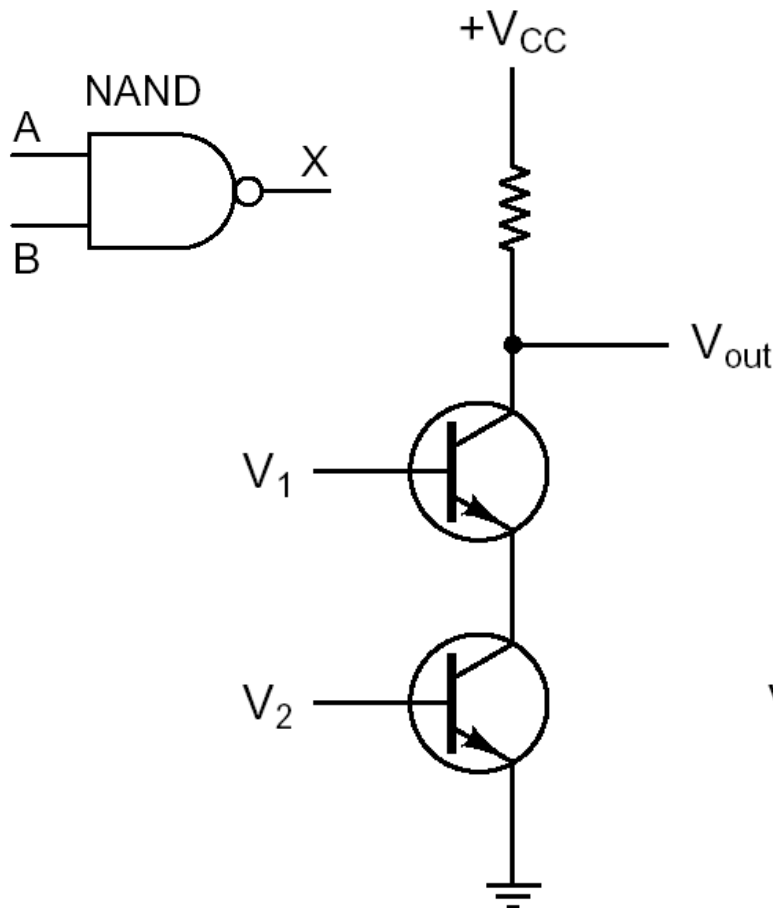
# Cenni alla realizzazione circuitale delle porte logiche

- Le porte logiche vengono realizzate attraverso dispositivi elettronici chiamati "transistor"
- Questi dispositivi, opportunamente alimentati, sono in grado di fornire segnali interpretabili come bit: basso livello di segnale come 0, alto livello di segnale come 1
- A fianco mostriamo un esempio di porta NOT realizzata con transistor "bipolari"
- Quando  $V_{in}$  è superiore a una certa soglia, il transistor funziona da corto circuito
- Altrimenti il transistor funziona da circuito aperto



# Porte NAND e NOT con transistor

- Per il funzionamento "naturale" dei transistor, è più facile realizzare dispositivi in "logica negata" (porte NAND e NOR) che in logica non negata (porte AND e OR)



# Funzioni booleane

---

- Forme canoniche:
  - Prodotti di Somme (PS)
    - Vi è un termine per ogni valore 0 dell'uscita nella tabella di verità
    - Ogni termine viene chiamato "maxtermine"
    - Es.:  $F = (A+B) \cdot (A+\overline{B}) \cdot (\overline{A}+B)$
  - Somme di Prodotti (SP)
    - Vi è un termine per ogni valore 1 dell'uscita nella tabella di verità
    - Ogni termine viene chiamato "mintermine"
    - Es.:  $F = \overline{A}BC + A\overline{B}C + ABC$
- Si passa da una forma all'altra applicando il teorema di De Morgan.

# Insiemi di operatori/porte funzionalmente completi

---

- Sono sufficienti per rappresentare tutte le funzioni booleane:
  - **AND, OR, NOT**
    - Infatti sono i tre operatori di base dell'algebra booleana.
  - **AND, NOT**
    - Possono essere implementate con un'unica porta NAND.
  - **NAND**
    - Per il teorema di De Morgan:  $A + B = \overline{\overline{A} \bullet \overline{B}}$
  - **OR, NOT**
    - Possono essere implementate con un'unica porta NOR.
  - **NOR**
    - Per il teorema di De Morgan:  $A \bullet B = \overline{\overline{A} + \overline{B}}$



# Reti logiche

---

- Una rete logica è un insieme di blocchi funzionali realizzati mediante porte logiche ed elementi di memoria
- E' caratterizzata da  $n$  variabili "di ingresso" e  $m$  variabili "di uscita"
- Le reti logiche si dividono in:
  - reti senza memoria, o reti combinatorie
  - reti con memoria, o reti sequenziali



# Reti combinatorie

---

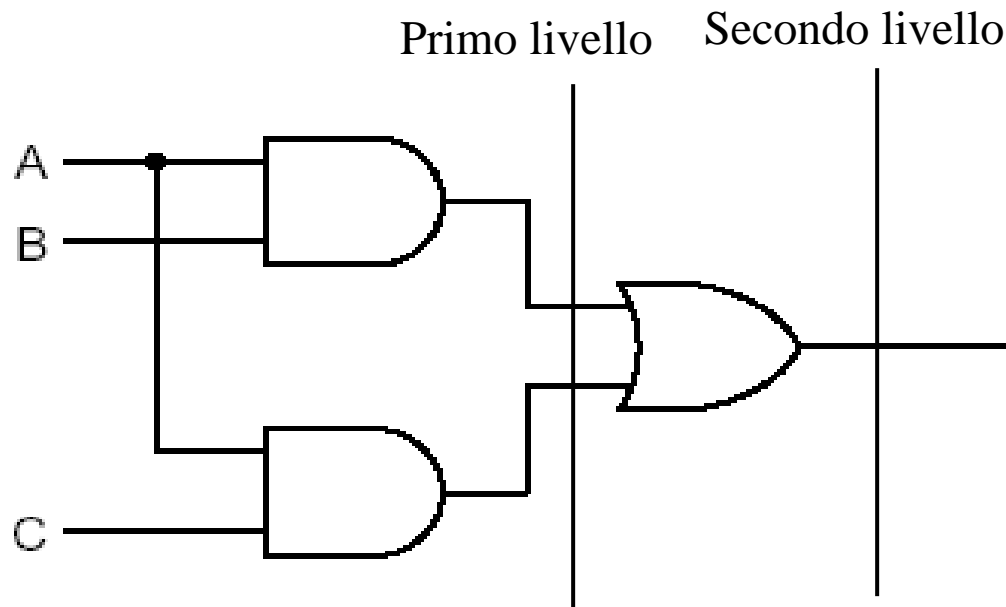
- Una rete combinatoria è un insieme di porte logiche che realizzano una certa funzione booleana
- E' caratterizzata da  $n$  variabili di ingresso e  $m$  variabili di uscita
- Per ognuna delle  $2^n$  combinazioni degli ingressi è fissato il valore delle uscite
- In un dato istante, l'uscita dipende *solo* dal valore degli ingressi della rete



# **Livelli di una rete combinatoria**

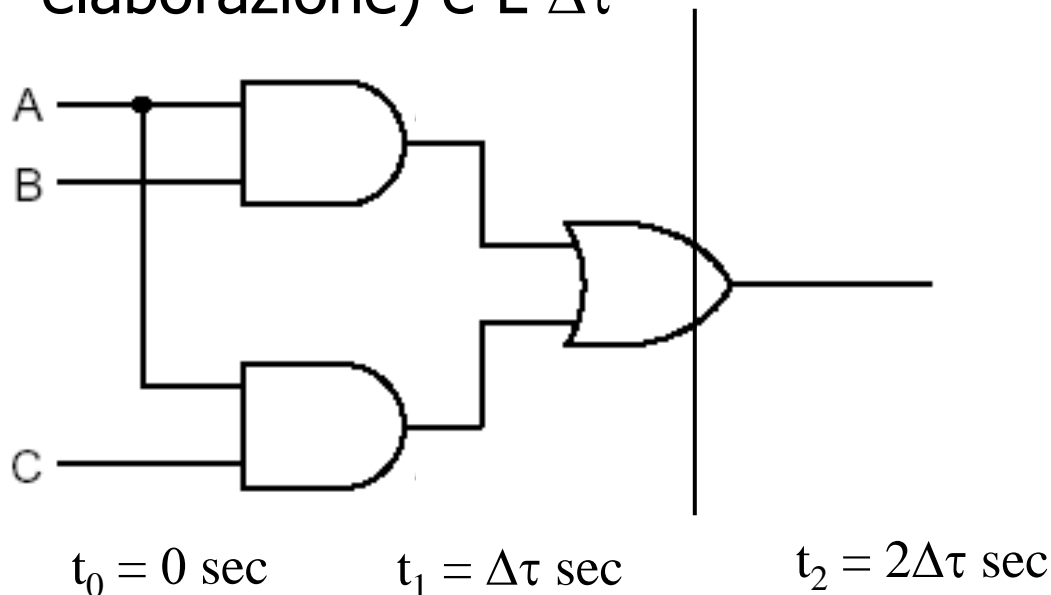
---

- Una rete combinatoria è caratterizzata da un determinato numero di "livelli" di porte logiche
- Un livello è costituito dalle porte i cui ingressi ricevono i "bit" ("segnali" elettrici) nello stesso istante
- Spesso una rete combinatoria non presenta più di tre livelli di logica (porte NOT + porte AND + porte OR)



# Ritardi in una rete combinatoria

- Il numero di livelli di una rete combinatoria caratterizza il tempo di elaborazione della rete. O "ritardo" introdotto dalla rete
- Es. se per ogni livello di logica il ritardo fra ingresso ed uscita è  $\Delta\tau$ , e la rete combinatoria presenta L livelli di logica, allora il ritardo complessivo (o tempo di elaborazione) è  $L \Delta\tau$



•E' chiaro che conviene minimizzare il numero di livelli della rete, in modo da minimizzare il ritardo (tempo di elaborazione) della rete

•Noi useremo reti a 2 livelli sintetizzate con espressioni SP

# **Analisi e sintesi di una rete combinatoria**

---

- **Analisi**

- dall'esame delle porte logiche che compongono la rete capire quale è la funzione implementata

- **Sintesi**

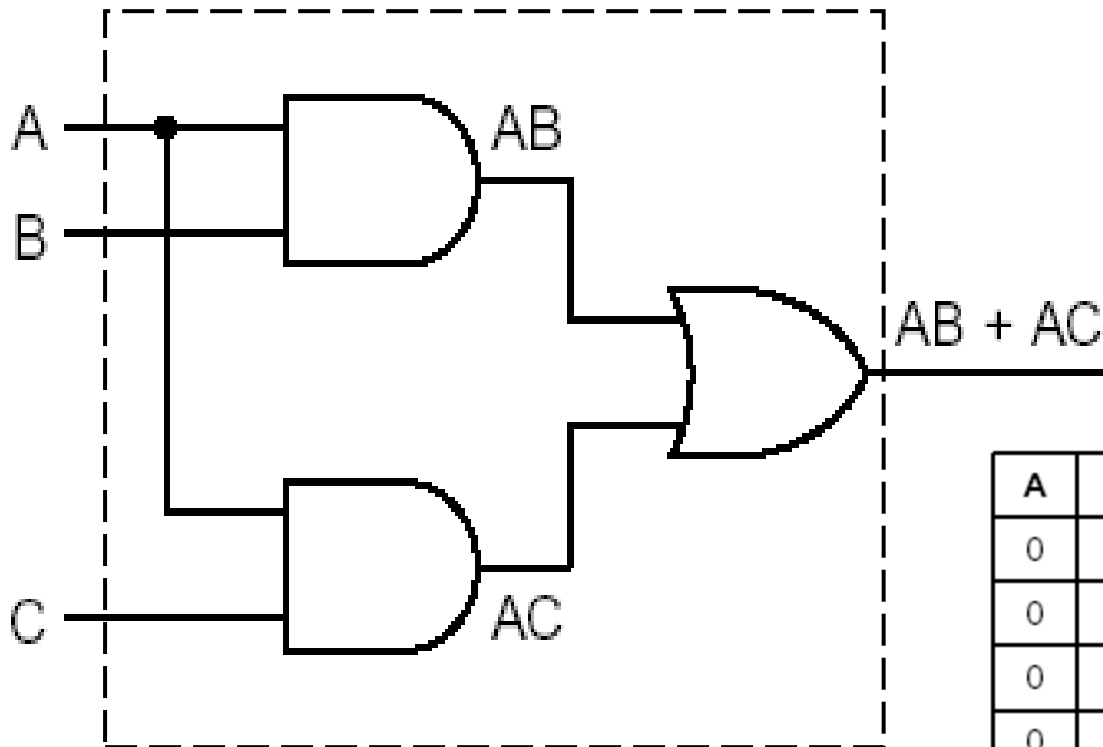
- dall'analisi dei requisiti, ovvero dalle corrispondenze ingressi-uscite, implementare la funzione con il minimo numero possibile di porte logiche ed ingressi. In altre parole, dalla tabella di verità sintetizzare la rete "minima"

- si desidera che il numero di porte/ingressi sia il più piccolo possibile

- riduzione di spazio sul "chip" (circuito integrato)

- costo del 'chip'

# Esempio di analisi di reti combinatorie



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

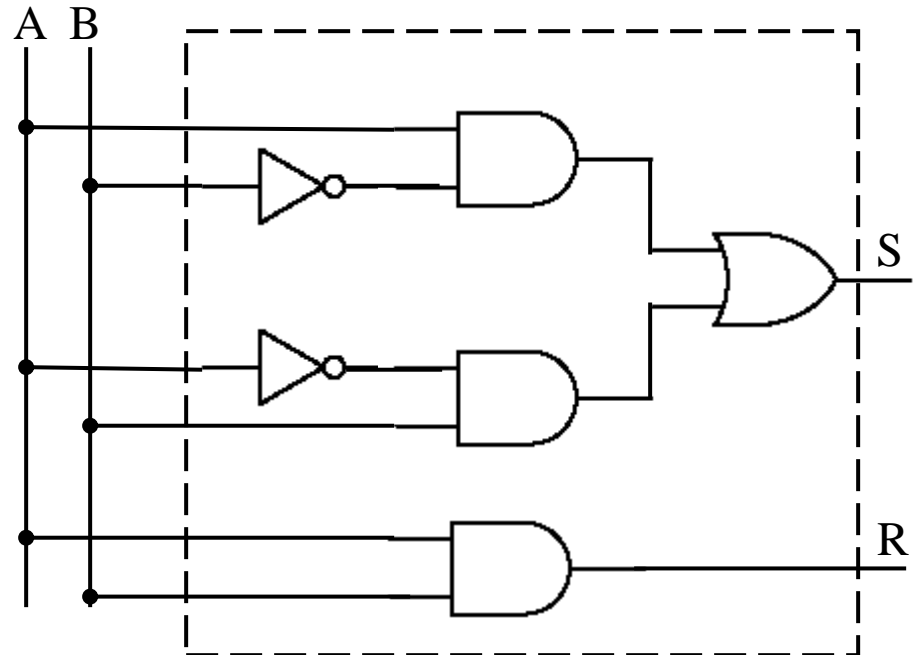
# Esempio di sintesi di rete combinatoria

- Progettare una rete combinatoria che, dati due ingressi A e B, presenti in uscita somma e riporto (half adder)

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

$$R = AB$$



# Sintesi minima di una rete logica:

## **Minimizzazione delle funzioni booleane**

---

- Significa trasformare una generica funzione booleana in una equivalente che abbia il minor numero possibile di termini (ovvero di porte) e di letterali (ovvero di ingressi).
- Possono essere usati tre metodi:
  - Semplificazione algebrica
    - Si usano le relazioni notevoli viste per ridurre un'espressione booleana in una con il minor numero di termini possibile.
  - **Mappe di Karnaugh**
    - Si basa sulla rappresentazione di una funzione su una mappa (tabella) di  $2^n$  "celle" ( $n$  è il numero di variabili/ingressi alla rete).
  - Tabelle di Quine-McKluskey
    - Si basa sul confronto dei termini e la costruzione di successive tabelle in cui i termini hanno un numero sempre minore di variabili. Non lo vediamo in questo corso.



# Metodo delle mappe di Karnaugh

---



- E' un metodo sistematico per minimizzare, in modo manuale, funzioni con poche variabili (al più 6). Ovviamente lo posso implementare su calcolatore per applicarlo a più variabili.

Dott. Maurice Karnaugh

<http://www.ithistory.org/honor-roll/dr-maurice-karnaugh>

# Metodo delle mappe di Karnaugh

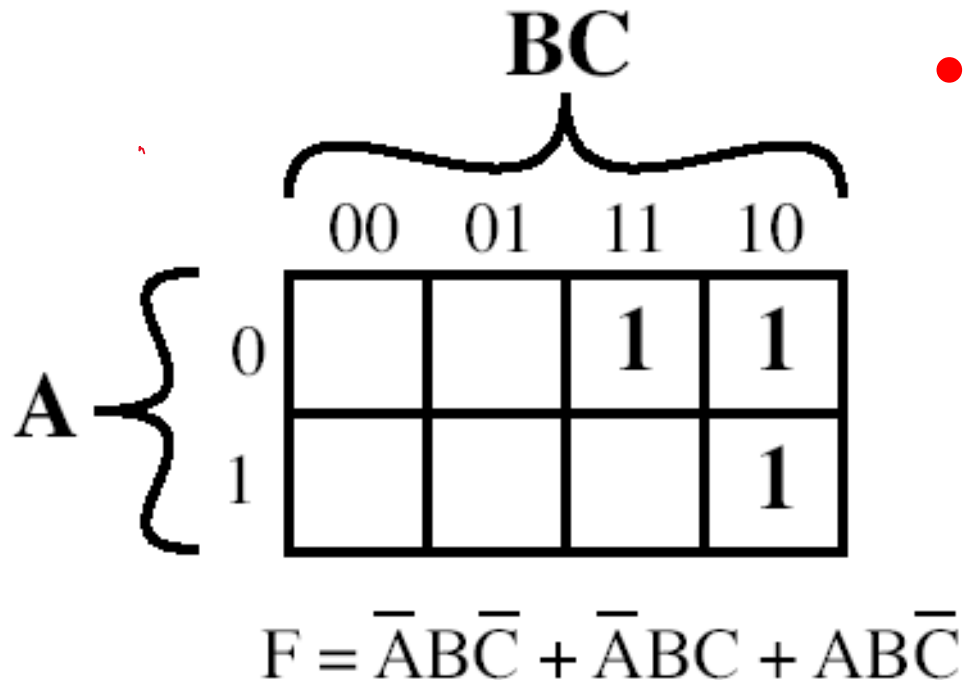
---

## Concetti base

- La tabella rappresenta tutti i valori della funzione per le  $2^n$  combinazioni delle  $n$  variabili.
- Per costruire la mappa si deve avere:
  - la forma canonica
  - la tabella di verità
- Se la funzione è scritta in forma SP, ogni cella rappresentante un mintermine vale 1
- Se la funzione è scritta in forma PS, ogni cella rappresentante un maxtermine vale 0

# Mappe di Karnaugh

- Noi utilizzeremo mappe di Karnaugh a tre ed a quattro variabili in rappresentazione **SP**
- Esempio di mappa di Karnaugh a tre variabili:



- **Importante:**

- Le celle rappresentano combinazioni di variabili
- Nelle celle adiacenti tali combinazioni differiscono per *un solo bit*

# Mappa di Karnaugh a quattro variabili

		CD			
		00	01	11	10
AB	00			1	
	01				
	11	1			
	10		1		

$$F = \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D}$$

# Definizioni

- Siano  $p$  ed  $f$  due funzioni booleane
- **Implicante** di  $f$ 
  - si dice che  $p$  è implicante di  $f$ , se  $f = 1$  quando  $p=1$
- **Implicante primo** di  $f$ 
  - si dice che  $p$  è implicante primo di  $f$  se non esiste un altro implicante  $p'$  di  $f$  con meno letterali che implichi  $p$

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

p

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

# Definizioni

- Siano  $p$  ed  $f$  due funzioni booleane
- **Implicante** di  $f$ 
  - si dice che  $p$  è implicante di  $f$ , se  $f = 1$  quando  $p = 1$
- **Implicante primo** di  $f$ 
  - si dice che  $p$  è implicante primo di  $f$  se non esiste un altro implicante  $p'$  di  $f$  con meno letterali che implichi  $p$

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

p

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

# Implicanti primi non ridondanti

- Implicante primo non ridondante (o essenziale)
  - p si dice implicante primo non ridondante di f, se esiste almeno un vertice del sottocubo di p non coperto da altri implicanti primi
- Per ogni funzione f, esiste almeno un insieme di implicanti primi *non ridondanti*  $Q = \{p_1, \dots, p_q\}$  tale che:

$$f = \bigwedge_{p_i \in Q} p_i$$

- Esempio:

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

implicante primo ridondante

Espressione con implicanti primi ridondanti:

$$f = AB + BC + \overline{AC}$$

Espressione con implicanti primi **non** ridondanti:

$$f = BC + \overline{AC}$$

# Semplificazioni con mappe di Karnaugh: relazioni con l'algebra booleana

		BC			
		00	01	11	10
A	0			1	1
	1				1

- La mappa a fianco rappresenta la funzione

$$F = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C}$$

- Una prima semplificazione può essere fatta applicando la proprietà distributiva e dell'elemento inverso:

$$F = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\overline{C} = \overline{A}B(C + \overline{C}) + A\overline{B}\overline{C} = \overline{A}B + A\overline{B}\overline{C}$$



# Semplificazioni con mappe di Karnaugh: relazioni con l'algebra booleana

		BC			
		00	01	11	10
A	0			1	1
	1				1

- La precedente semplificazione algebrica può essere rappresentata graficamente cerchiando i due termini coinvolti
- Motivo: aver unito i due mintermini coincide con la semplificazione del letterale C. Sto facendo "graficamente" la semplificazione algebrica

$$F = \overline{A}B(C + \overline{C}) + A\overline{B}\overline{C} = \overline{A}B + A\overline{B}\overline{C}$$

- Avremmo potuto anche semplificare così:

$$F = \overline{A}BC + \overline{B}\overline{C}(\overline{A} + A) = \overline{A}BC + \overline{B}\overline{C}$$

- Sarebbe stato equivalente alla "cerchiatura" a fianco

		BC			
		00	01	11	10
A	0			1	1
	1				1

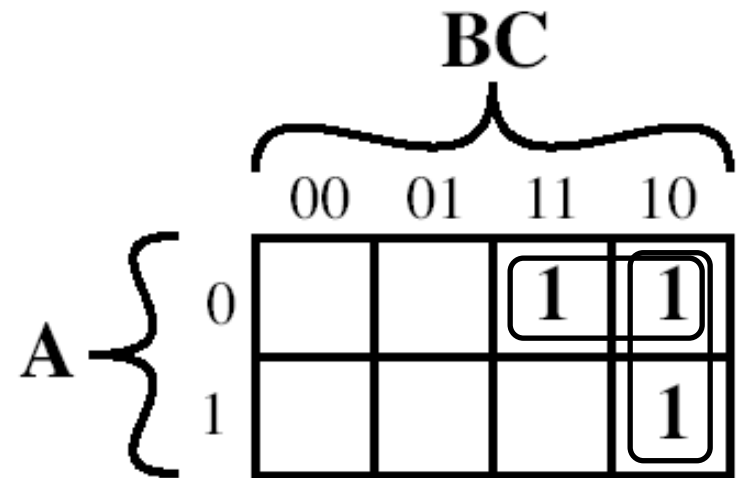
# Semplificazione con mappe di Karnaugh: relazioni con l'algebra booleana

- Notiamo che, usando le proprietà algebriche, si può ulteriormente semplificare:—

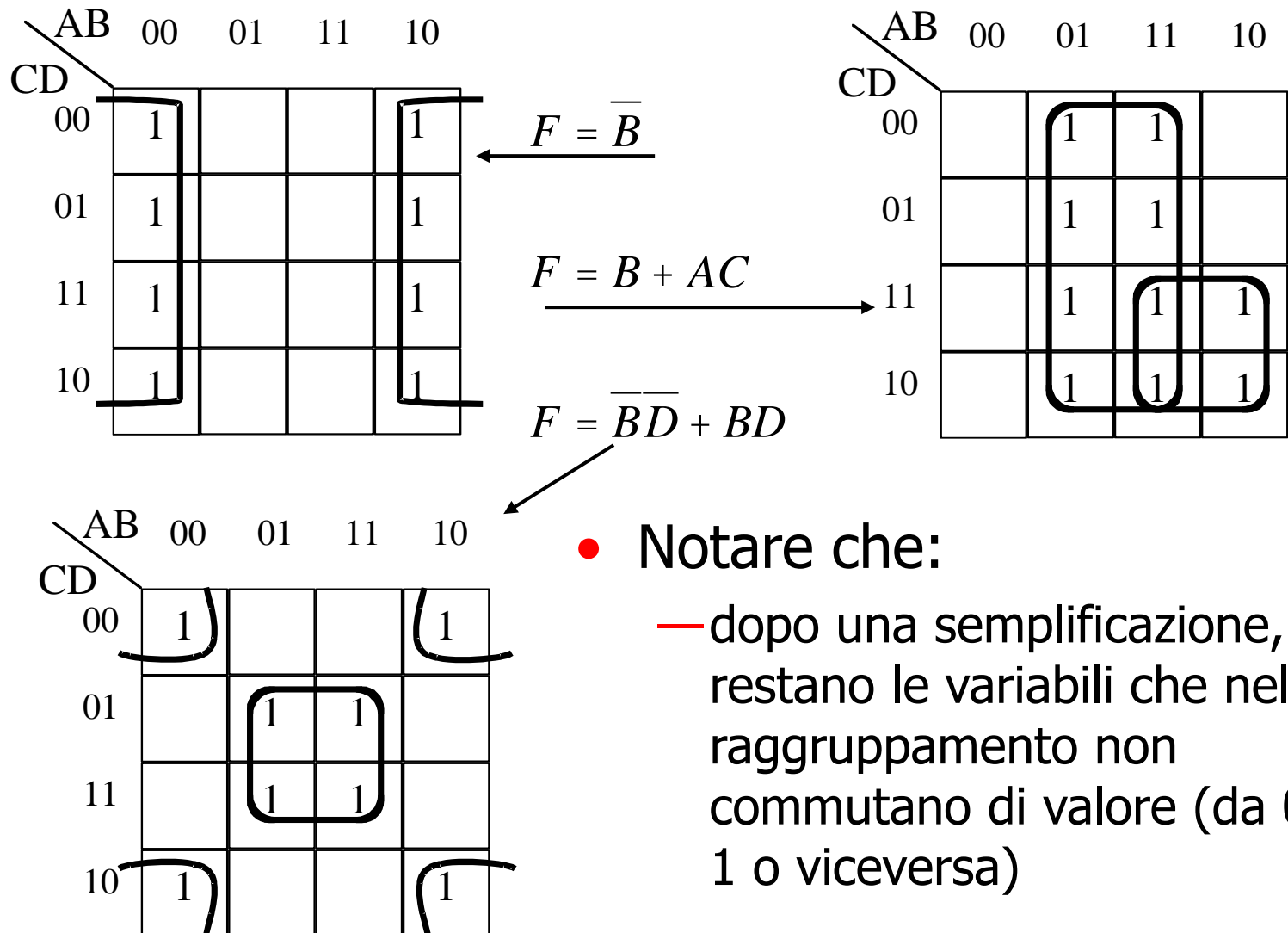
$$\begin{aligned} F &= AB + ABC = B \times (A + A \times \bar{C}) = \\ &= B \times (\bar{A} + A) \times (\bar{A} + \bar{C}) = B \times (\bar{A} + \bar{C}) = \bar{A}B + B\bar{C} \end{aligned}$$

- E' possibile allora "sovrapporre" le due "cerchiature" e scrivere semplicemente:

$$F = \bar{A}B + B\bar{C}$$



# Altri esempi di semplificazioni



• Notare che:

- dopo una semplificazione, restano le variabili che nel raggruppamento non commutano di valore (da 0 a 1 o viceversa)

# Punti importanti

---

- Nelle mappe di Karnaugh gli “1” corrispondono a mintermini, ed i loro “raggruppamenti” a implicant
- Mediante le mappe di Karnaugh posso semplificare le espressioni booleane in modo “geometrico”: raggruppando mintermini adiacenti elimino le variabili che cambiano valore
- E’ ovviamente molto più semplice che fare le semplificazioni algebriche usando le relazioni notevoli, anche nel caso di volerlo implementare su calcolatore !
- Più i raggruppamenti sono grandi, più sto semplificando l’espressione
- E’ quindi chiaro che le mappe di Karnaugh possono essere molto utili per “minimizzare” una espressione booleana, e quindi per sintetizzare una rete logica minima

# **Vantaggi delle mappe di Karnaugh**

---

- L'ispezione della mappa di Karnaugh consente di individuare velocemente il minimo numero di mintermini (o maxtermini) per sintetizzare l'espressione booleana
- La semplificazione può avvenire attraverso l'individuazione di raggruppamenti di  $2^k$  celle purché adiacenti
- Tali raggruppamenti vengono chiamati "sottocubi"
- Ricordarsi che in una mappa di Karnaugh a quattro variabili sono sottocubi anche:
  - la prima e l'ultima colonna
  - la prima e l'ultima riga
  - le celle disposte ai vertici

# Sintesi minima con le Mappe di Karnaugh

---

L'algebra booleana ci garantisce che:

1. Ogni espressione SP non ridondante di una funzione booleana è un OR di implicant primari
2. Una espressione SP minima è una espressione non ridondante

**Per sintetizzare una rete logica minima è chiaro che conviene:**

1. Usare implicant primari essenziali, per minimizzare il numero di porte AND
2. Usare implicant primari più “grandi” possibile, in modo da minimizzare il numero di “letterali” dell'espressione, cioè di ingressi alla rete

# Algoritmo di sintesi minima con le mappe di Karnaugh

Passi fondamentali:

1. Si esprime la funzione booleana che rappresenta la rete logica da minimizzare in forma canonica SP
2. Si rappresenta la funzione con una mappa di Karnaugh
3. Nella mappa si identificano gli **implicanti primi essenziali** più grandi possibile (contenenti cioè il maggior numero possibile di “1”)
4. Si sintetizza la rete minima come SP degli implicanti primi essenziali individuati

## **Condizioni di indifferenza (don't care)**

- Nella definizione di funzioni booleane, può non interessare il valore assunto dalla funzione per una o più combinazioni delle variabili
  - perché quelle combinazioni potrebbero non essere previste (si pensi alla rappresentazione decimale a 4 bit)
- In questi casi si parla di “condizioni d’indifferenza” o “don't care” e si indicano nella tabella di verità con una ‘d’ (o con \*, o con -)
- Significa che la scelta del valore della funzione booleana per quella combinazione dipende dal progettista



# Esempio di funzione con don't care

A	B	C	Y
0	0	0	0
0	0	1	d
0	1	0	d
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	d
1	1	1	d

AB \ C	00	01	11	10
0		d	d	1
1	d	1	d	

- Possiamo imporre che nelle combinazioni don't care, se utile per la semplificazione, la funzione assuma il valore 1

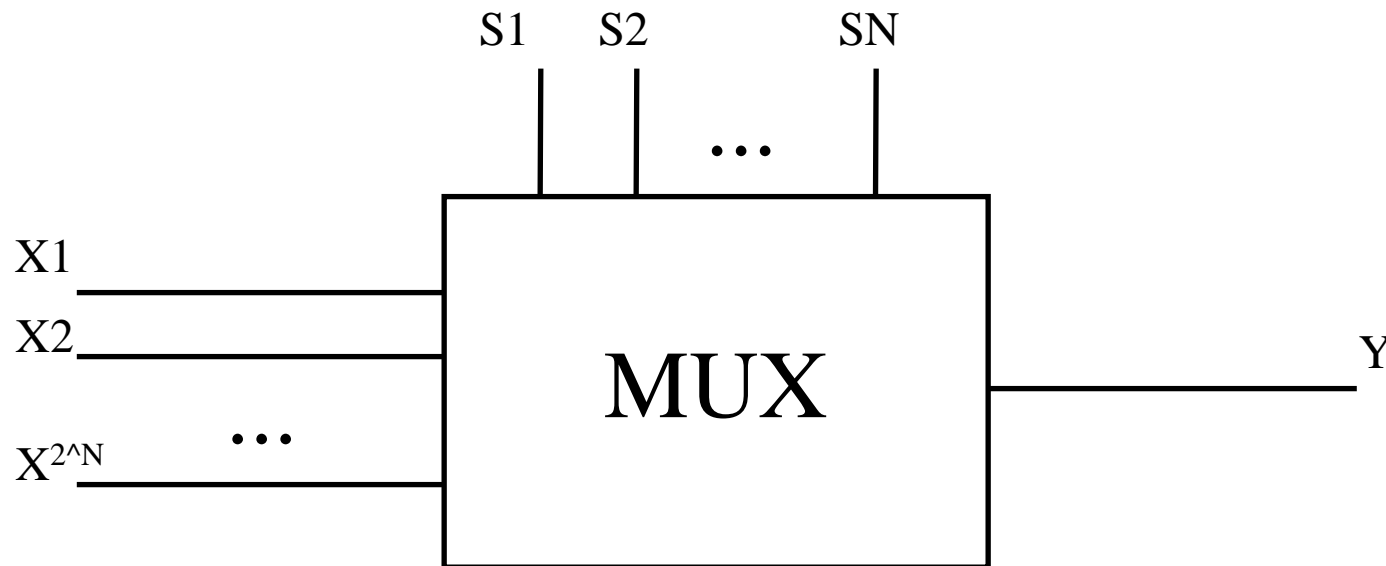
AB \ C	00	01	11	10
0		d	d	1
1	d	1	d	

# Reti combinatorie notevoli

## Il Multiplexer (MUX)

---

- Caratterizzato da  $2^N$  ingressi,  $N$  bit di controllo e un'uscita
- Ogni combinazione dei bit di controllo corrisponde a uno degli ingressi
- L'uscita assume il valore dell'ingresso associato alla combinazione dei bit di controllo



# Un semplice esempio di multiplexer

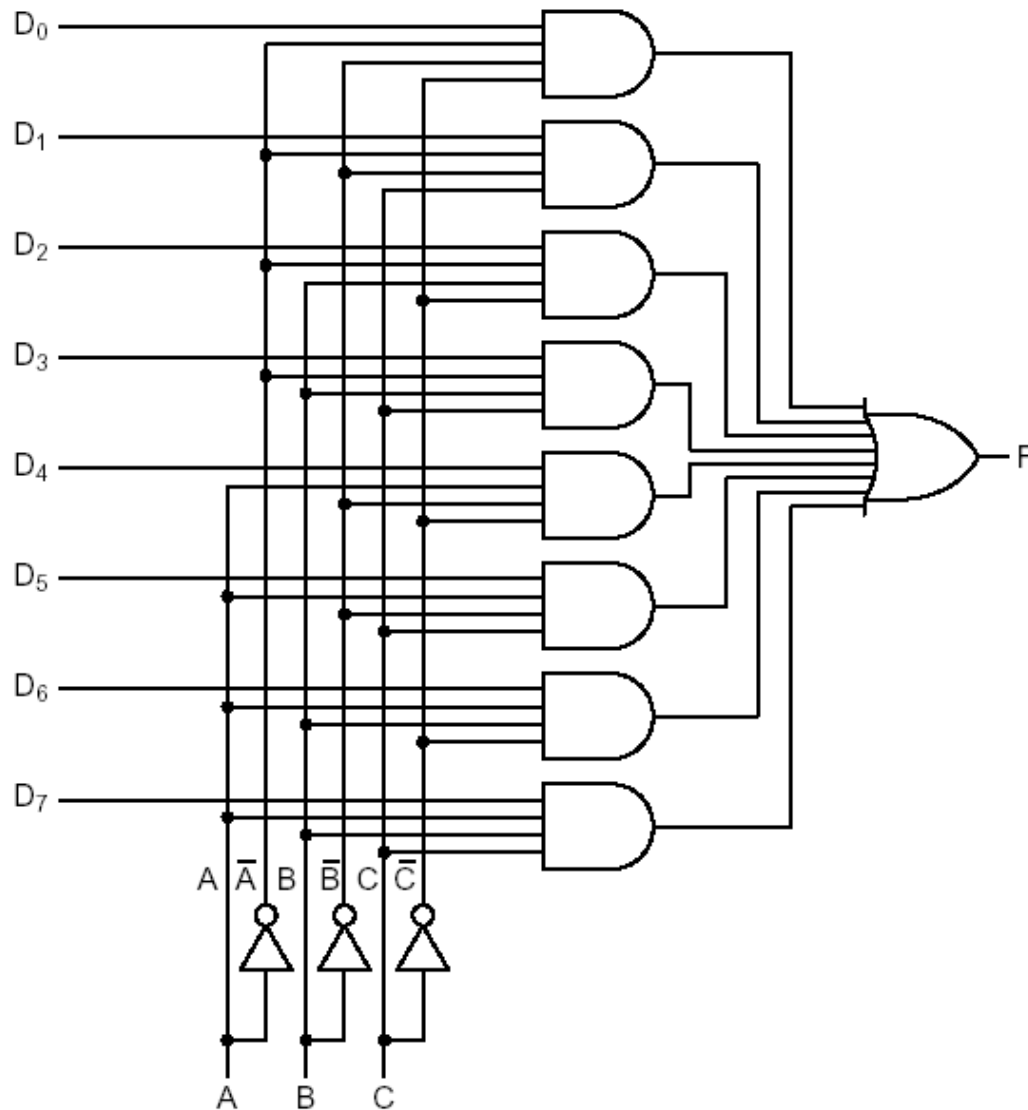
- Dati due ingressi A e B, è sufficiente un bit di controllo S per “pilotare” opportunamente l’uscita Y

S	A	B	Y
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

AB	00	01	11	10
S 0			1	1
1		1	1	

$$Y = \bar{S}A + SB$$

# Esempio di MUX a 8 ingressi



E' intuitivo che l'utilizzo pratico di un MUX riguarda situazioni del tipo:

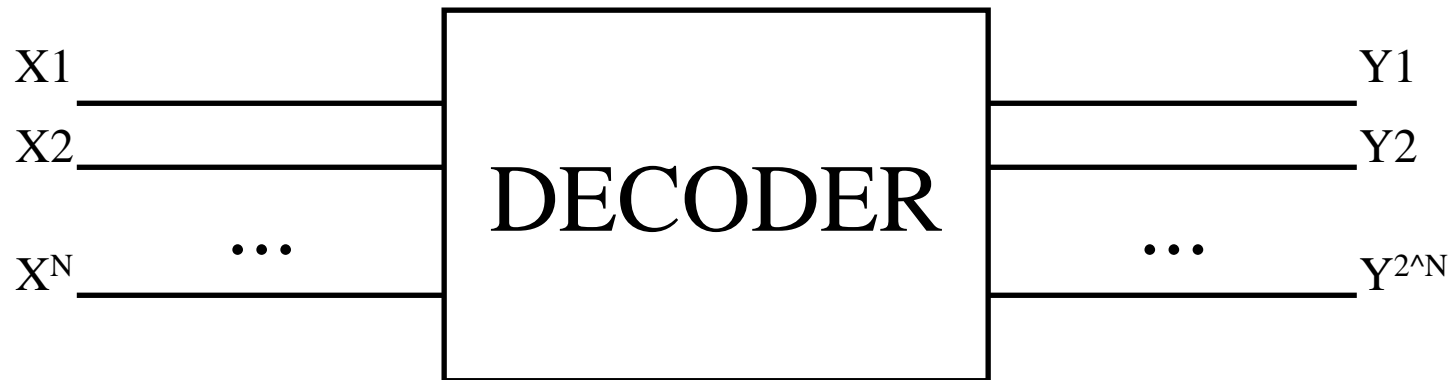
- Diverse sorgenti di dati devono “condividere” la stessa linea di trasmissione  $F$
- Devo decidere quale sorgente di dati mandare sull'uscita
- Se i dati sono dei “segnali di comando/controllo” devo decidere quale comando/controllo dare/eseguire

# Reti combinatorie notevoli

## Il Decoder N-to- $2^N$ (DEC)

---

- Caratterizzato da N ingressi e da  $2^N$  uscite
- L' $i$ -esima uscita vale 1 in corrispondenza dell' $i$ -esima combinazione degli ingressi



# Un semplice esempio di decoder

- Dati due ingressi, siamo in grado di pilotare quattro possibili uscite
- Si ottiene così il DEC 2-to-4

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

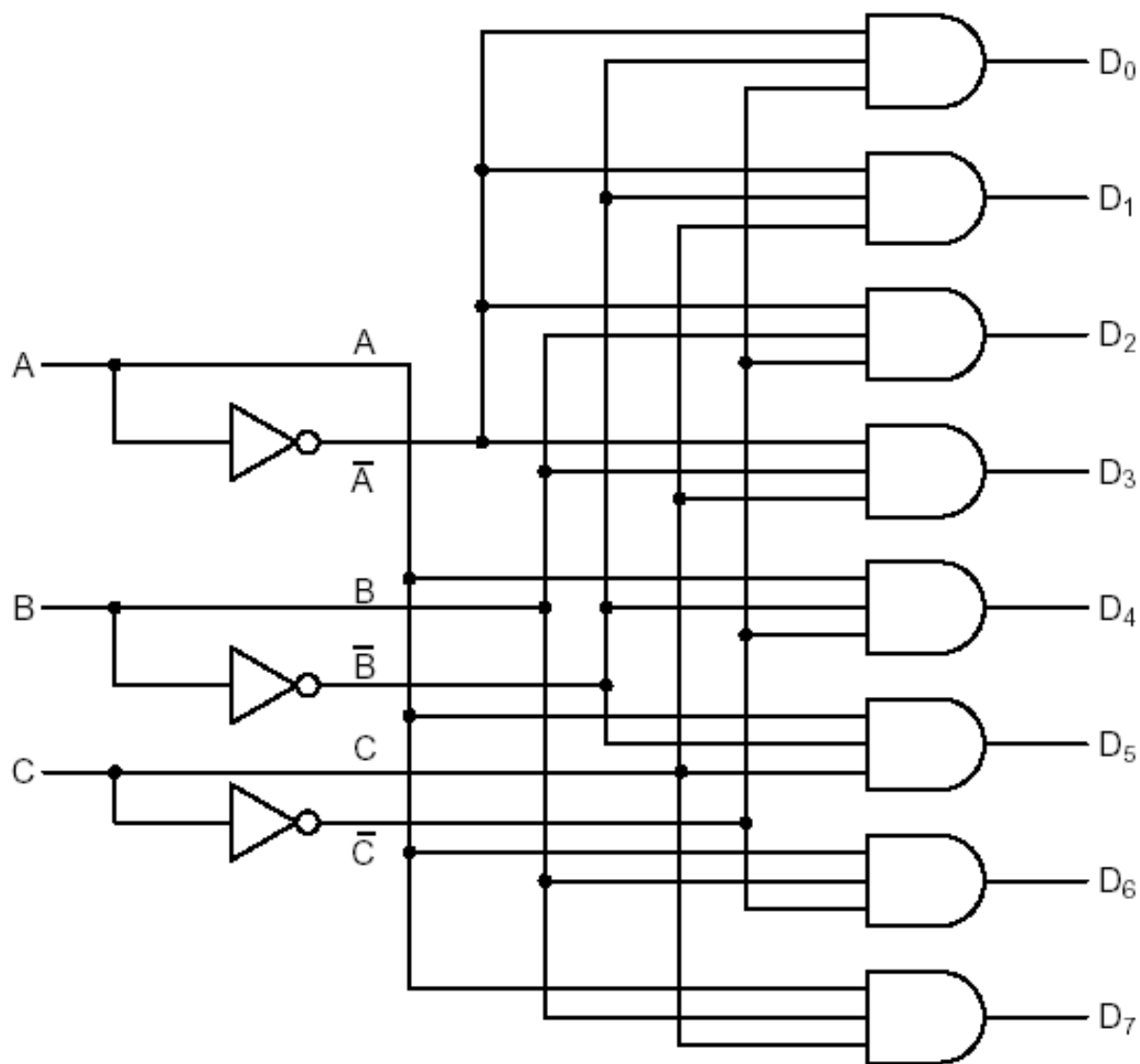
$$Y_0 = \overline{X_1} \overline{X_0}$$

$$Y_1 = \overline{X_1} X_0$$

$$Y_2 = X_1 \overline{X_0}$$

$$Y_3 = X_1 X_0$$

# Decoder 3-to-8



Nei calcolatori i decoder hanno molti utilizzi, ad esempio:

- Come “selettori”, per decidere quale dispositivo deve essere “abilitato”
- Nell’indirizzamento dei moduli di memoria
- Per connettere un dispositivo con uno fra un insieme (cioè per fare un demultiplexing, inverso del multiplexing)

# Reti sequenziali

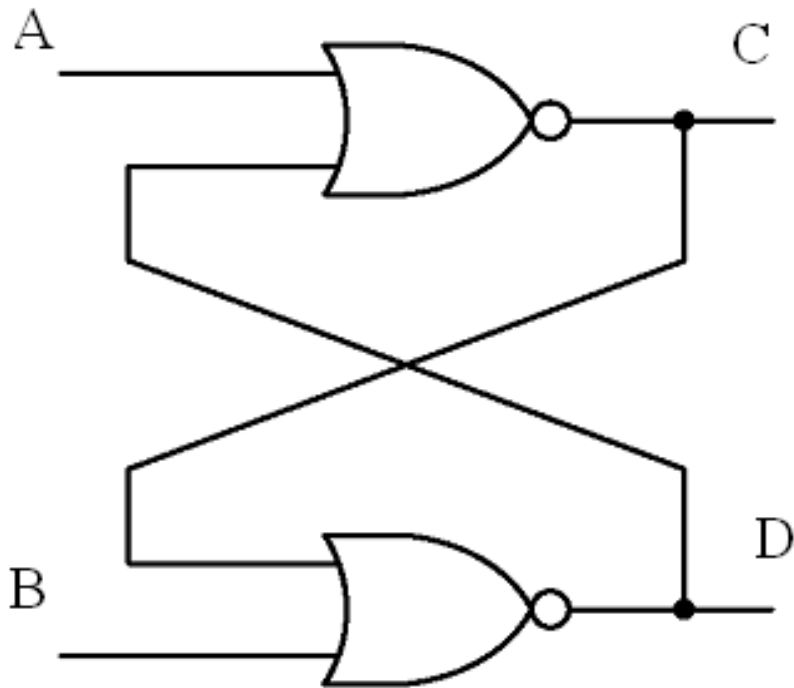
---

- Le reti sequenziali presentano  $N$  ingressi ed  $M$  uscite come le combinatorie, ma i valori delle uscite dipendono
  - dai valori degli ingressi a un dato istante
  - dai valori degli ingressi assunti negli istanti precedenti
- Per realizzare una rete sequenziale occorre sapere rappresentare la storia passata della rete, attraverso una qualche forma di “memoria”
- Essendo tale memoria limitata, una rete sequenziale è in grado di memorizzare un numero finito di possibili condizioni “passate”



# Esempio di rete sequenziale

---



- Quella a lato è una semplicissima rete sequenziale
- E' sequenziale perché, come è facile capire, le uscite dipendono, oltre che dagli ingressi, anche dalle uscite assunte in precedenza
- Come è ovvio essendoci una “retroazione”

•Vedremo fra breve che questa rete sequenziale altro non è che il più semplice elemento di memoria (un “latch”)

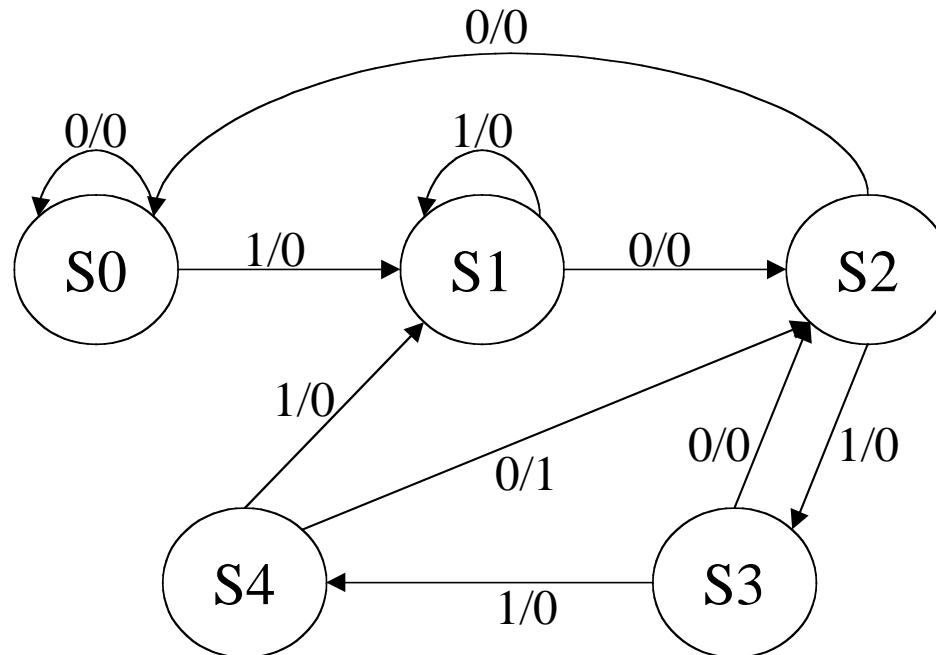
# Definizione di una rete sequenziale

---

- Una rete sequenziale è definita da:
  - l'insieme degli stati di ingresso: insieme di configurazioni delle variabili booleane di ingresso
  - l'insieme degli stati interni: ogni stato corrisponde a una possibile configurazione "passata" della rete
  - l'insieme degli stati di uscita: insieme di configurazioni delle variabili booleane di uscita
  - il diagramma degli stati: descrive, in funzione delle variabili di ingresso e degli stati interni ad un dato istante, lo stato all'istante successivo e le relative configurazioni degli stati di uscita
- La definizione appena data è del tutto coincidente con quella di "automa a stati finiti"

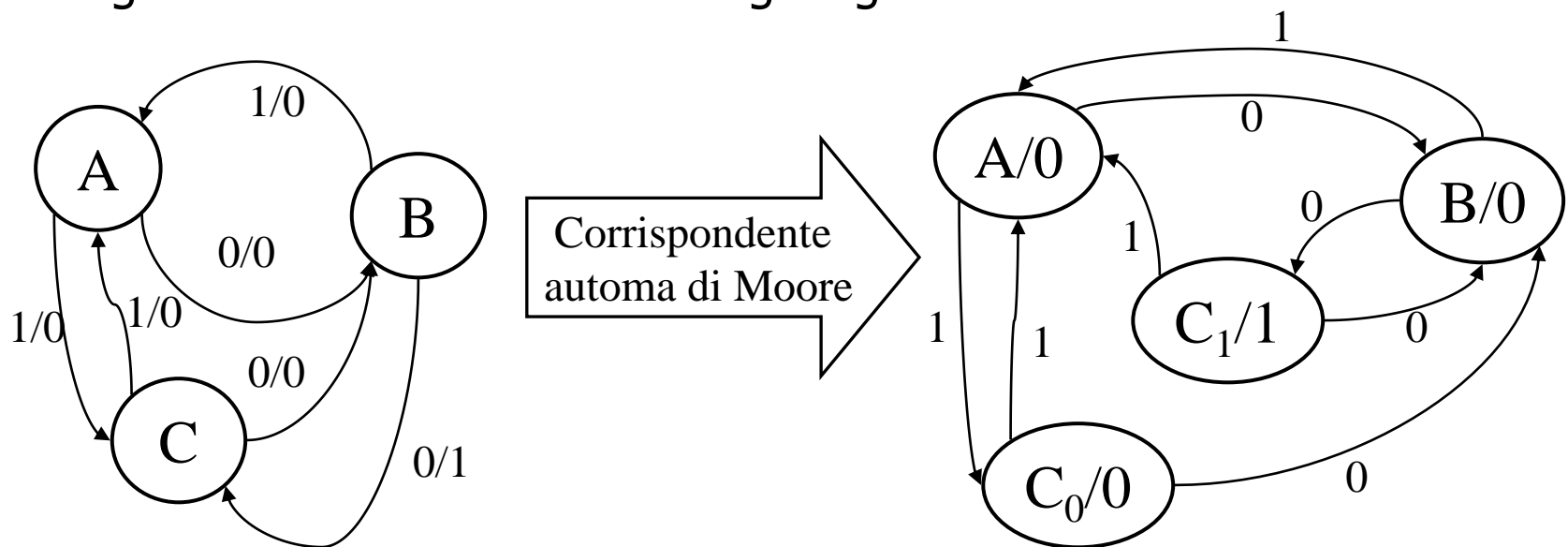
# Il diagramma degli stati

- Il diagramma degli stati è un grafo orientato:
  - i nodi sono etichettati con uno dei possibili stati
  - gli archi sono diretti dallo stato attuale a quello successivo e sono etichettati con l'ingresso presente e la corrispondente uscita



# Diagramma di Mealy e di Moore

- Il diagramma degli stati precedente si riferisce a un modello di macchina sequenziale detto di "Mealy"
- Un'alternativa è quello di "Moore", che differisce dal precedente in quanto:
  - i nodi sono etichettati con lo stato e con l'uscita in corrispondenza di detto stato
  - gli archi sono etichettati con gli ingressi



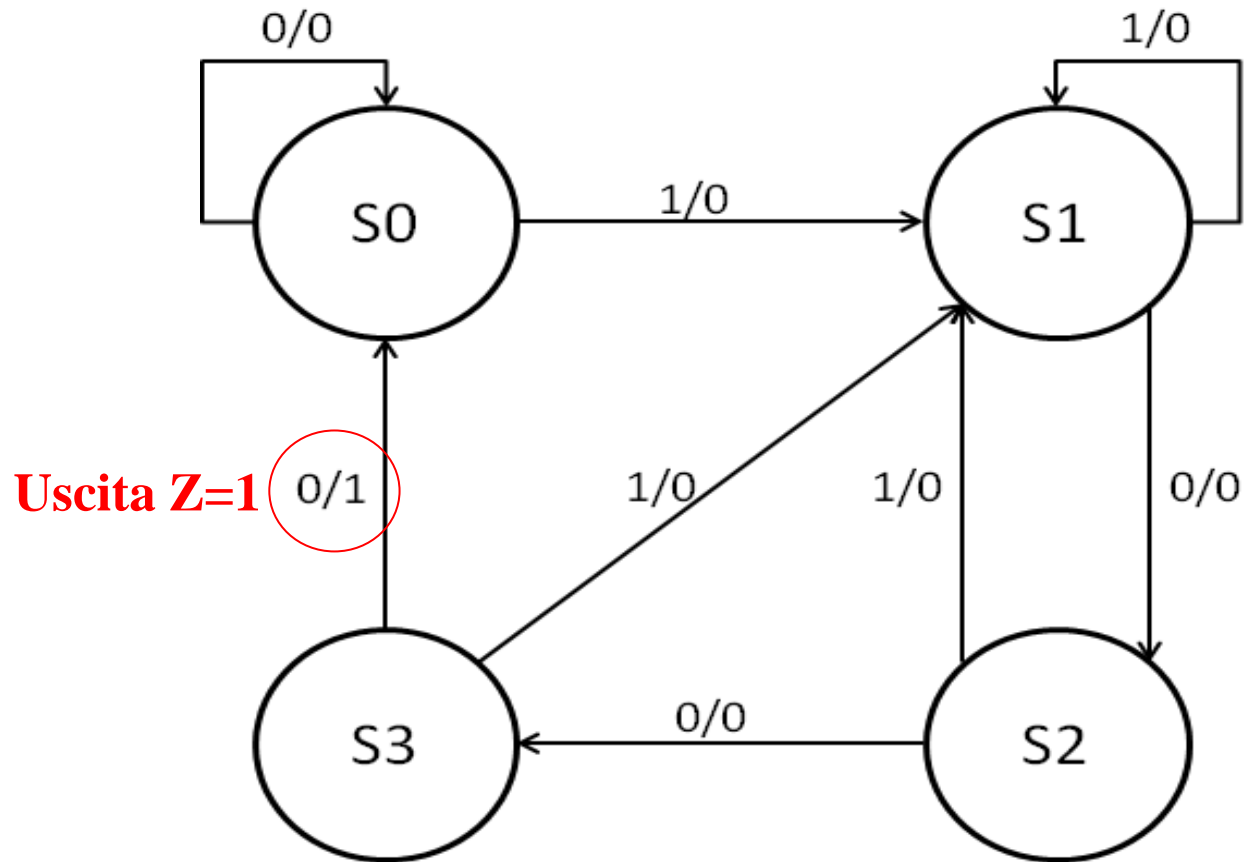
# Esempio di diagramma degli stati

Si voglia progettare una rete logica in grado di riconoscere in una sequenza di bit in ingresso la stringa **1000** ponendo a 1 l'uscita Z solo quando si abbia il riconoscimento di tale stringa.



- Vediamo il diagramma degli stati di questa rete logica nella pagina seguente

# Esempio di diagramma degli stati



# La tabella di flusso

---

- Relativa alle transizioni {ingresso/stato presente}  $\rightarrow$  {stato successivo/uscita}
- E' l'equivalente tabellare del diagramma degli stati

Stato Presente	Stato Successivo/Uscita	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S3/0	S1/0
S3	S0/1	S1/0

X è l'ingresso

# Implementazione dello stato di una rete sequenziale

---

- Ogni stato è rappresentato in una rete sequenziale da una configurazione di un insieme di variabili booleane dette appunto di “stato”
- Il numero delle variabili (bit) le cui configurazioni rappresentano ciascuno stato è dato dal minimo intero  $N$  per cui, se  $S$  è il numero degli stati:

$$2^N \geq S$$

- Es. quattro stati  $\{S_0, S_1, S_2, S_3\}$  possono essere rappresentati dalle configurazioni di due bit:  $S_0=00, S_1=01, S_2=10, S_3=11$
- E' chiaro che per tenere traccia dello stato serve un “dispositivo” in grado di memorizzare uno o più bit. Vedremo a breve questi **elementi di memoria (latch e flip flop)**



# La tabella delle transizioni

- Sostituendo nella tabella di flusso la notazione simbolica degli stati con le configurazioni dei bit che li rappresentano, si ottiene la tabella delle transizioni

Tabella delle transizioni

relativa all'esempio  
precedente

<b>Stato Presente</b>	<b>Stato Successivo/Uscita</b>	
	<b>X=0</b>	<b>X=1</b>
00	00/0	01/0
01	10/0	01/0
10	11/0	01/0
11	00/1	01/0

# **Le funzioni di una rete sequenziale**

---

- La tabella delle transizioni non è altro che la tabella di verità di una rete sequenziale che determina:
  - in funzione dei bit dello stato attuale e dell'ingresso, il valore di ciascun bit dello stato successivo
  - in funzione dei bit dello stato attuale e dell'ingresso, il valore dell'uscita
- Le due funzioni sopra descritte prendono il nome di:
  - funzione di transizione dello stato
  - funzione di uscita

# Le funzioni di una rete sequenziale

- Funzione di transizione dello stato  $F$ 
  - dato l'input e lo stato ad un dato istante, calcola lo stato all'istante successivo
  - lo stato non è "visibile" all'esterno

$$S' = F(X, S)$$

- Funzione di uscita  $G$ 
  - dato l'input e lo stato a un dato istante, calcola l'uscita
  - l'uscita è "visibile" all'esterno

$$Y = G(X, S)$$

$X, S, Y$  sono  
rispettivamente  
i vettori delle  
variabili di  
ingresso, di  
stato e di uscita

➤ *Le funzioni  $F$  e  $G$  vengono implementate attraverso reti combinatorie*

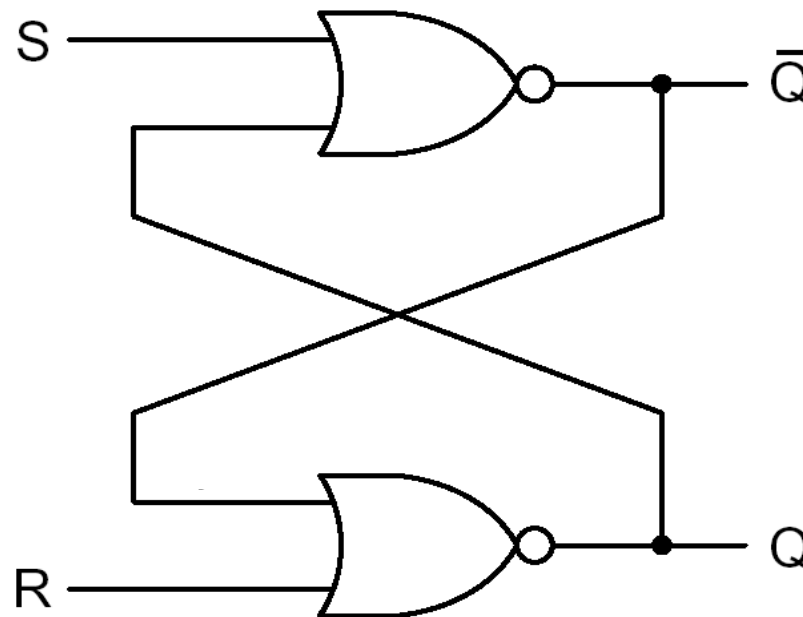
# Elementi di memoria

---

- Come visto lo stato di una rete sequenziale è rappresentato da un certo numero di bit.
- Vediamo ora dei semplici elementi di memoria, ognuno dei quali può memorizzare un bit
- L'unione di più elementi di memoria da 1 bit consente di memorizzare i possibili stati assunti da una rete sequenziale
- Gli elementi di memoria si dividono in due categorie
  - elementi di memoria asincroni e sincroni (latch)
  - elementi di memoria "master-slave" (flip flop)

## **Il latch Set-Reset asincrono (SR)**

- Il principio base dei latch è la retroazione del segnale di uscita sull'ingresso:



$$Q(t + t) = \overline{(R + \overline{Q(t)})} = \overline{(R + \overline{(S + Q(t))})} = (S + Q(t)) \times \overline{R}$$

# Il latch SR asincrono

- Dall'equazione precedente si ricava la tabella di verità

- In sintesi:

— se  $S = 1$ ,  $Q(t+\tau) = 1$ ;

—  $S$  = Set signal

— se  $R = 1$ ,  $Q(t+\tau) = 0$ ;

—  $R$  = Reset signal

— se  $S = R = 0$ ,  $Q(t+\tau) = Q(t)$

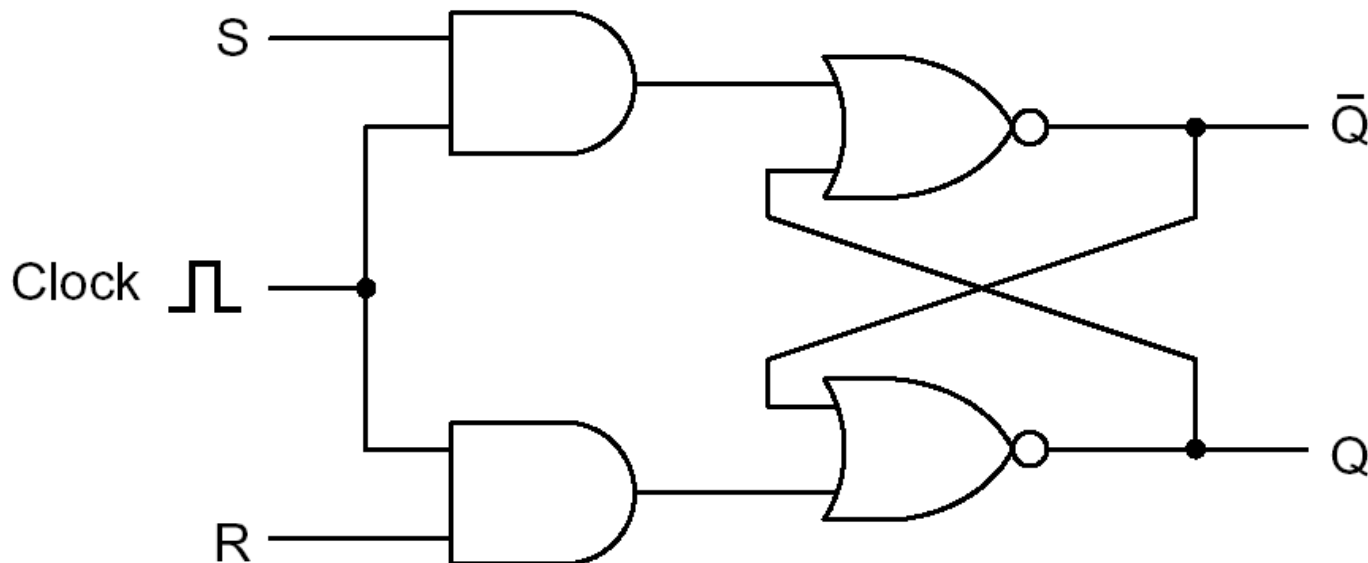
— stato invariato

—  $S = R = 1$  : configurazione non ammessa

S	R	Q(t)	Q(t+ $\tau$ )
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	-

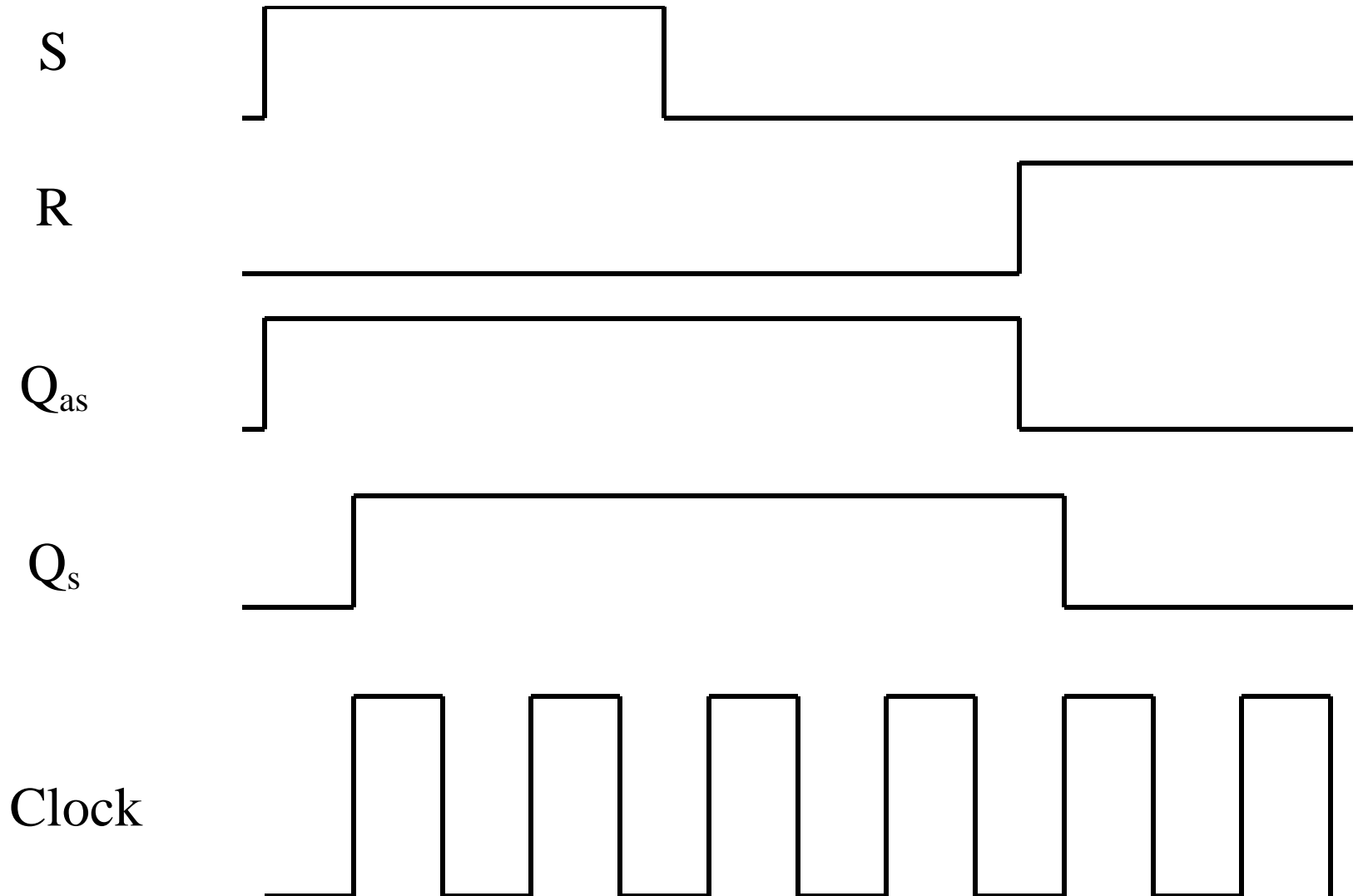
# Il latch SR sincrono

- Il latch appena presentato è un dispositivo "asincrono"
  - non c'è alcun modo per "sincronizzare" la lettura degli ingressi con le uscite
  - i valori delle uscite cambiano in funzione del livello dei segnali S e R istante per istante
- Connettendo il latch SR a due porte AND ed a un segnale di sincronismo si ottiene il latch SR "sincrono", in modo che le uscite cambino solo in presenza di tale segnale:



# Differenza fra latch SR asincrono e sincrono

---





# Latch JK sincrono

- E' un latch SR in cui viene ammessa la configurazione  $J=K=1$ , per cui il valore del bit di uscita viene invertito

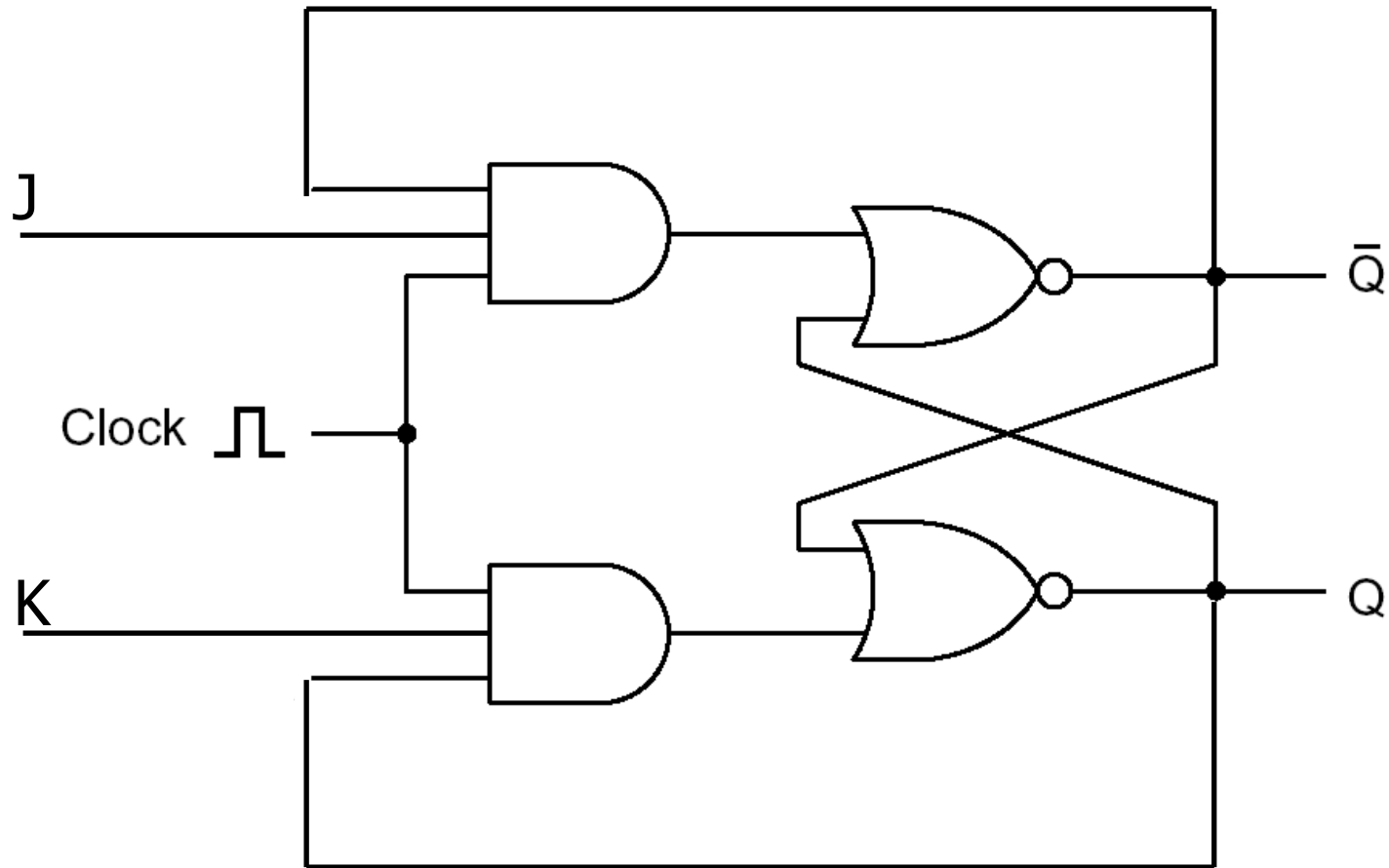
J	K	Q(t)	Q(t+ $\tau$ )
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	<b>0</b>	<b>1</b>
1	1	<b>1</b>	<b>0</b>

Inversione dell'uscita

Tabella di verità compatta

Q(t)	Q(t+ $\tau$ )	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

# Circuito logico del latch JK sincrono



# Altri latch sincroni

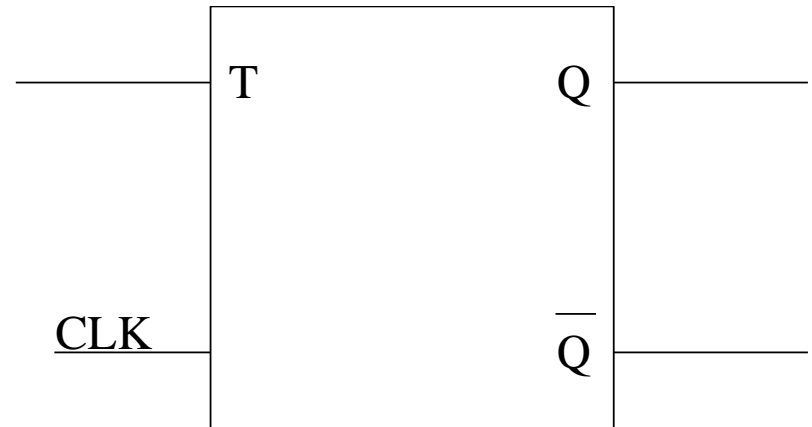
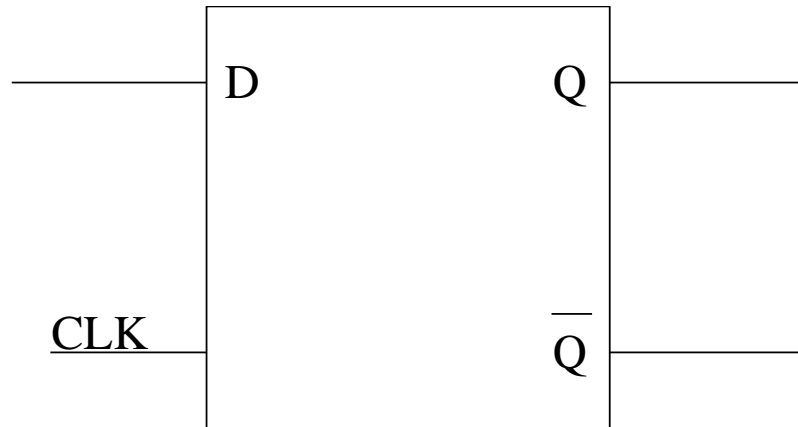
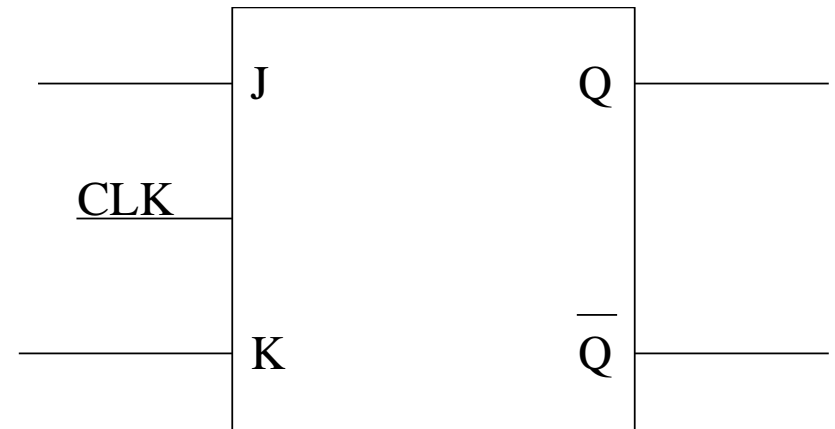
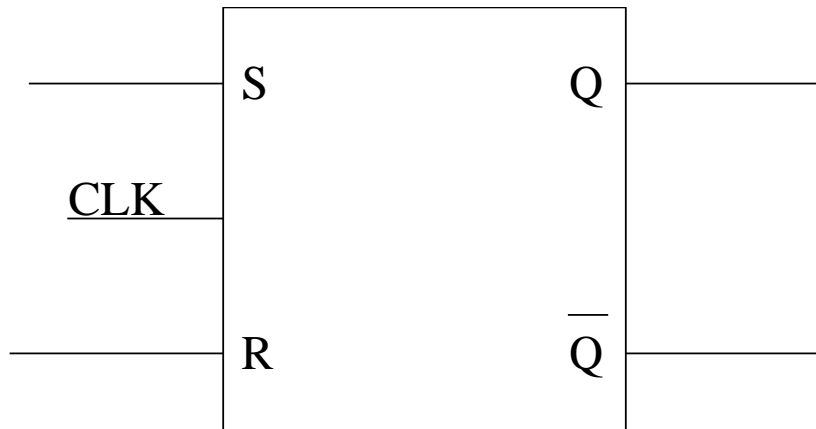
- Latch D (Delay)
  - latch JK per cui  $\mathbf{J} = \overline{\mathbf{K}} = \mathbf{D}$
  - l'effetto è di "ritardare" di un ciclo di clock il valore dell'ingresso D sull'uscita Q
- Latch T (Trigger)
  - latch JK per cui  $\mathbf{J} = \mathbf{K} = \mathbf{T}$
  - l'effetto è di invertire il valore dell'uscita Q se l'ingresso T vale 1

D	Q(t)	Q(t+τ)
0	0	0
0	1	0
1	0	1
1	1	1

T	Q(t)	Q(t+τ)
0	0	0
0	1	1
1	0	1
1	1	0

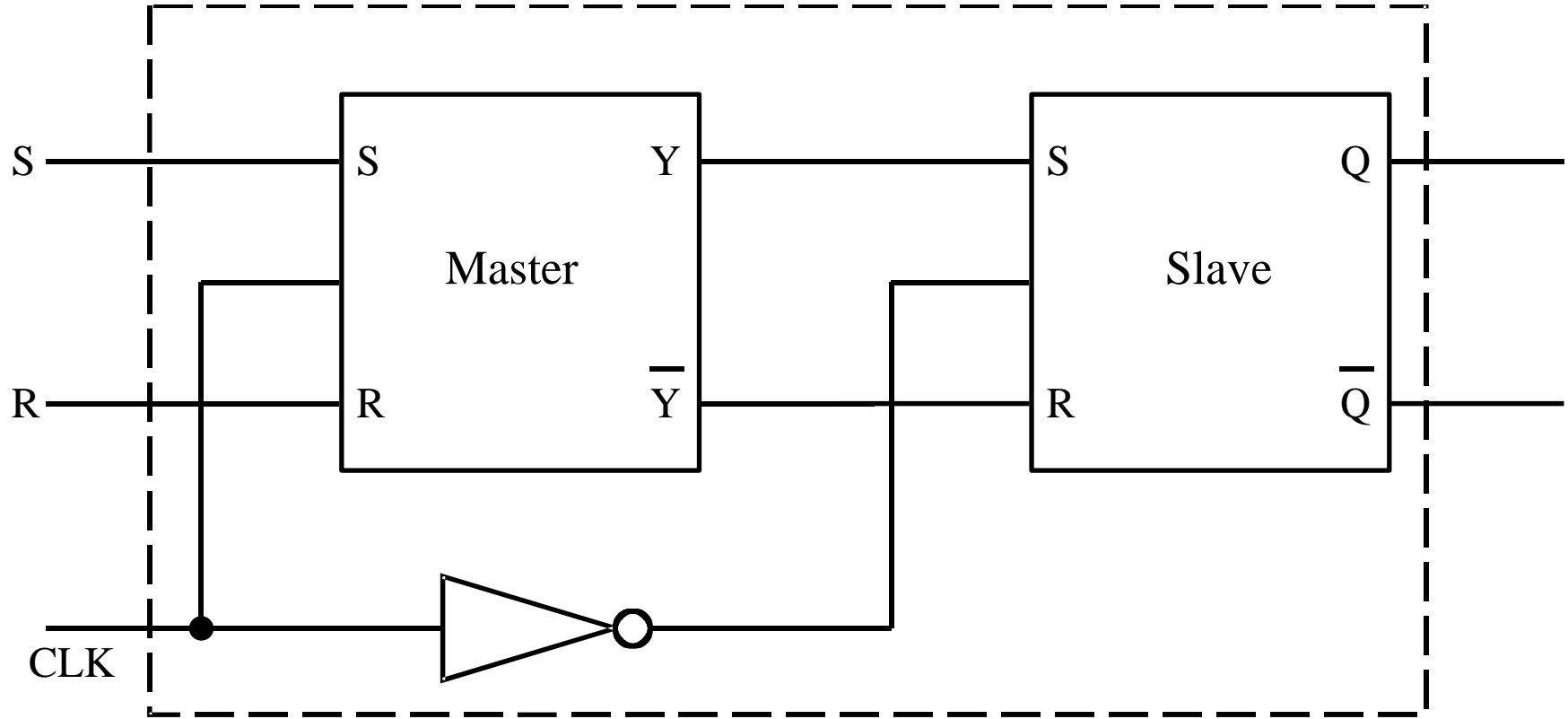
# Rappresentazione grafica dei latch

---



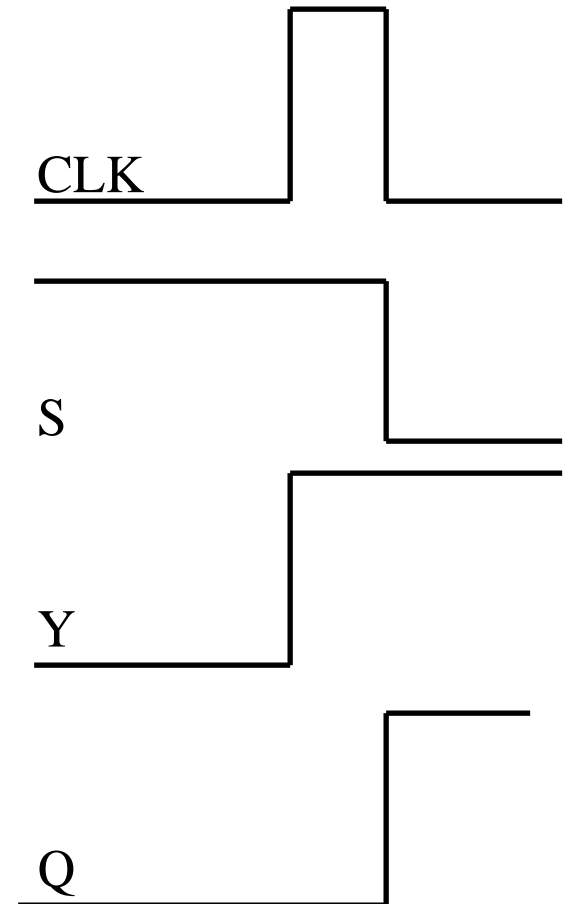
# Elementi di memoria “master-slave”

- Chiamati “flip flop”
- Formati da un inverter (porta NOT) e due latch, uno chiamato “master”, l’altro “slave”

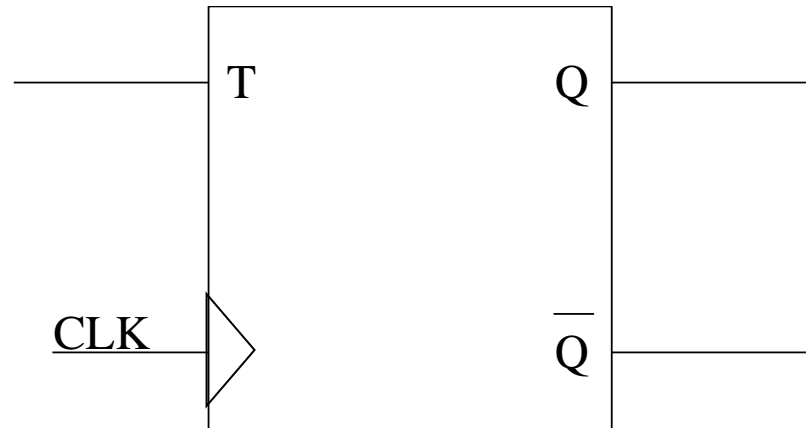
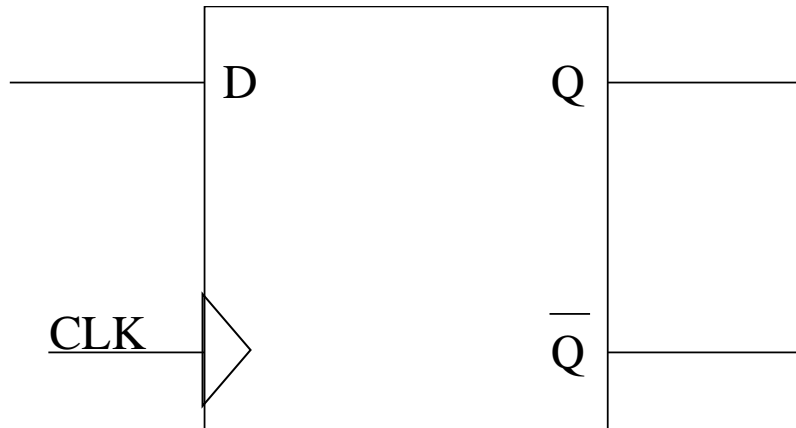
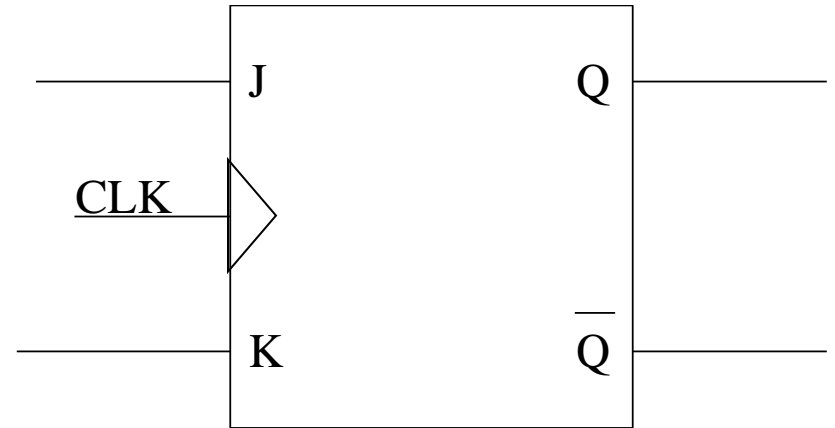
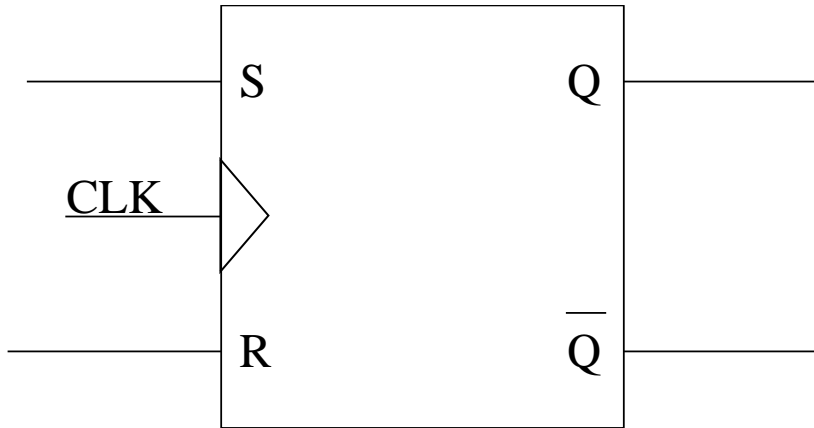


# Disaccoppiamento tra caricamento dati e rilascio dei dati

- I flip flop consentono il disaccoppiamento tra la funzione di caricamento dei dati, realizzata dal master, e rilascio dei dati, realizzata dallo slave
- Il flip flop master immagazzina i dati durante la commutazione 0-1 del clock ("fronte di salita") ma lo slave è disabilitato
- Quando il clock va a 0, il master viene disabilitato (variazioni dei segnali in ingresso non si ripercuotono su Y) mentre i dati precedentemente immagazzinati passano allo slave che li mette a disposizione sull'uscita Q

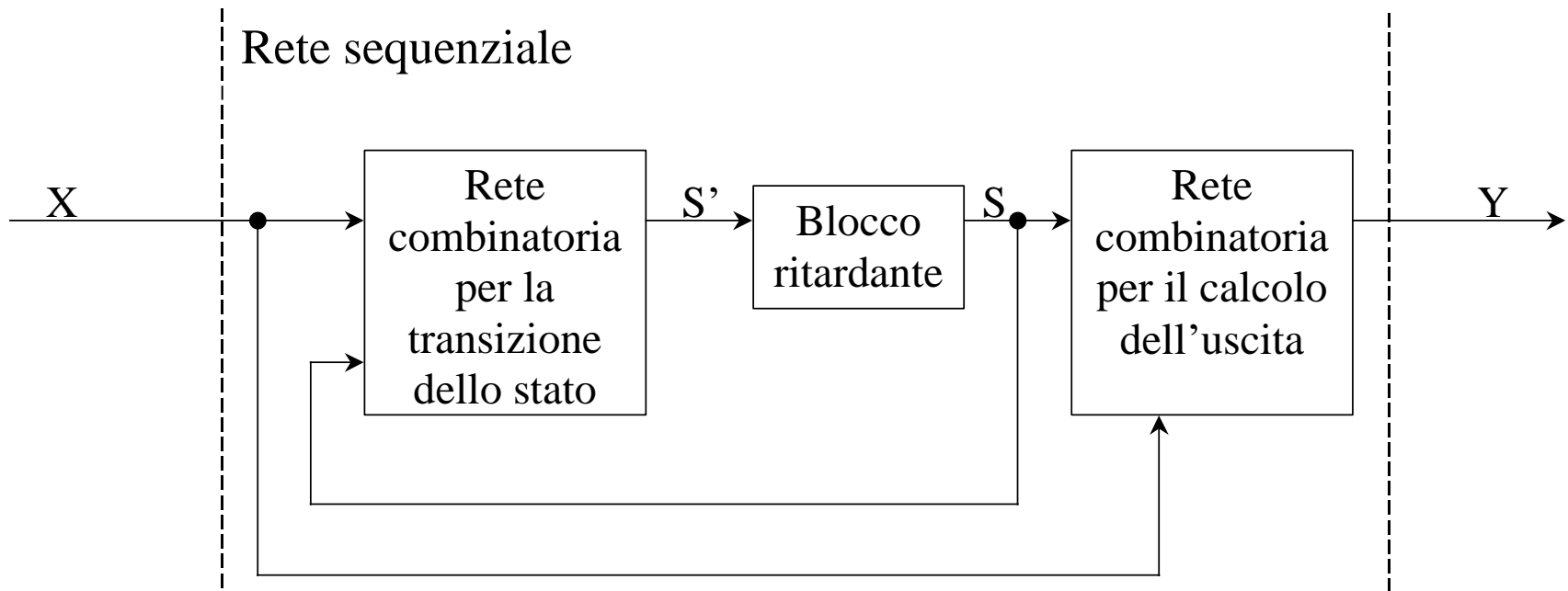


# Rappresentazione grafica dei flip flop



# Architettura di una rete sequenziale

- $X$ ,  $S$ ,  $Y$  sono rispettivamente i vettori delle variabili di ingresso, di stato e di uscita
- Il blocco ritardante ha in ingresso lo stato successivo  $S'$  e in uscita lo stato attuale  $S$





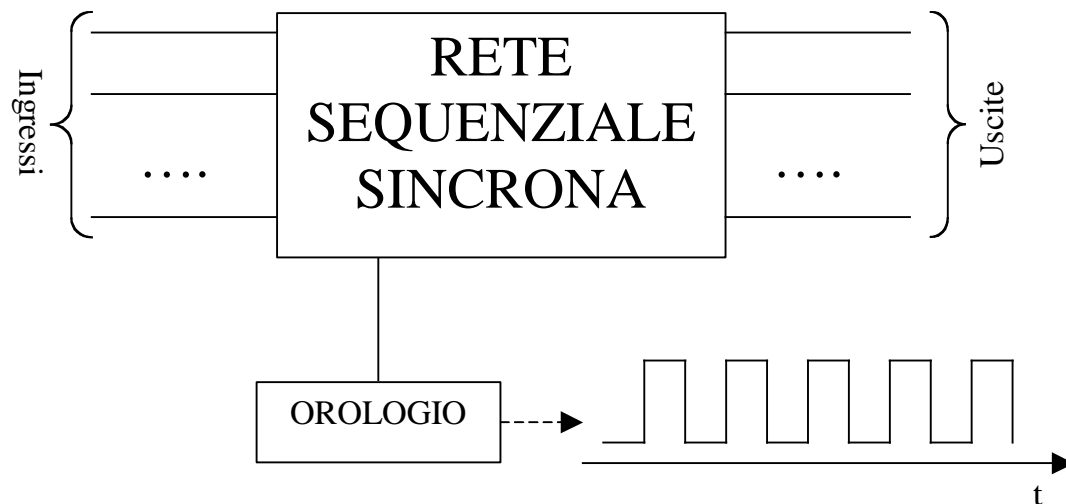
# Tipi di rete sequenziale

---

- Reti sequenziali asincrone
  - prive del “blocco ritardante”
  - i segnali dello stato interno e delle uscite evolvono fino al raggiungimento di un assetto “stabile”
  - di difficile progettazione, si usano per la comunicazione fra circuiti sincronizzati indipendentemente l’uno dall’altro
- Reti sequenziali sincrone
  - i segnali sono letti a intervalli di tempo regolari
  - i segnali dello stato interno evolvono ad ogni intervallo
  - il blocco ritardante è costituito da elementi di memoria sincronizzati con la lettura dei segnali di ingresso

# Reti sequenziali sincrone

- Una rete sequenziale sincrona “interroga” gli ingressi a istanti definiti a priori ed equidistanziati
- Tali istanti sono scanditi da un “orologio” (clock), generatore di impulsi di sincronismo
- Ad ogni impulso la rete interroga gli ingressi ed emette l’uscita in funzione di essi e della configurazione delle variabili di stato



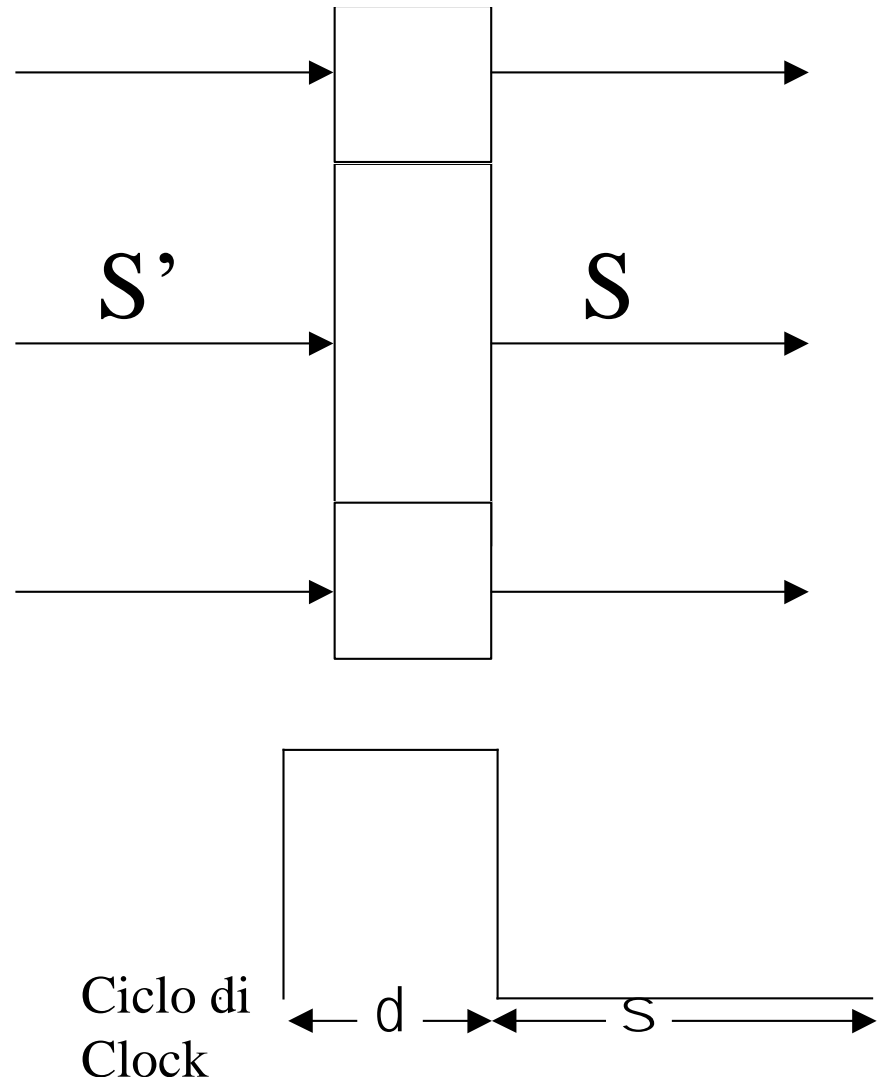
# Il “blocco ritardante”

---

- Destinato a memorizzare gli stati della rete
- E' implementato da un “registro”
- Il registro è formato da un certo numero di elementi di memoria chiamati “flip flop”
- Il registro è il cuore della rete sequenziale sincrona: grazie ad esso ogni impulso di sincronismo permette l'elaborazione dei dati presenti in ingresso, in funzione della storia passata della rete
- Il contenuto del registro è infatti uno degli  $S$  stati consentiti, ed è rappresentato da una configurazione degli  $N$  bit necessari per rappresentarlo

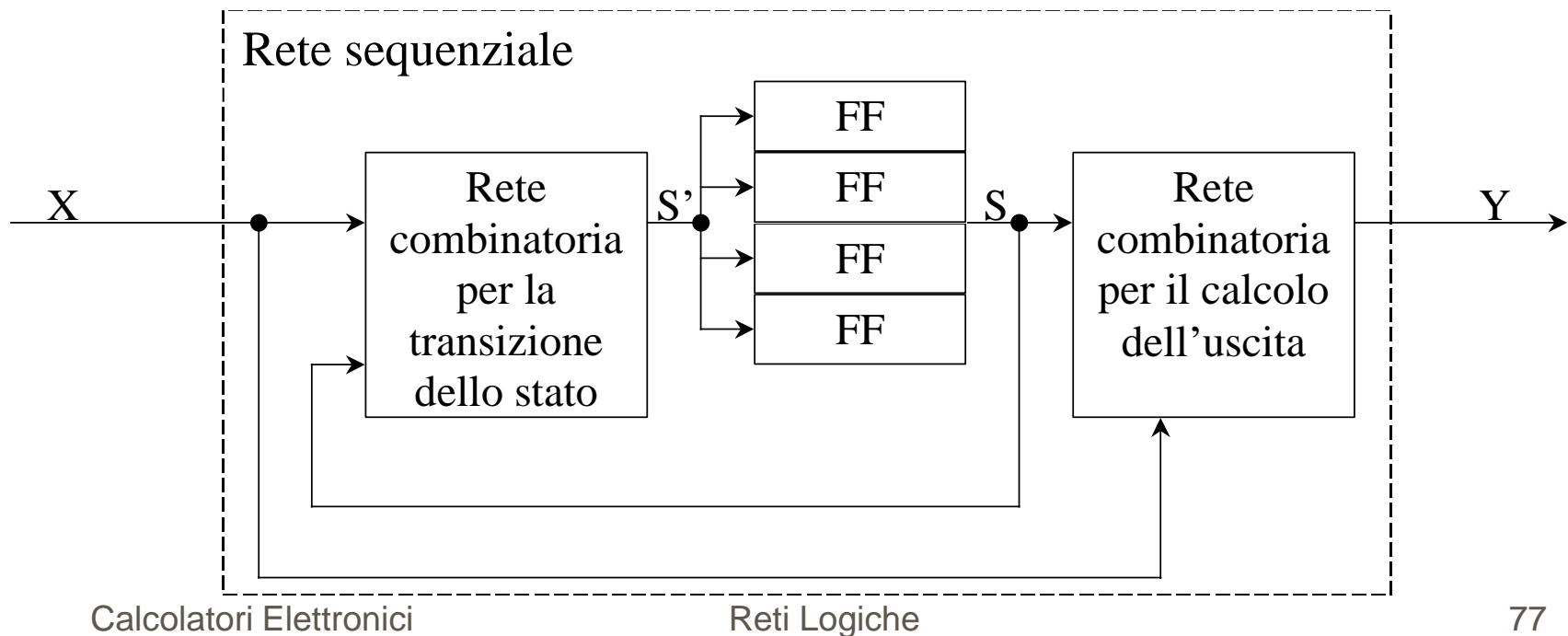
# Disaccoppiamento fra caricamento dello stato e calcolo dello stato successivo

- Periodo  $\delta$ : lo stato viene caricato nel registro
- Periodo  $\sigma$ : lo stato viene messo a disposizione, insieme agli ingressi, per la transizione ed il calcolo dell'uscita
- E' evidente che gli ingressi possono essere alterati solo durante il periodo di assenza del segnale di sincronismo



# Implementazione del blocco ritardante

- Poiché i flip flop implementano la proprietà di “disaccoppiare” il caricamento del dato dal rilascio dello stesso, possiamo utilizzarli per memorizzare ogni bit dello stato della rete sequenziale sincrona, ovvero il “blocco ritardante”
- Tale insieme di flip flop è il cosiddetto “registro”



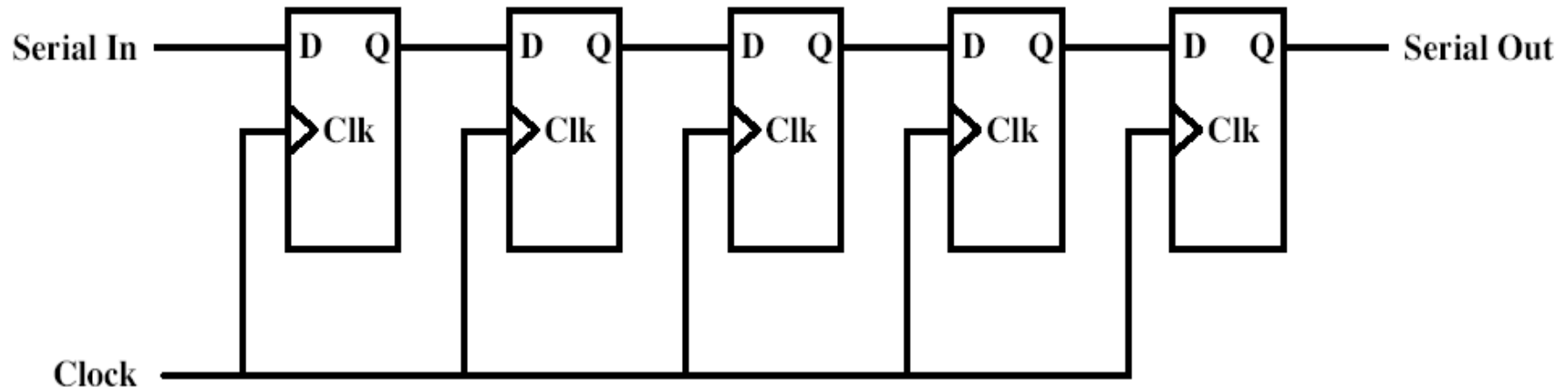
# **Vantaggi del disaccoppiamento**

---

- Realizza l'effetto "ritardante", in quanto, mentre viene presentato in ingresso lo stato successivo, viene contemporaneamente letto in uscita lo stato precedente
- Il nuovo stato viene memorizzato soltanto nel periodo in cui i segnali si sono stabilizzati
- Non ci possono essere variazioni dello stato indesiderate, in quanto il funzionamento sincrono forza una sola variazione di stato per ogni impulso

# **Esempio di registro con flip flop D: il registro “a scorrimento” (shift-register)**

- Ad ogni impulso di clock, vi è lo scorrimento di un bit da sinistra verso destra



# Analisi e sintesi di una rete sequenziale sincrona

---

- Analisi
  - dall'esame delle porte logiche e dei flip flop che compongono la rete, capire qual è la funzione implementata in termini di tabella delle transizioni e diagramma degli stati
- Sintesi
  - dall'analisi dei requisiti, ovvero dalle corrispondenze sequenza di ingresso-sequenza di uscita, implementare la funzione di transizione dello stato e la funzione di uscita utilizzando un opportuno numero di flip flop e di porte logiche



# Esempio di sintesi di una rete sequenziale sincrona

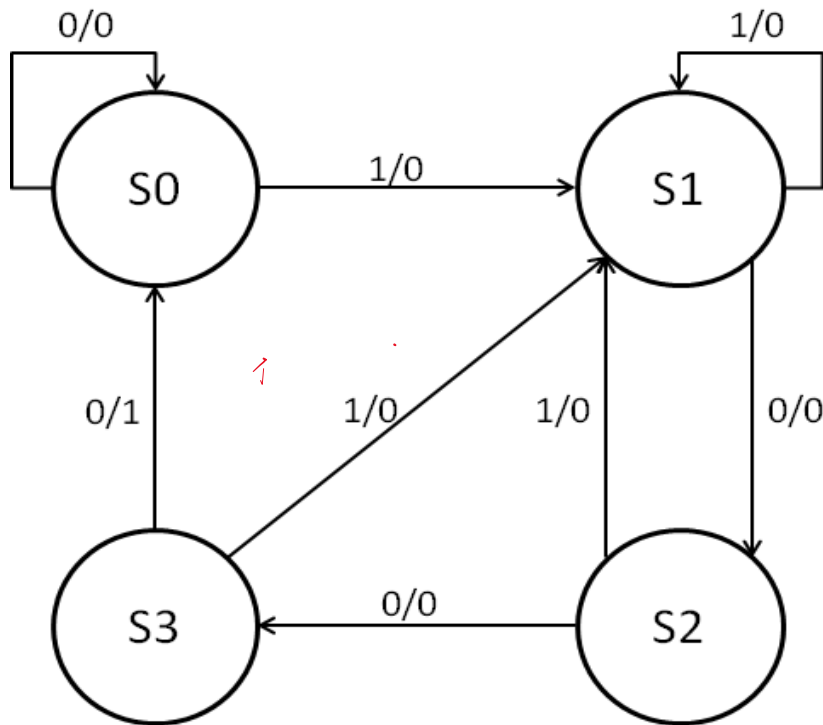
- Si voglia progettare una rete logica in grado di riconoscere in una sequenza di bit in ingresso la stringa 1000 ponendo a 1 l'uscita Z solo quando si abbia il riconoscimento di tale stringa.



- Si richiede:
  - il diagramma degli stati;
  - la tabella delle transizioni mediante l'uso di flip flop D;
  - La sintesi della rete logica per la transizione di stato e per il calcolo dell'uscita usando le mappe di Karnaugh.

# Esempio di sintesi di una rete sequenziale sincrona

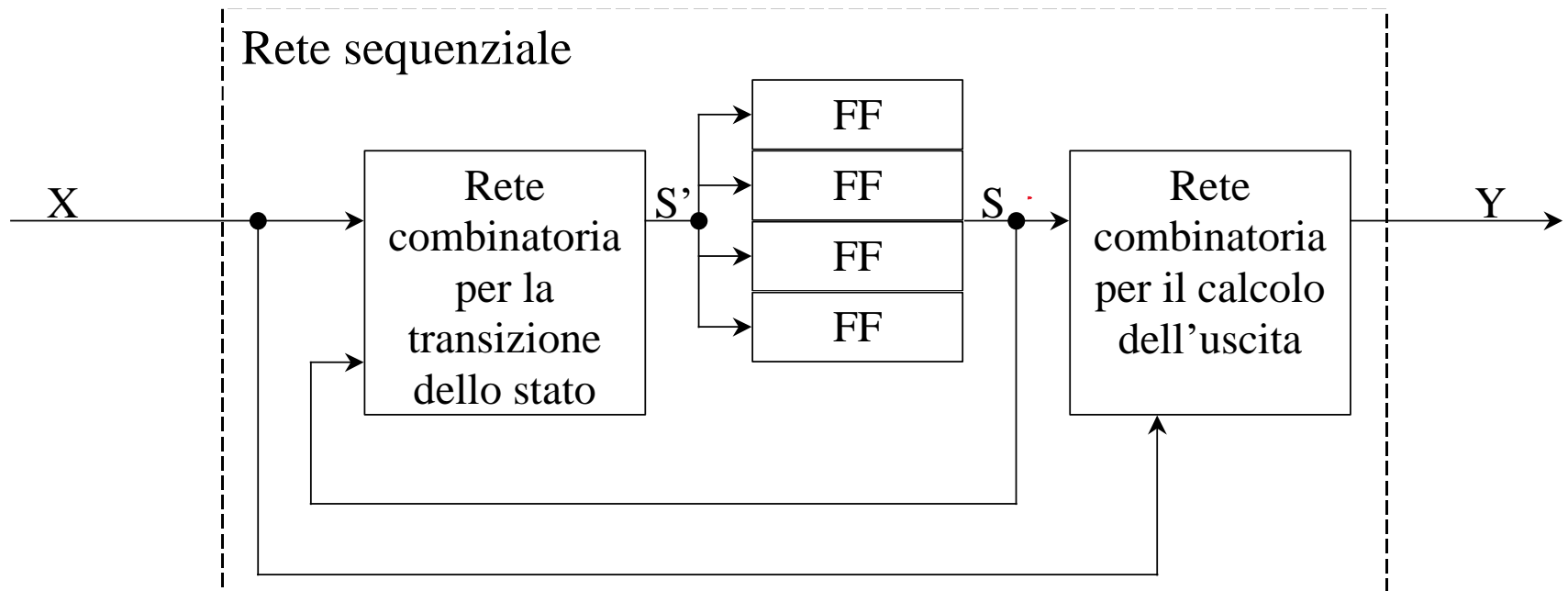
(a) grafo degli stati



(b) tabella delle transizioni

Stato Presente	Stato Successivo/Uscita	
	X=0	X=1
00	00/0	01/0
01	10/0	01/0
10	11/0	01/0
11	00/1	01/0

# Schema generale rete sequenziale



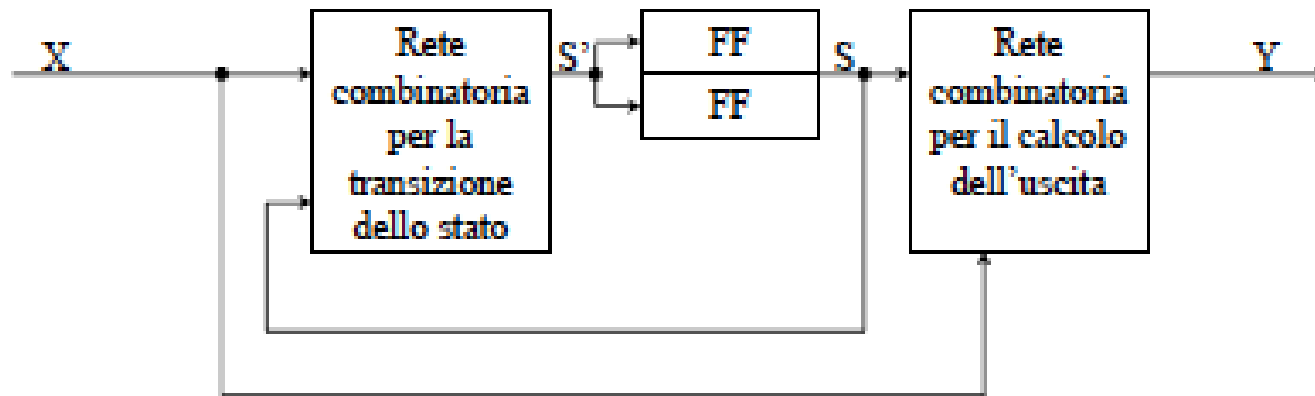
- Nel nostro caso abbiamo **4 stati**, quindi servono **2 flip-flop D**
- Ora dobbiamo progettare la «rete combinatoria per la transizione dello stato», la rete che «piloterà» le transizioni di stato dei flip-flop in accordo con il diagramma degli stati precedente

# Rete combinatoria per transizione stato

Funzione di transizione dello stato F

- dato l'input e lo stato S ad un dato istante, calcola lo stato S' all'istante successivo

$$S' = F(X, S)$$



Per progettare la rete per transizione di stato deve tenere in conto il funzionamento del flip-flop che voglio usare (D in questo caso)

# Rete combinatoria per transizione stato

Funzionamento

Flip flop D

D	Q(t)	Q(t+ $\tau$ )
0	0	0
0	1	0
1	0	1
1	1	1

La rete combinatoria per la transizione dello stato deve fornire ai 2 flip-flop D gli ingressi opportuni che «pilotino» i flip-flop a compiere le transizioni di stato definite dalla tabella qui sotto

Stato Successivo/Uscita		
Stato Presente	X=0	X=1
00	00/0	01/0
01	10/0	01/0
10	11/0	01/0
11	00/1	01/0

# Tabella «pilotaggio» flip-flop

- A e B rappresentano lo stato attuale memorizzato in ciascuno dei 2 flip-flop, A' e B' lo stato successivo in cui voglio andare in funzione dell'ingresso X e dello stato

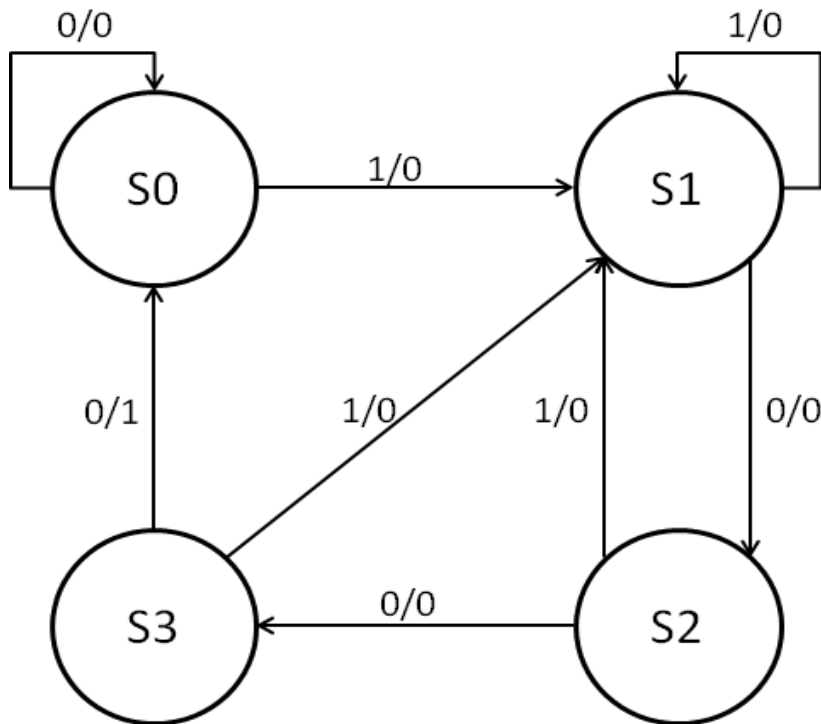
$$A' = F_A(X, A), B' = F_B(X, B)$$

- La tabella di verità qui sotto descrive le due funzioni che voglio realizzare
- DA e DB sono gli ingressi che devo fornire ai 2 flip-flop per fargli fare le transizioni di stato desiderate

A	B	X	A'	DA	B'	DB
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

# Da grafo stati a tabella pilotaggio flip-flop

- E' semplice trovare la corrispondenza fra nodi del grafo (diagramma degli stati) e le righe della tabella di pilotaggio dei flip-flop
- Nel caso del flip-flop D è anche molto semplice trovare quali sono gli ingressi DA e DB che devo fornire ai flip-flop per ottenere la transizione desiderata (il flip-flop D semplicemente «memorizza/ricopia» il valore dell'ingresso)



<b>A</b>	<b>B</b>	<b>X</b>	<b>A'</b>	<b>DA</b>	<b>B'</b>	<b>DB</b>
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

# Sintesi rete transizione dello stato

Per sintetizzare la rete combinatoria «minima» per la transizione dello stato che «pilota» gli ingressi dei due flip-flop devo semplicemente sintetizzare due reti logiche combinatorie con le mappe di Karnaugh. Le reti che producono in uscita DA e DB dati in ingresso A, B ed X.

<b>A</b>	<b>B</b>	<b>X</b>	<b>A'</b>	<b>DA</b>	<b>B'</b>	<b>DB</b>
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1



# Sintesi rete per transizione dello stato

Sintesi delle due reti logiche combinatorie con le mappe di Karnaugh. Le reti che producono in uscita DA e DB dati in ingresso A, B ed X

AB					
X		00	01	11	10
	0		1		1
	1				

$$D_A = \bar{A}\bar{B}\bar{X} + A\bar{B}\bar{X}$$

AB					
X		00	01	11	10
	0				1
	1	1	1	1	1

$$D_B = A\bar{B} + X$$

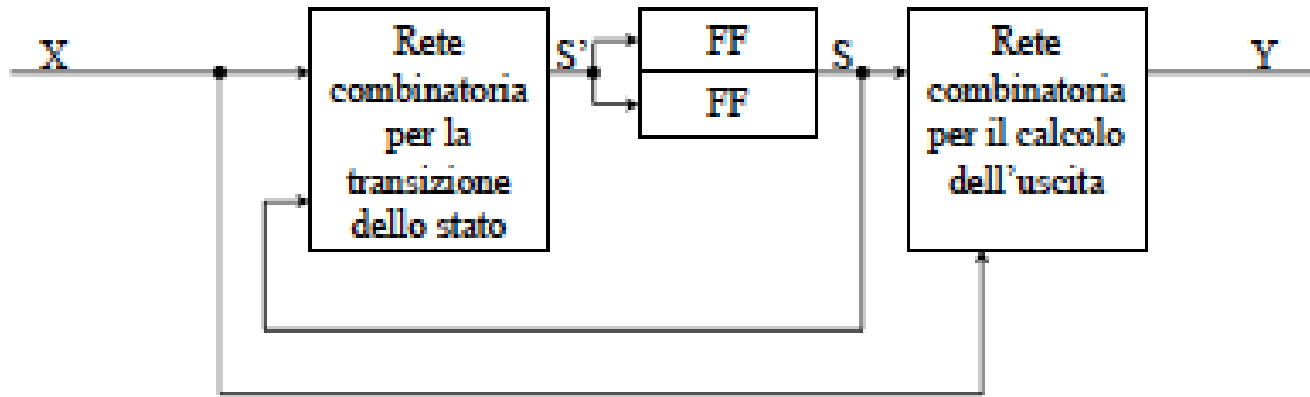
# Sintesi rete combinatoria per uscita

A	B	X	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	<b>1</b>
1	1	1	0

In questo caso la sintesi della rete combinatoria è così semplice che non servono neppure le mappe di Karnaugh. Z vale 1 solo in una configurazione.

$$Z = AB\bar{X}$$

# Schema logico della rete logica finale



- Per ottenere lo schema logico finale è sufficiente sostituire nello schema logico generale di cui sopra lo schema delle due reti che costituiscono la rete per la transizione dello stato:

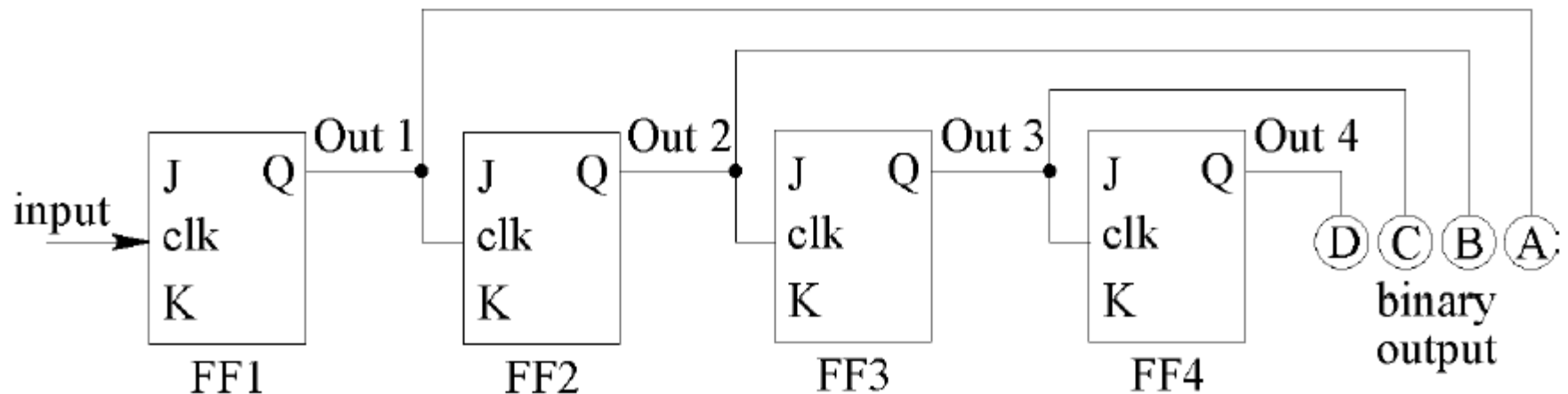
$$D_A = A' B X' + A B' X'$$

$$D_B = A B' + X$$

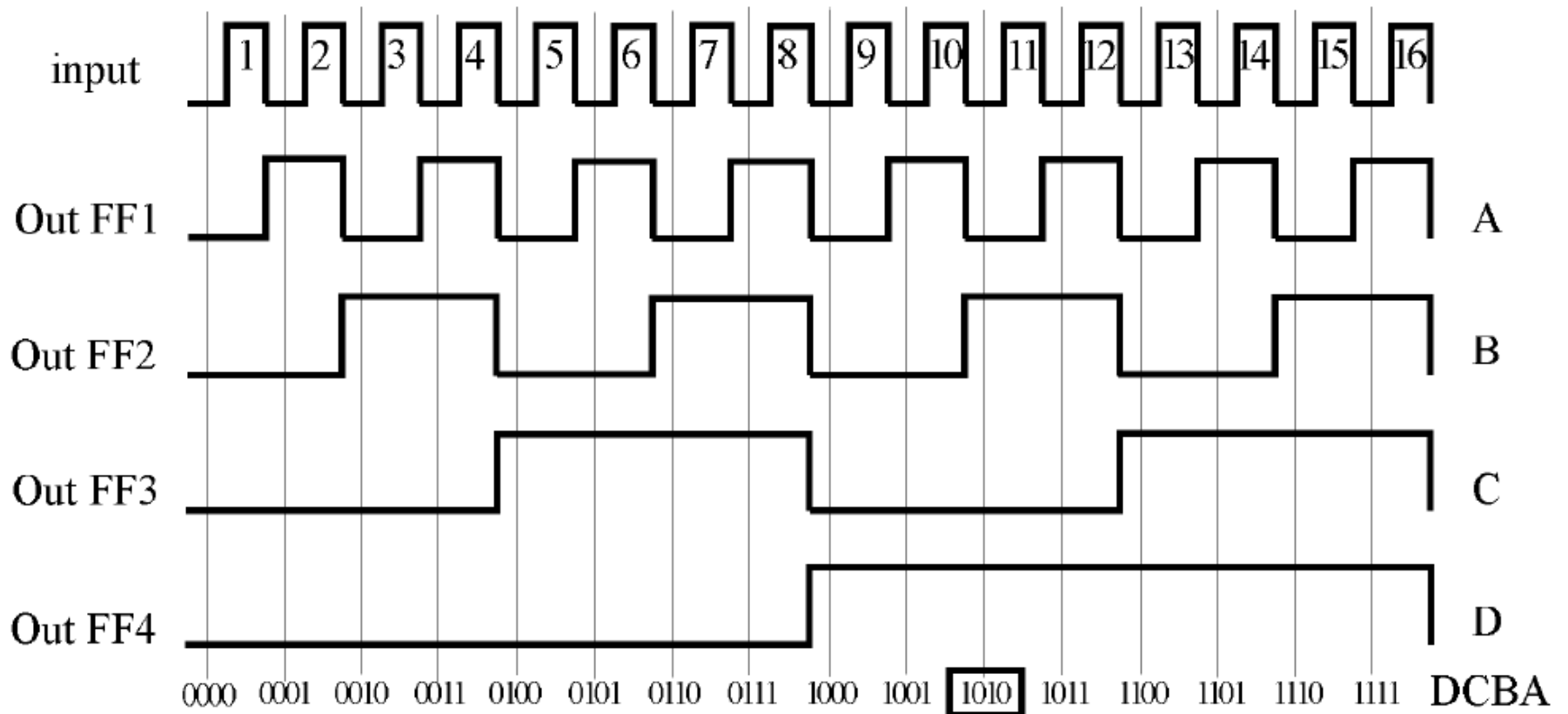
- E lo schema logico della rete per il calcolo dell'uscita  $Z$

# **Reti sequenziali notevoli: i contatori**

- Ad ogni segnale di clock, il segnale Q del flip flop i-esimo commuta con frequenza dimezzata rispetto a quella del flip flop (i-1)-esimo
- Le uscite dei flip flop (indicate con A, B, C, D) realizzano un contatore binario da 0 a 15



# Funzionamento del contatore



# Reti sequenziali notevoli: le memorie

