Architetture sincrone e asincrone

Sintesi di circuiti sequenziali

Federico Pedersini

Dipartimento di Informatica
Università degli Studi di Milano

Circuiti combinatori = circuiti **senza memoria:**

Le uscite al tempo *t* dipendono unicamente dagli ingressi al tempo *t*

$$out_i = f_i\left(in_1, in_2, \cdots, in_M\right) \quad , \quad i = 1 \cdots N$$

- Ogni uscita ***out$_i$*** è una **funzione logica** degli ingressi ***in$_i$ … in$_M$***
- Indipendenza dal tempo
- È impossibile modificare la "storia" del circuito ➔ non c'è memoria

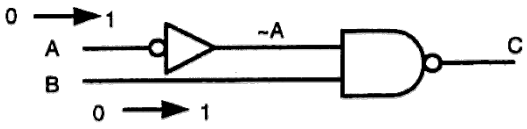
Circuito combinatorio

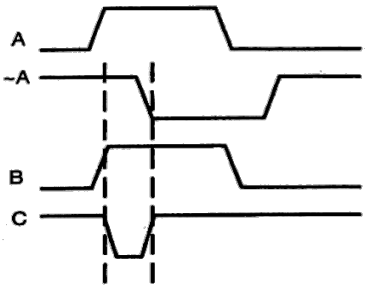
Data races

Al crescere della complessità dei circuiti combinatori, si verificano fenomeni critici:

DATA RACES: *transizioni spurie (glitch) su un segnale che deve rimanere costante*

➔ C'è un ***limite alla complessità*** dei circuiti combinatori!



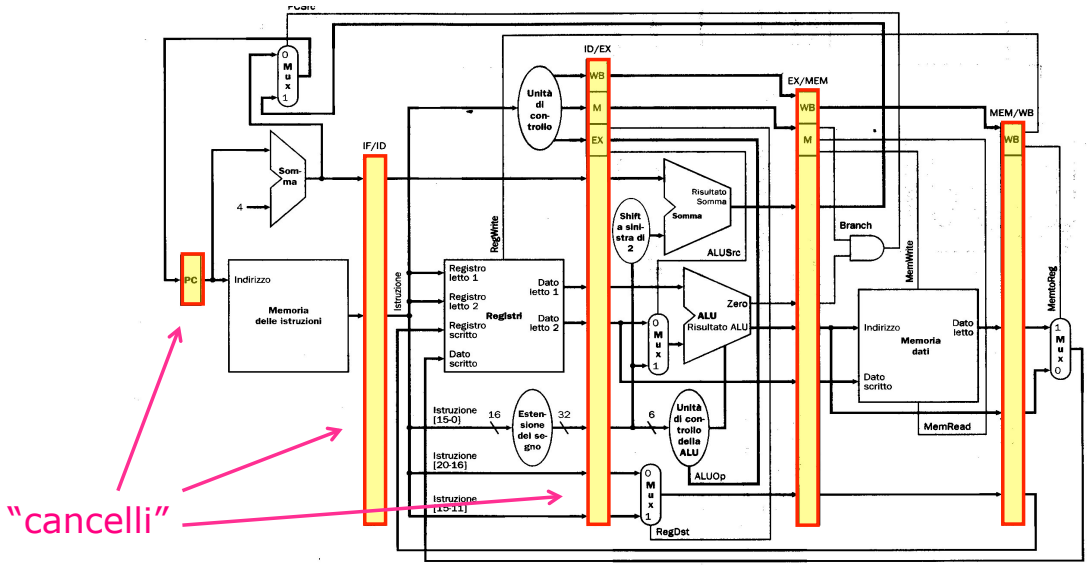


IDEA:
*spezzo il circuito in **SEZIONI** ed inserisco tra le sezioni delle **BARRIERE** al segnale, che APRO solo quando i segnali sono STABILI.*

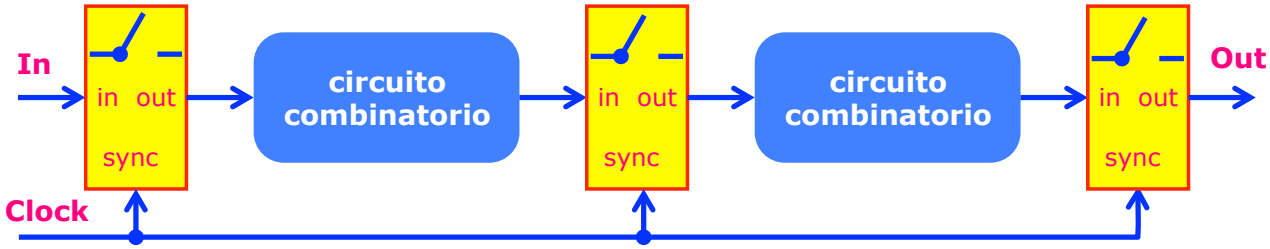
"Cancelli" nelle architetture

Esempio: CPU multi-ciclo

Un "cancello" dopo ogni stadio di elaborazione dell'istruzione



Struttura di architetture sincrone



Cancello → Circuito combinatorio → Cancello

- Sincronizzazione: la logica combinatoria deve terminare la propria commutazione in tempo utile
- Dimensionamento del periodo di clock **T**:
  - La commutazione del clock deve avvenire **dopo** che la logica combinatoria abbia terminato tutte le commutazioni
  - Il tempo necessario alla logica combinatoria per commutare dipende dal *cammino critico*

Perchè esiste il "clock" ?

Dove il "cancello" viene inserito, si ***sincronizza*** l'attività di elaborazione

- Altrimenti i cammini critici renderebbero il funzionamento del circuito sempre più casuale.

Cancelli = Barriere di sincronizzazione tra circuiti

- Memorizzano l'ingresso ad un certo istante
  → dispositivi di memorizzazione: **registri**
- Lo mantengono stabile in uscita, per un certo periodo
  → orologio in comune: **segnale di clock**

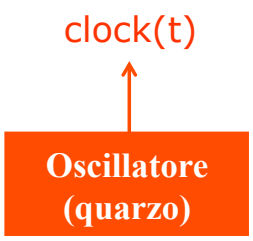
Orologio in comune → **circuiti sincroni**

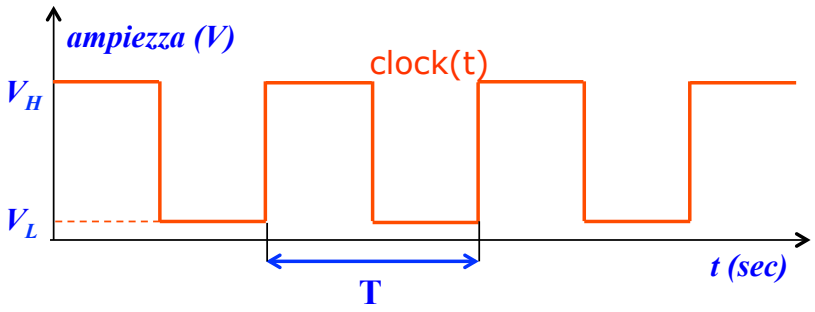
Presenza di registri → **circuiti sequenziali** (con memoria)

Circuiti *sincroni* e *asincroni*

- **Architettura sincrona:**
  l'elaborazione e propagazione dei segnali è scandita da un **orologio comune a tutto il circuito (clock)**
  - Il Clock regola l'attività dei "cancelli"
  - Il circuito ha il tempo di stabilizzarsi (transitori critici) fino al successivo impulso di Clock
- **Architettura asincrona:**
  l'elaborazione e propagazione dei segnali avviene **in modo incontrollato**, secondo le velocità di reazione dei circuiti
  - Non ci sono cancelli
  - Non devo mai aspettare l'impulso di clock → massima velocità
- **Progettazione sincrona**
  - il controllo dei transitori/cammini critici è limitato alla parte di circuito tra due cancelli (porte di sincronizzazione)
- **Progettazione asincrona**
  - Devo progettare il circuito in modo che nessun transitorio/cammino critico causi problemi → analisi di tutti i transitori critici possibili.
  - Eccezioni: TRANSPUTER (INMOS, UK)
  - Architettura di calcolatore con CPU asincrona
  - Complessità enorme, sia di progettazione che di programmazione → progetto abbandonato.

Clock





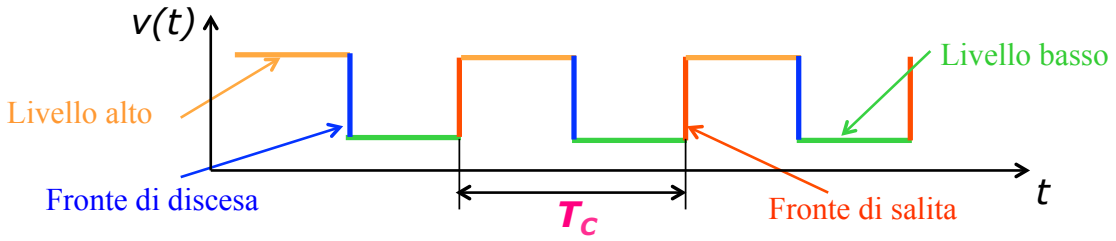
Periodo: **T** [s] durata di 1 ciclo (secondi)

Frequenza: ***f*** [Hz]=[s^{-1}] numero di cicli al secondo

$$\mathbf{T = 1/f}$$

- Tempo di salita e discesa trascurabile, rispetto al periodo T.

Utilizzo del clock



- **Architettura sensibile ai livelli:**
  - Le variazioni di stato avvengono quando il clock è alto (basso). La parte bassa (alta) del ciclo di clock permette la propagazione tra sottocircuiti, così che i segnali si siano stabilizzati quando il clock cambia livello.

- **Architettura sensibile ai fronti:**
  - Le variazioni di stato avvengono in corrispondenza di un fronte di clock (salita o discesa).

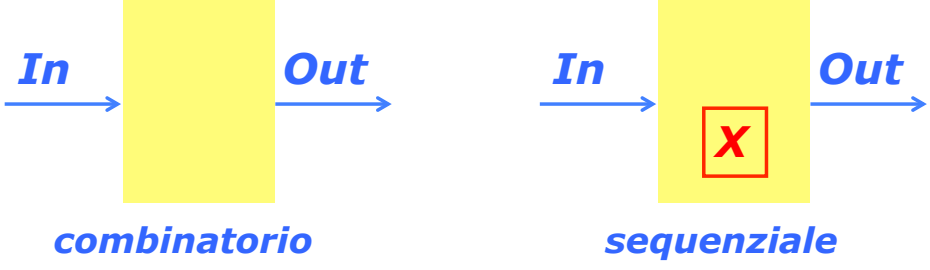
Circuiti sequenziali

- **Circuiti combinatori** = circuiti senza memoria
  - Gli output al tempo *t* dipendono unicamente dagli input al tempo *t*

    $Out = f(In)$
  - Per consentire ad un dispositivo di mantenere le informazioni, sono necessari circuiti con memoria

- **Circuiti sequenziali** = circuiti con memoria (stato)
  - La memoria contiene lo **stato (X)** del sistema:

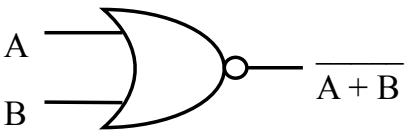
    $Out = f(In, X)$



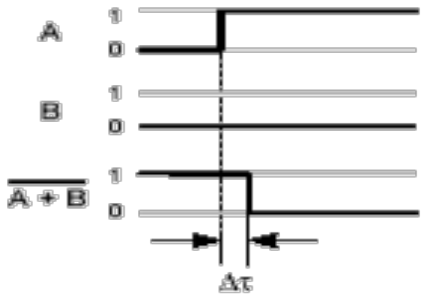
Latch & Bistabili

- Elemento cardine dei circuiti sequenziali è lo **stato**
  - Lo stato riassume il funzionamento negli istanti precedenti e deve essere immagazzinato (memorizzato)
  - Necessità della memoria (bistabili → registri → memorie)
- Elemento base della **memoria** è il **bistabile**
  - dispositivo in grado di **mantenere *indefinitamente* il valore di input**
  - Il suo valore di uscita coincide con lo stato
  - Memoria: insieme di bistabili (bistabili → registri → memorie)
- **Tipologie di bistabile**
  - Bistabili non temporizzati (asincroni) / temporizzati (sincroni).
  - Bistabili (sincroni) che commutano sul livello (**latch**) o sul fronte (**flip-flop**)

Latch Set-Clear (SR) asincrono

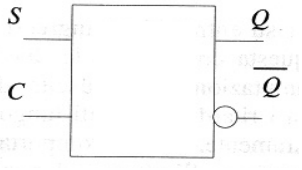


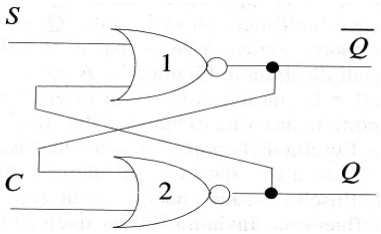
Porta NOR



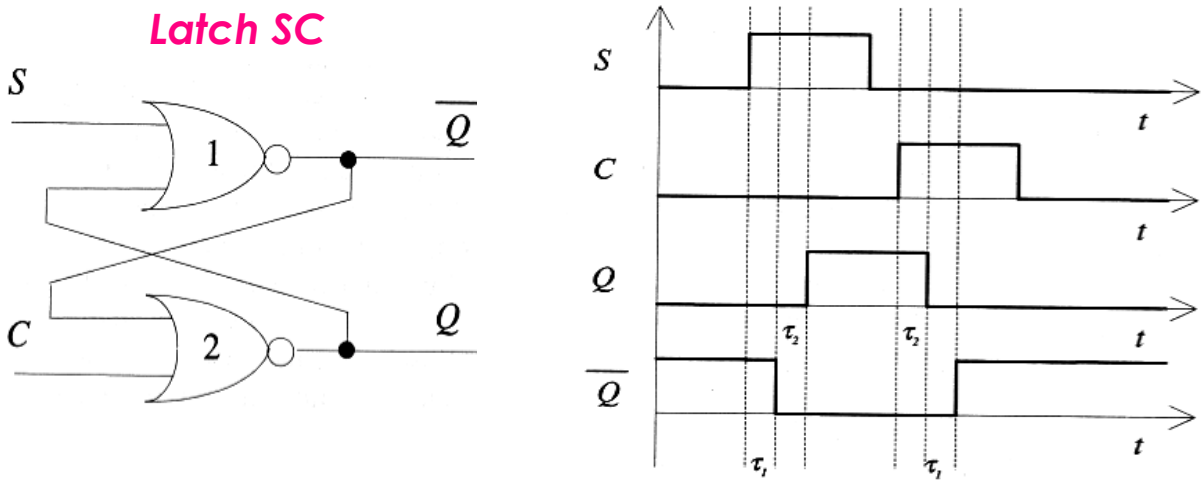
- Una coppia di porte NOR retroazionate può memorizzare un bit!

Latch Set-Clear





Funzionamento del Latch SC (SR)



- Funzionamento:
  - Set: C = 0, S → 1 Q → 1 (~Q → 0)
  - Reset: S = 0, C → 1 Q → 0 (~Q → 1)
  - Comportamenti anomali:
  - S = 1 , C: 0→1 Q=~Q=0 (anomalia)

Tabella delle transizioni

- Un circuito sequenziale non si può rappresentare mediante una tabella di verità
- Per i circuiti sequenziali si definisce la tabella delle transizioni

Tabella delle transizioni: Q* = f(Q, I) = f(Q, S,C)

Q: valore dell'uscita attuale: stato **corrente**

Q*: uscita al tempo successivo: stato **prossimo**

Q*= f(Q,I)

Q \ I	SC = 00	SC = 01	SC = 11	SC = 10
0	0	0	X	1
1	1	0	X	1

Q* = Q Clear / Reset SET

Tabella delle transizioni

- Tabella delle transizioni ***f*** :

$$Q^* = f(I, Q)$$

SC / Q	SC=00	SC=01	SC=11	SC=10
Q=0	**0**	**0**	**X**	**1**
Q=1	**1**	**0**	**X**	**1**

S	C	Q	Q*
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	**X**
1	1	1	**X**

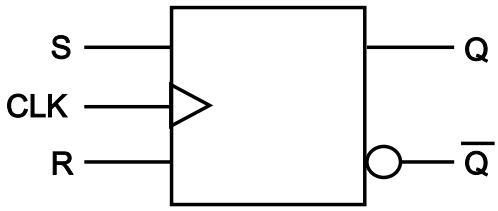
- Considerando lo stato Q come ingresso ottengo la tabella delle verità di Q*:

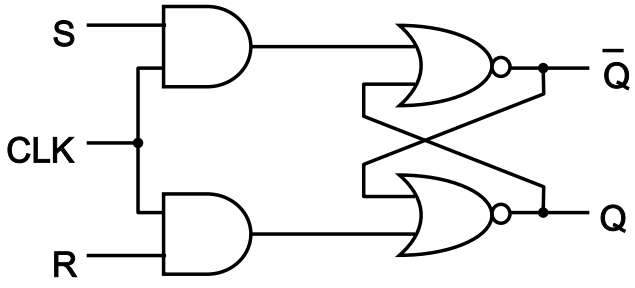
Latch SR sincrono

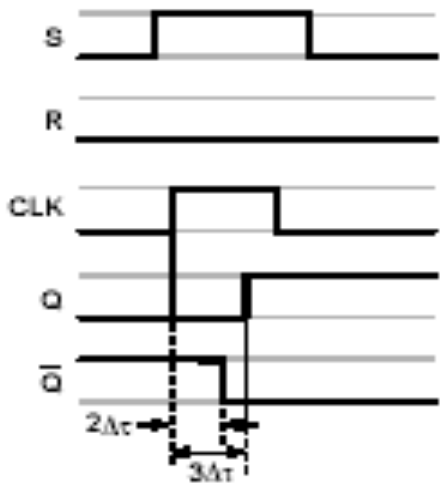
Latch Set-Reset (SR) sincrono

Struttura: Latch SR + Porte AND tra il clock e gli ingressi.

- Solo quando il clock è alto i "cancelli" (porte AND) fanno passare gli input → LATCH







Sintesi della funzione logica

T	Q	S	R	Q*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	X
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	X

Tabella delle transizioni: Q*= f(S,R,T,Q)

TQ	SR = 00	SR = 01	SR = 11	SR = 10
00	0	0	X	0
01	1	1	X	1
11	1	0	X	1
10	0	0	X	1

$$Q^* = \bar{T}Q\bar{S}\bar{R} + \bar{T}Q\bar{S}R + \bar{T}QS\bar{R} + T\bar{Q}S\bar{R} + TQ\bar{S}\bar{R} + TQS\bar{R} + \left(+\bar{T}QSR + T\bar{Q}SR\right) =$$
$$= \bar{T}Q\bar{R} + TQ\bar{R} + \bar{T}QR + T\bar{Q}S =$$
$$= \bar{T}Q + T\left(Q\bar{R} + \bar{Q}S\right)$$

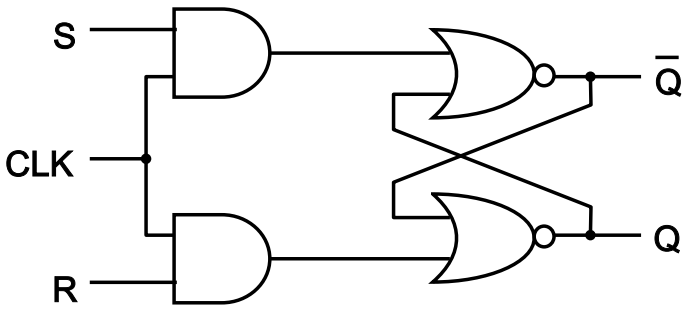
clock alto → Q* = Q~R + ~QS

clock basso → Q* = Q (*status quo*)

Analisi della funzione logica sintetizzata

Latch Set-Reset (SR) sincrono

$$Q^* = \bar{T}Q + T\left(Q\bar{R} + \bar{Q}S\right)$$



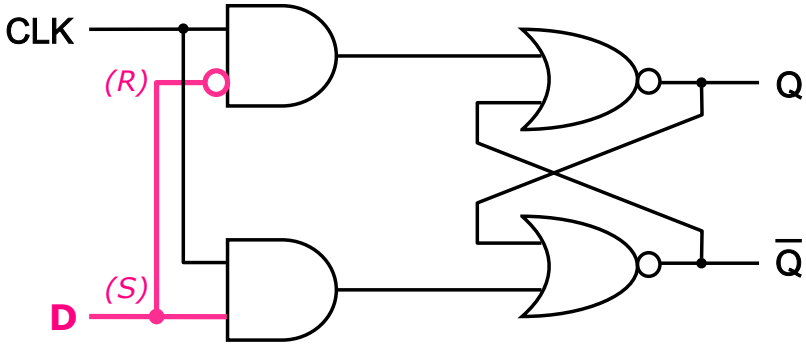
$$T = 1 \rightarrow \quad Q^* = Q\bar{R} + \bar{Q}S: \quad \begin{cases} \text{Set}: S = 1, R = 0 & Q^* = Q + \bar{Q} = 1 \\ \text{Reset}: S = 0, R = 1 & Q^* = 0 + 0 = 0 \end{cases}$$

$$T = 0 \rightarrow \quad Q^* = Q: \quad \text{"}status\ quo\text{"}$$

Comportamento del Latch D

Latch SR è un po' scomodo: 2 ingressi separati per memorizzare "0" o "1"

IDEA: piloto **S e R** con un solo ingresso: **D**
D = 1 → S=1, R=0 (Set) → **Q=1**
D = 0 → S=0, R=1 (Reset) → **Q=0**



LATCH D:

Clock ALTO (CLK=1):

- L'uscita Q insegue l'ingresso D (con 3Δτ di ritardo)

Clock BASSO (CLK=0):

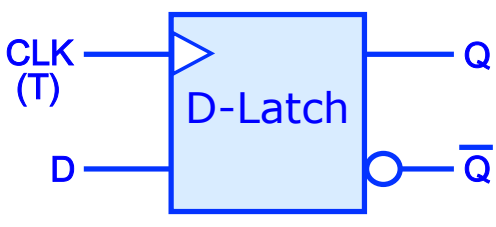
- L'uscita Q rimane bloccata sull'ultimo valore assunto

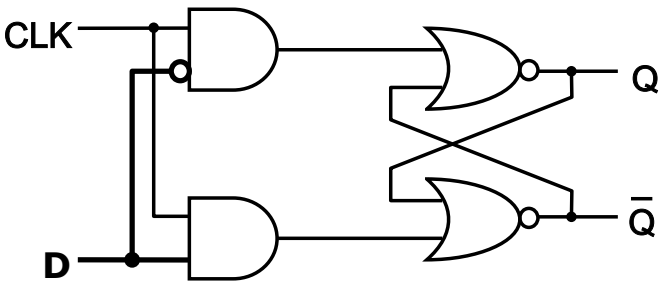
Latch D sincrono

LATCH D sincrono:

Memorizza il valore presente all'ingresso dati quando il clock è alto, altrimenti (clock=0) si mantiene sul valore memorizzato

if CLK = 1 **then** Q*=D **else** Q*=Q





La struttura del latch D

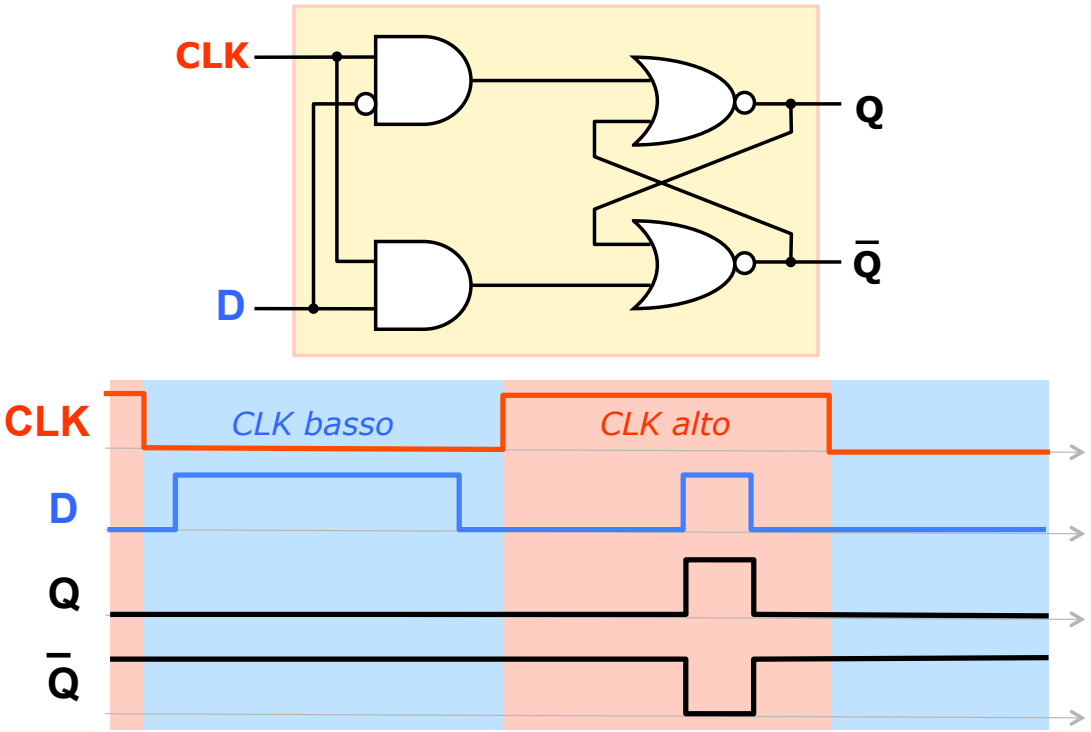


Tabella delle transizioni

- Dalla tabella delle transizioni calcolo l'espressione logica della funzione stato prossimo **Q* = f(T,Q,D):**

Tabella delle transizioni → Funzione stato prossimo *Q* = f(Q, T,D):*

T	Q	D	Q*
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

$$Q^* = \bar{T}Q\bar{D} + \bar{T}QD + T\bar{Q}D + TQD =$$

$$= \bar{T}Q + TD$$

Clock T alto:
→ Q* = D (input)

Clock T basso
→ Q* = Q (status quo)

$$T = 1 \rightarrow Q^* = D$$
$$T = 0 \rightarrow Q^* = Q$$

Latch: Bistabili level-sensitive

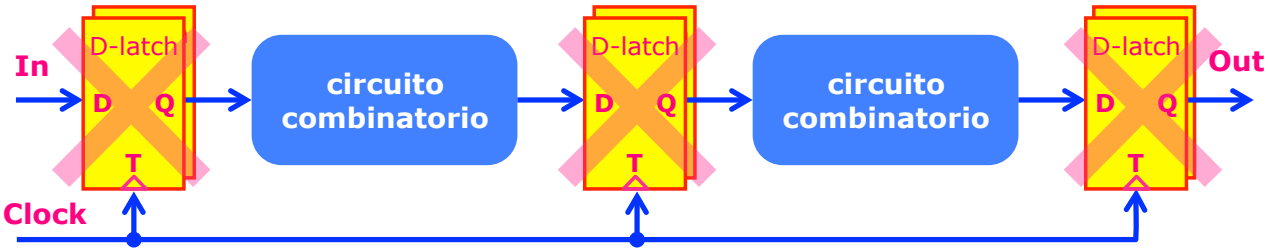
I latch sono dispositivi **trasparenti:** per tutto il tempo in cui il clock è a livello alto, il valore di D viene riportato in uscita

Clock = 1 ➔ Q*=D (uscita collegata all'ingresso)

Possono funzionare come **"cancelli"**?

NO! Nel momento in cui "apro" (T=1), i segnali possono attraversare **TUTTI gli stadi**.

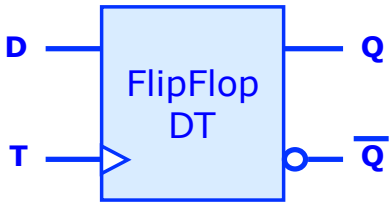
Soluzione: "cancello doppio"
Quando apro in ingresso, l'uscita è chiusa, e viceversa.



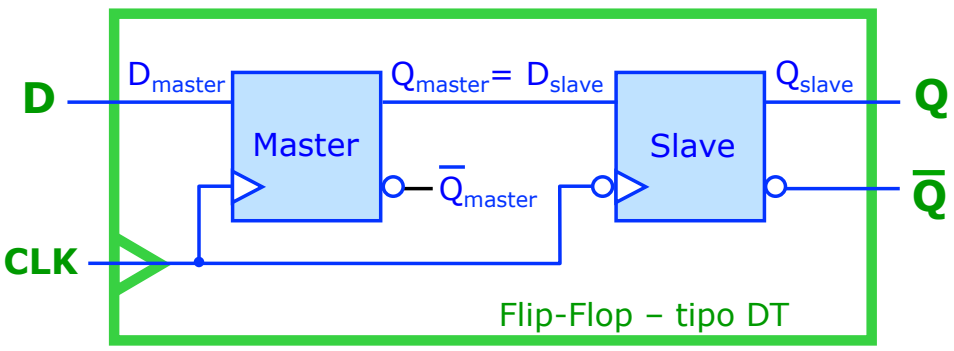
Bistabili "edge sensitive": i Flip-Flop

FLIP-FLOP: bistabile *edge sensitive* (attivo sui fronti del clock):

lo stato (uscita) commuta solo in corrispondenza del fronte di salita o di discesa del clock.

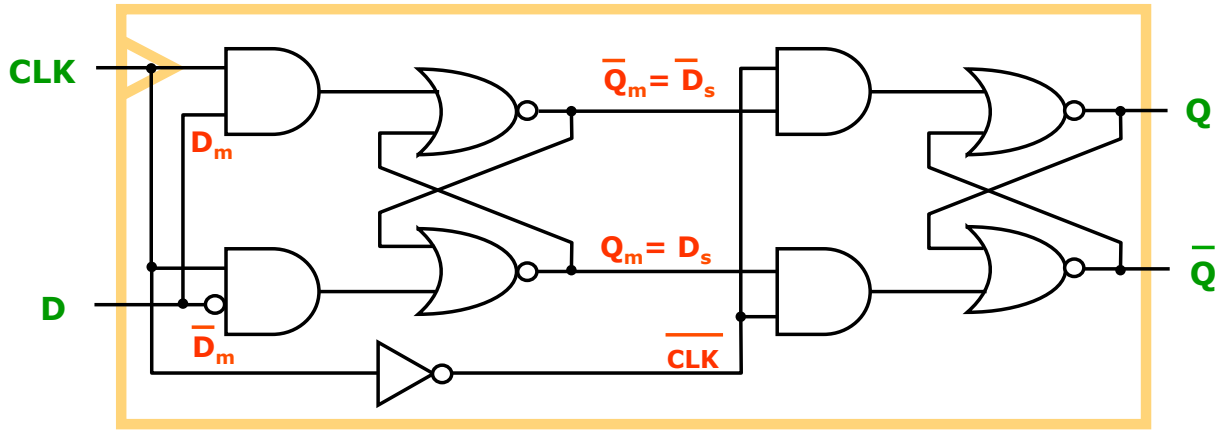


Flip-Flop tipo DT
configurazione "Master-Slave":

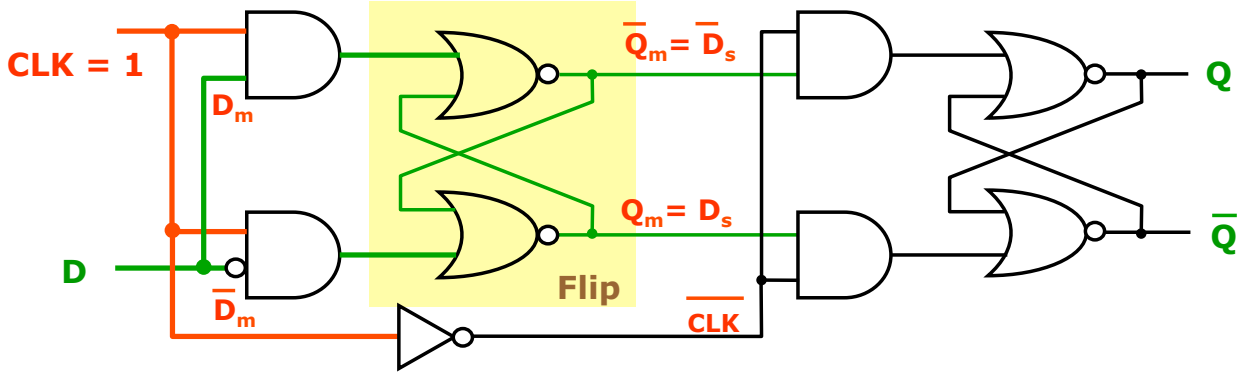


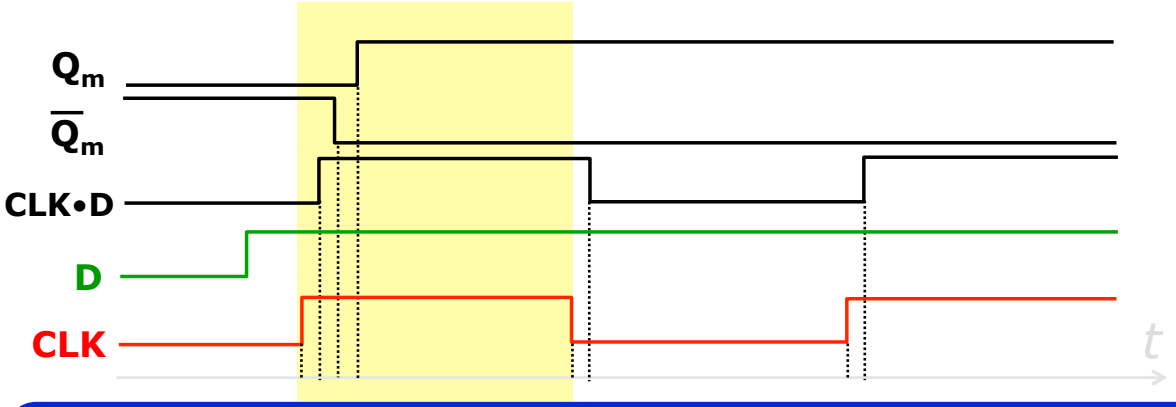
Flip-flop: struttura master–slave

- **FLIP (clock ALTO)**: L'ingresso D viene memorizzato nel latch MASTER
  - L'ingresso è aperto, ma l'uscita è bloccata
- **FLOP (clock BASSO)**: l'uscita stabile del latch MASTER viene propagato al latch SLAVE
  - L'ingresso è bloccato, l'uscita è stabile.

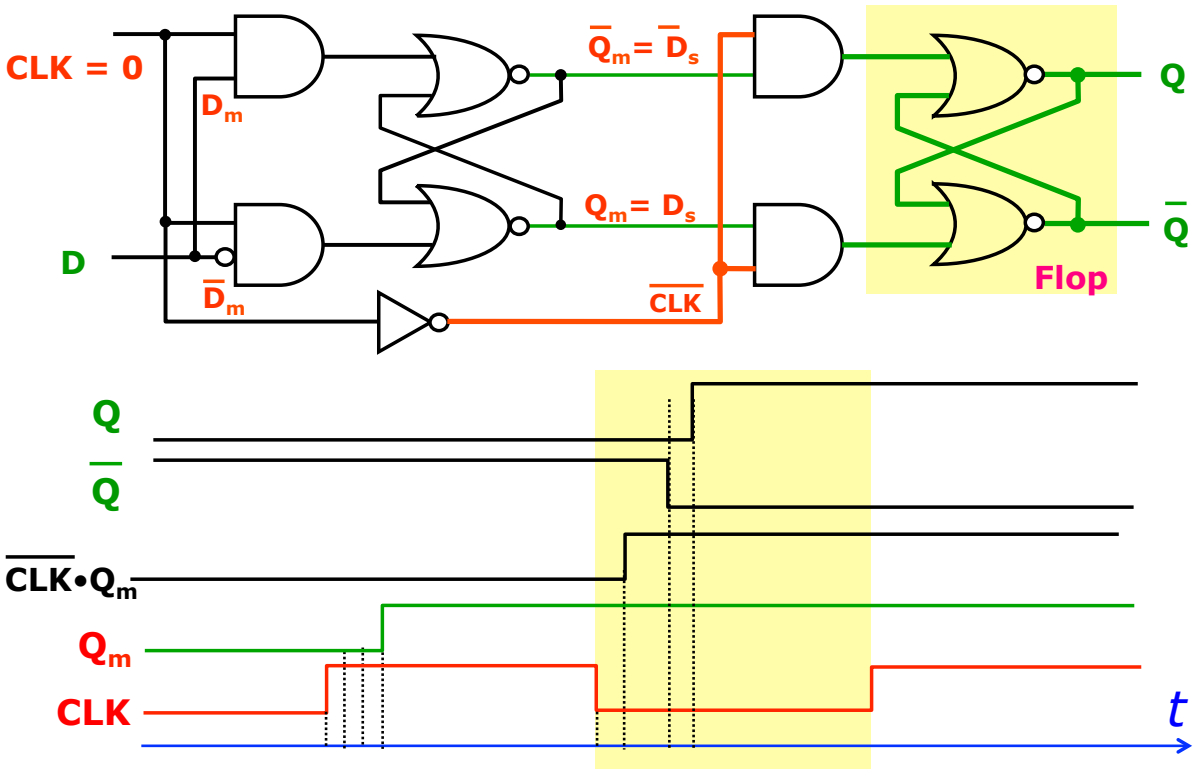


Funzionamento: *FLIP*



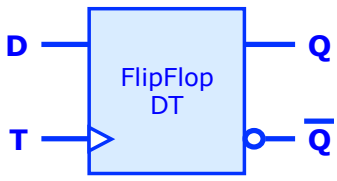


Funzionamento: **FLOP**



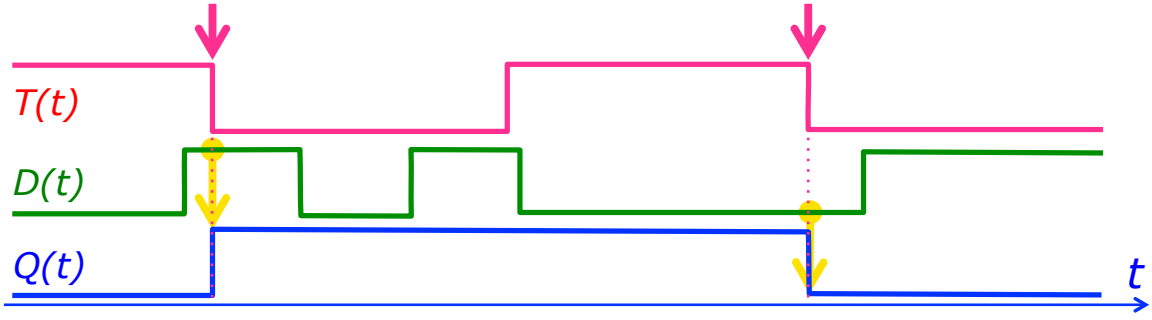
Funzionamento dei Flip-Flop

- Fronte di SALITA – FLIP
  - Attivato lo stadio **MASTER**
  - Uscita (stadio slave) invariata



- **Fronte di DISCESA – FLOP**
  - Attivato stadio **SLAVE**
  - Memorizzato il dato sull'ingresso:
  - Presenta il dato memorizzato in uscita:
  - Ingresso isolato

D → STATO
STATO → Q



Latch: Bistabili level-sensitive

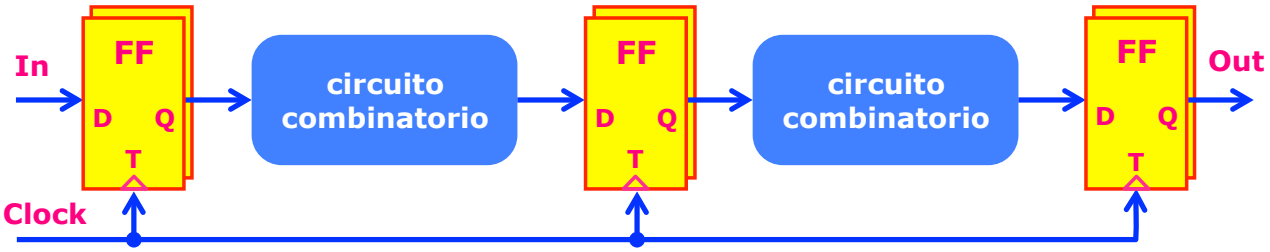
I latch sono dispositivi **trasparenti:** per tutto il tempo in cui il clock è a livello alto, il valore di D viene riportato in uscita

Clock = 1 ➔ Q*=D (uscita collegata all'ingresso)

Possono funzionare come **"cancelli"**?

SÌ! Nel momento in cui "apro" (T=1), i segnali possono attraversare **TUTTI gli stadi**.

Soluzione: "cancello doppio"
Quando apro in ingresso, l'uscita è chiusa, e viceversa.

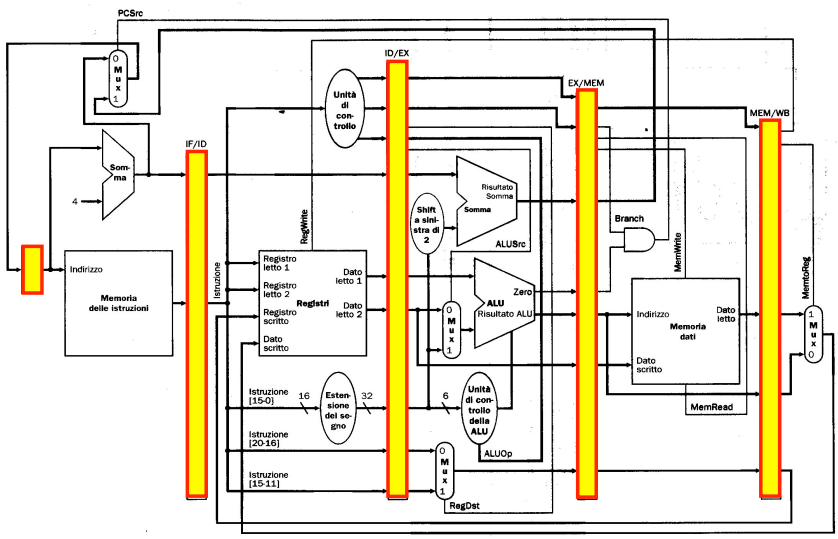


Sincronizzazione mediante flip-flop

- I "cancelli" devono **disaccoppiare** i diversi sottosistemi logici
  - "memorizzare" i segnali e "rilanciarli", tutto ad un determinato istante
  - **prima e dopo**, uscite **non sensibili** agli ingressi
  - **Cancello doppio**: **ingresso** e **uscita,** mai aperti contemporaneamente

Cancelli:

registri costituiti da gruppi di **flip-flop**



Registro: unità di memorizzazione di parole di **N bit**

Struttura: **N Flip-flop tipo DT**

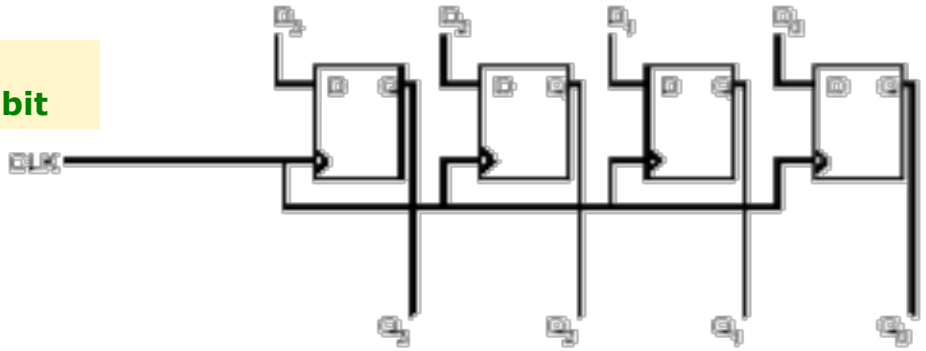
Operazioni: **LETTURA**:

I dati memorizzati sono sempre presenti sulle uscite Q

SCRITTURA:

L'impulso di CLK memorizza i dati sugli ingressi D

Esempio: registro a 4 bit



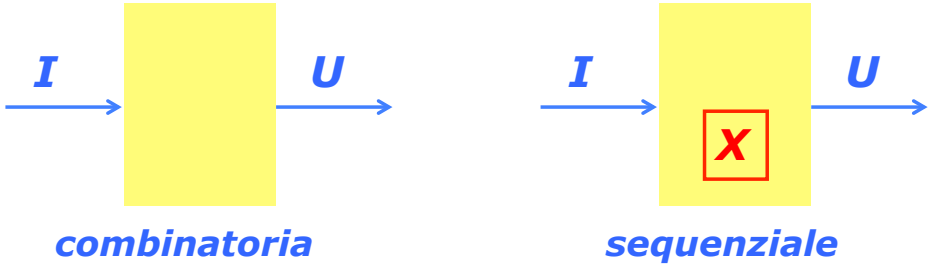
Circuiti sequenziali: macchine a stati finiti

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Macchine sequenziali

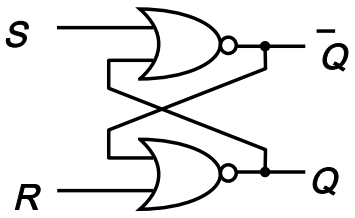
- Macchina combinatoria: **U** = *f* (**I**)
  - senza memoria, uscita dipende solo dagli ingressi
- Macchina sequenziale: **X*** = *f* (**X**, **I**)
  U = *g*(**X**)
  - 2 funzioni: **uscita** e **stato prossimo**
  - esiste la memoria: lo **STATO**

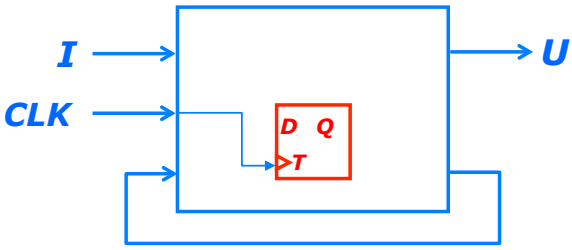


combinatoria *sequenziale*

Macchine sequenziali

- Elemento necessario di ogni macchina sequenziale è la **retroazione**
  - Uscita riportata in ingresso
  - Bistabile: (macchina sequenziale elem.): 2 porte NOR +retroazione
- Macchina sequenziale sincrona
  - Impiega bistabili sincroni
  - Es: Flip-Flop tipo DT





Macchina a Stati Finiti - di Moore

- Una Macchina a Stati Finiti (MSF) è definita dalla quintupla:

$$< X,\ I,\ Y,\ f(\cdot),\ g(\cdot) >$$

X: insieme degli stati (in numero finito).

I: alfabeto di ingresso: l'insieme dei simboli che si possono presentare in ingresso. Con **n** ingressi, avremo $\mathbf{2^n}$ possibili configurazioni.

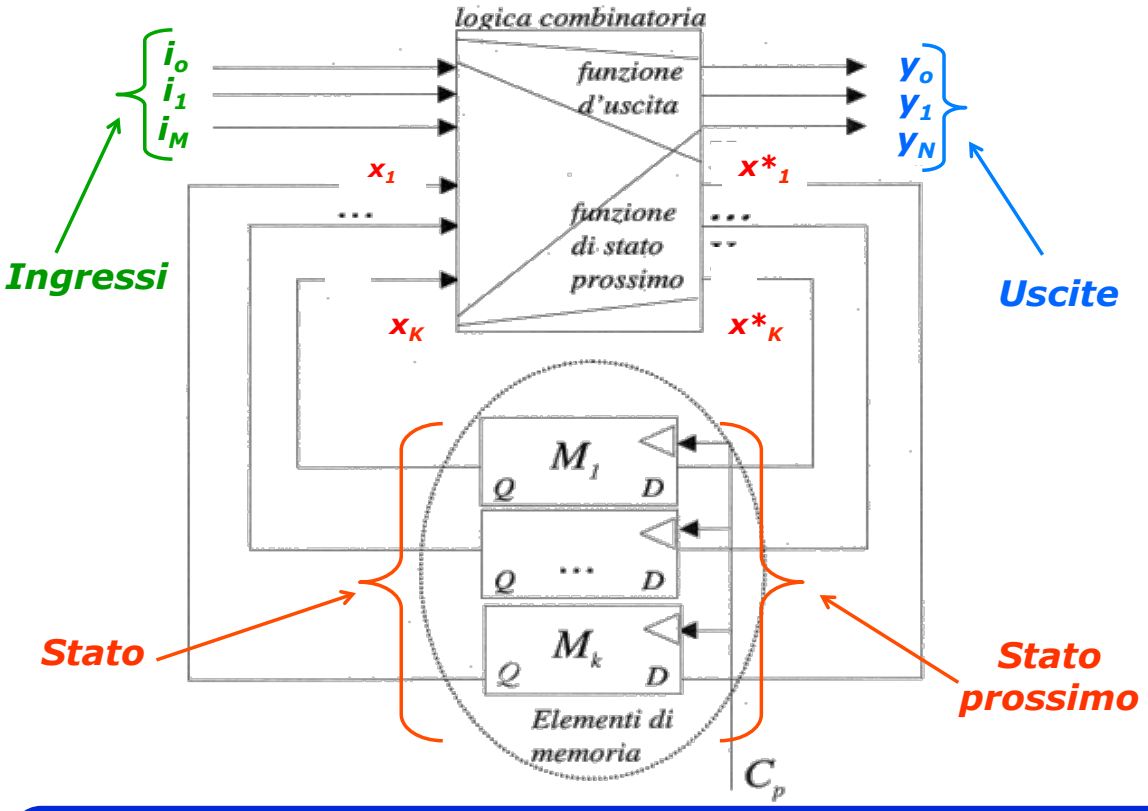
Y: alfabeto di uscita: l'insieme dei simboli che si possono generare in uscita. Con **m** uscite, avremo $\mathbf{2^m}$ possibili configurazioni

***f*(·)**: funzione stato prossimo: $\mathbf{X^* = f(X, I)}$
Definisce l'evoluzione della macchina nel tempo, in modo deterministico

***g*(·)**: funzione di uscita: $\mathbf{Y = g(X)}$ (macchina di **Moore**)
$\mathbf{Y = g(X, I)}$ (macchina di **Mealy**)

- Per il buon funzionamento della macchina è previsto uno **stato iniziale**, al quale la macchina può essere portata mediante un comando di **reset**.

La macchina sequenziale di Huffman



Macchina di Moore: State Transition Table (STT)

- **STT**: State Transition Table (Tabella delle transizioni di stato)
  - Per ogni coppia: <stato attuale, ingresso>
    definisco uscita **y** e stato prossimo **x***

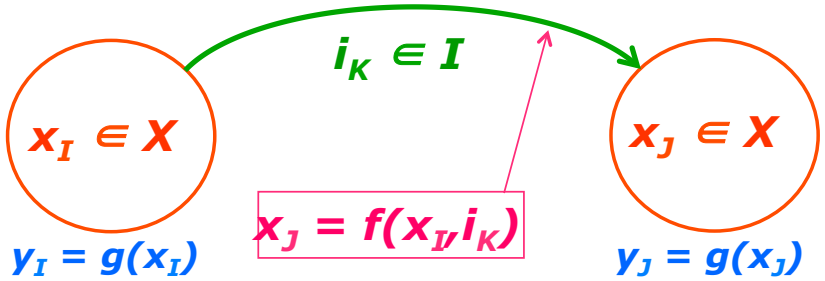
    $(x_i \in X, i_j \in I) \rightarrow y(x_i) ; x^*(x_i, i_j)$

  - Esempio: M stati (log2M bit di stato), N ingressi (log2M bit d'ingresso):

	i_1	i_2	...	i_N	Y
x_1	$x^*(x_1,i_1)$	$x^*(x_1,i_2)$		$x^*(x_1,i_N)$	$y(x_1)$
x_2	$x^*(x_2,i_1)$	$x^*(x_2,i_2)$		$x^*(x_2,i_N)$	$y(x_2)$
...					...
x_M	$x^*(x_M,i_1)$	$x^*(x_M,i_2)$		$x^*(x_M,i_N)$	$y(x_M)$

Macchina di Moore: State Transition Graph (STG)

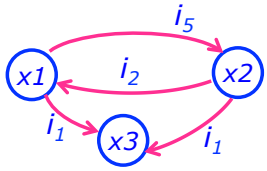
- **STG: State Transition Graph**
  (*Diagramma degli Stati o Grafo delle transizioni*)
  - Ad ogni nodo è associato uno stato: $x_i \in X$
    ... ed un valore della funzione d'uscita: $y_i \in Y$, $y_i = g(x_i)$
  - Un arco orientato da uno stato x_i ad uno stato x_j, contrassegnato da un simbolo (di ingresso) i_K, rappresenta una transizione che si verifica quando la macchina, essendo nello stato x_i, riceve come ingresso i_K



Sintesi circuito sequenziale

Sequenza operazioni di sintesi
di un circuito sequenziale mediante modello a FSM:

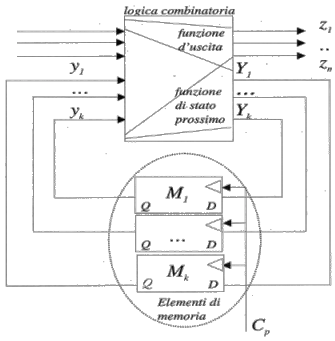
$$I = \{i_1, i_2, \cdots, i_M\},$$
$$Y = \{y_1, y_2, \cdots, y_P\}$$



1. Definizione insiemi ingressi **I** e uscite **Y**
2. Costruzione grafo delle transizioni **STG**
   ➔ definizione insieme X, f(X,I) e g(X)
3. Traduzione **STG ➔ STT**

	i_1	...	i_N	Y
x_1	$x^*(x_1,i_1)$		$x^*(x_1,i_N)$	$y(x_1)$
x_2	$x^*(x_2,i_1)$		$x^*(x_2,i_N)$	$y(x_2)$
...				...
x_M	$x^*(x_M,i_1)$		$x^*(x_M,i_N)$	$y(x_M)$

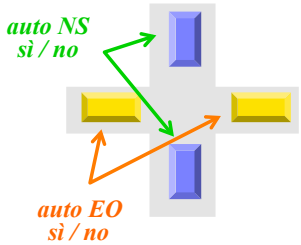
4. **Codifica binaria** degli elementi di **X, Y, I** nella **STT**
   ➔ f(X,I) e g(X) diventano funzioni logiche
5. Sintesi delle funzioni logiche f(X,I) e g(X)
   ➔ circuito finale con struttura di Huffman



$$x_i^* = f_i(x_1, \cdots, x_Q,\ i_1, \cdots, i_R)$$
$$y_i = g_i(x_1, \cdots, x_Q)$$

Progetto con macchina di Moore

Esempio: controllore di un semaforo



SEMAFORO:

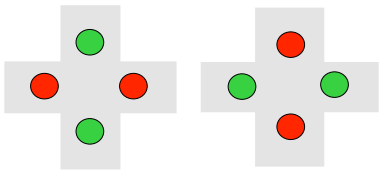
- Incrocio tra 2 strade: nord-sud (NS) ed est-ovest (EO) controllate da un semaforo
  - per semplicità consideriamo solamente rosso e verde
- Il semaforo può commutare ogni 30 secondi
  - Macchina sincrona, clock con frequenza = ?
- E' presente un sensore in grado di "leggere", per ogni direttrice, se esiste almeno un'auto in attesa, oppure un'auto che si accinga ad attraversare (condizioni trattate allo stesso modo).
- Il semaforo deve cambiare colore, da rosso a verde, quando esiste un'auto in attesa sulla sua direttrice.
- Se ci sono auto in attesa sulle entrambe le direttrici il semaforo deve cambiare colore (al termine del tempo di commutazione)

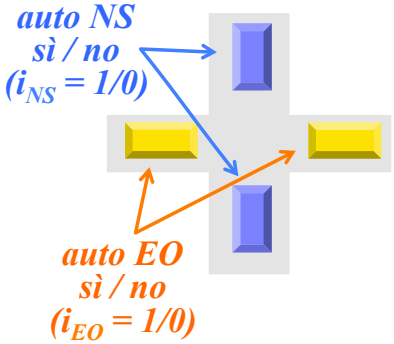
SEMAFORO: Stato, Ingresso, Uscita

INGRESSI

- Auto NS presente / non presente
  - AutoNS=1/0
- Auto EO presente / non presente
  - AutoEO = 1/0

→ 2 bit di INGRESSO → 4 configurazioni d'ingresso: ***I = {00, 01, 10, 11}***





STATO

- Semaforo NS VERDE, semaforo EO ROSSO
- Semaforo NS ROSSO, semaforo EO VERDE

X = {VerdeNS, VerdeEO}

→ 1 bit di STATO (1 flip-flop):

USCITE: = STATO: ***Y = X***

- LuceEO verde (LuceNS rossa)
- LuceNS verde (LuceEO rossa)

→ 2 configurazioni d'uscita → 1 bit di USCITA

Funzionamento: uscita

- Funzione uscita: **Y = g(X)**
  - Per ogni stato, definire l'uscita della macchina
- Uscita ↔ STATO → **Y = X**
  - VerdeNS → Verde sulla direttrice NS, rosso sulla direttrice EO
  - VerdeEO → Verde sulla direttrice EO, rosso sulla direttrice NS
- Luce Verde NS / Rosso EO = **VerdeNS**
- Luce Verde EO / Rosso NS = **VerdeEO**

Funzionamento: stato prossimo

- **Stato prossimo:** evoluzione dello stato, in funzione dello stato attuale e degli ingressi attuali

$$X(t+1) = X^* = f(X(t), I)$$

- **X(t+1) = X*** = "VerdeNS"
  - Se X(t)="VerdeNS" AND non ci sono auto sulla direttrice EO
  - Se X(t)="VerdeEO" AND ci sono auto sulla direttrice NS

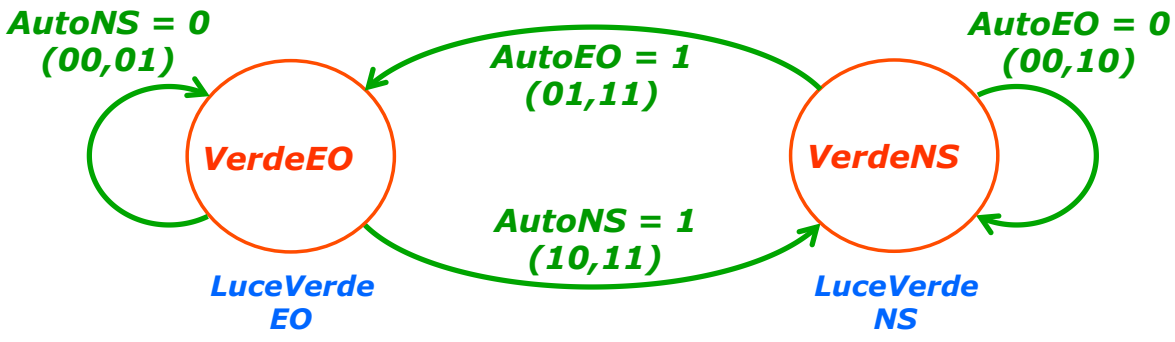
  VerdeNS · ~autoEO + VerdeEO · autoNS → X* = VerdeNS

- **X(t+1) = X*** = "VerdeEO"
  - Se X(t)="VerdeEO" AND non ci sono auto sulla direttrice NS
  - Se X(t)="VerdeNS" AND ci sono auto sulla direttrice EO

  VerdeEO · ~autoNS + VerdeNS · autoEO → X* = VerdeEO

STG del semaforo

Sintesi STG:



Risultato:

- Funzione stato prossimo:

  VerdeNS* = VerdeNS · ~autoEO + VerdeEO · autoNS
  VerdeEO* = VerdeEO · ~autoNS + VerdeNS · autoEO

- Funzione uscita: **Y = X**

STT del semaforo

Sintesi STT:

X \ $I(i_{EO}, i_{NS})$	$i_{EO}=0$ $i_{NS}=0$	$i_{EO}=0$ $i_{NS}=1$	$i_{EO}=1$ $i_{NS}=0$	$i_{EO}=1$ $i_{NS}=1$	Uscita
VerdeNS	VerdeNS	VerdeNS	VerdeEO	VerdeEO	Luce VerdeNS
VerdeEO	VerdeEO	VerdeNS	VerdeEO	VerdeNS	Luce VerdeEO

Funzione stato prossimo: $X^* = f(X, I)$

Funzione uscita: $Y = g(X)$

STT del semaforo: CODIFICA binaria

Codifica binaria della STT:

Stato:

- (**VerdeNS**/RossoEO, RossoNS/**VerdeEO**) → **X = {0, 1}**

Ingresso:

- 2 Variabili: (AutoEO, AutoNS) → 1 =“presente”, 0 =“assente”
- 4 Configurazioni: **$I = (i_{EO}, i_{NS}) = \{00, 01, 10, 11\}$**

Uscita:

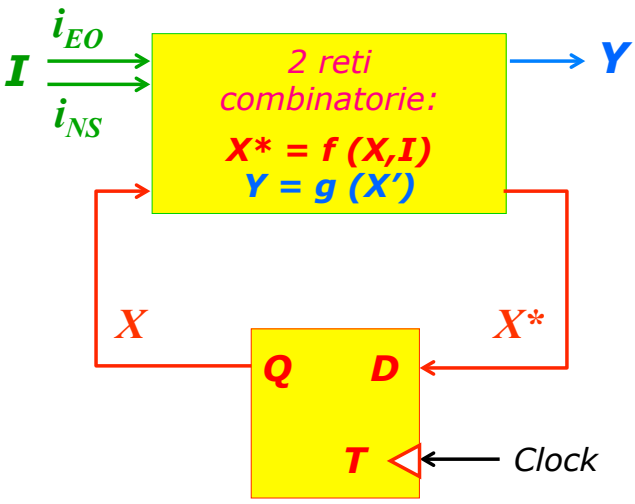
- (Luce_VerdeNS, Luce_VerdeEO) → **Y = {0, 1}**

X \ $I(i_{EO}, i_{NS})$	00	01	10	11	Uscita Y
0	0	0	1	1	0
1	1	0	1	0	1

Sintesi della MSF del semaforo

Sintesi del circuito:

- 2 stati (0,1) → 1 flip-flop DT
- 2 linee di ingresso
- 1 linea d'uscita



Architettura di **Huffman:**

- Rete combinatoria che implementa:
  - stato prossimo: ***f*(X,I)**
  - uscita: ***g*(X)**

Sintesi della MSF del semaforo

- Mediante la STT codificata in binario, posso esprimere **X*** e **Y** come somma di prodotti:
  - cerco i mintermini:

$$X^* = \overline{X}\, i_{EO}\, \overline{i_{NS}} + \overline{X}\, i_{EO}\, i_{NS} + X\, i_{EO}\, \overline{i_{NS}} + X\, \overline{i_{EO}}\, \overline{i_{NS}} =$$

$$= \overline{X}\, i_{EO} + X\, \overline{i_{NS}}$$

$$Y = X$$

X \ $I_{(i_{EO},i_{NS})}$	00	01	10	11	Y
0	0	0	1	1	0
1	1	0	1	0	1

MSF del semaforo: sintesi del circuito

- funzioni logiche rete combinatoria:

$$X^* = \overline{X}\, i_{EO}\, \overline{i_{NS}} + \overline{X}\, i_{EO}\, i_{NS} + X\, i_{EO}\, \overline{i_{NS}} + X\, \overline{i_{EO}}\, \overline{i_{NS}} =$$
$$= \overline{X}\, i_{EO} + X\, \overline{i_{NS}}$$
$$Y = X$$