

CALCOLATORI

Cenni alle reti logiche

Giovanni Iacca

giovanni.iacca@unitn.it

*Lezione basata su materiale preparato
dal Prof. Luigi Palopoli*



UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria
e Scienza dell'Informazione

Cosa sono le reti logiche?

- Fino ad ora abbiamo visto
 - Rappresentazione dell'informazione
 - Aritmetica dei calcolatori
- L'obiettivo principale di questo corso è mostrare come funziona (e come si progetta) un computer
- Per fare ciò, abbiamo bisogno di fare una piccola digressione su come si progettano i circuiti logici (esistono interi corsi dedicati a questo)

Valori logici

- I computer moderni sono realizzati tramite circuiti elettronici
- Trattandosi di elementi digitali avremo due livelli fondamentali:
 - Alto, asserito (1): associato alla tensione di alimentazione Vdd
 - Basso, negato (0): associato alla massa (tensione = 0)
- Altri livelli di tensione sono non significativi e assunti solo in fase transitoria

Reti logiche

- Le porte logiche sono dei circuiti che trasformano alcuni valori logici in ingresso in altri valori logici in uscita
- Le porte logiche sono di due tipi
 - Combinatorie
 - ✓ Relazione funzionale tra ingresso e uscita
 - ✓ Non hanno memoria
 - ✓ L'uscita dipende solo dal valore dell'ingresso
 - Sequentziali
 - ✓ L'uscita dipende dalla storia degli ingressi passati e non solo dal valore attuale
 - ✓ Hanno memoria (detta anche *stato* della rete)

Tabella di verità

- Una possibile maniera di specificare una rete logica combinatoria è tramite una tabella di verità che elenca i valori delle uscite in corrispondenza delle varie combinazioni in ingresso

INPUT			OUTPUT		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Algebra di Boole

- Una maniera più compatta è di specificare le funzioni logiche combinatorie tramite espressioni algebriche definite con l'algebra di Boole
- Esistono tre operatori di base
 - **AND**
 - ✓ Rappresentato tramite il simbolo di prodotto (\bullet), es. $A \bullet B$
 - ✓ Produce 1 se entrambi gli operandi sono 1, 0 negli altri casi
 - **OR**
 - ✓ Rappresentato tramite il simbolo della somma (+), es. $A + B$
 - ✓ Produce 0 se entrambi gli operandi sono 0, 1 negli altri casi
 - **NOT**
 - ✓ Rappresentato da una barra, es. \bar{A}
 - ✓ Ha l'effetto di invertire il valore logico

Algebra di Boole

- Alcune semplici regole ci permettono di manipolare (e semplificare) facilmente le espressioni logiche
 - Identità: $A+0=A$, $A \cdot 1=A$
 - Regola «zero e uno»: $A + 1 = 1$, $A \cdot 0=0$
 - Regola dell'inversa: $A + \bar{A}=1$, $A \cdot \bar{A}=0$
 - Regola commutativa: $A+B=B+A$, $A \cdot B=B \cdot A$
 - Regola associativa: $A+(B+C)=(A+B)+C$,
 $A \cdot (B \cdot C)=(A \cdot B) \cdot C$
 - Regola distributiva: $A \cdot (B+C)=(A \cdot B)+(A \cdot C)$,
 $A+(B \cdot C)=(A+B) \cdot (A+C)$

Algebra di Boole

- In più esistono due regole molto importanti, dette di De Morgan:

$$\begin{aligned}\overline{A \cdot B} &= \overline{A} + \overline{B} \\ \overline{A + B} &= \overline{A} \cdot \overline{B}\end{aligned}$$

- Queste regole ci dicono che se abbiamo un elemento logico che implementa l'operazione NAND (NOT AND), oppure NOR (NOT OR), tutti gli altri operatori logici si possono ricavare da questo

Algebra di Boole - Esempio

- Torniamo alla tabella che abbiamo visto prima:

INPUT			OUTPUT		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

- Possiamo verificare facilmente:

$$D = A + B + C$$

$$F = A \cdot B \cdot C$$

Algebra di Boole - Esempio

- Torniamo alla nostra tabella

INPUT			OUTPUT		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

E vale 1:

- Se A=1, B=1, C=0
oppure
- Se A=1, C=1 B = 0
oppure
- Se B=1, C=1, A= 0

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

- O usando De Morgan

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

Porte logiche

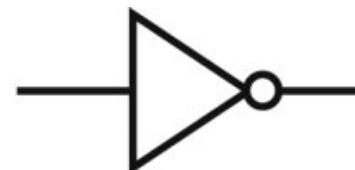
- In effetti, esistono dei circuiti elettronici (cosiddette «porte logiche») che implementano proprio gli operatori Booleani fondamentali



AND



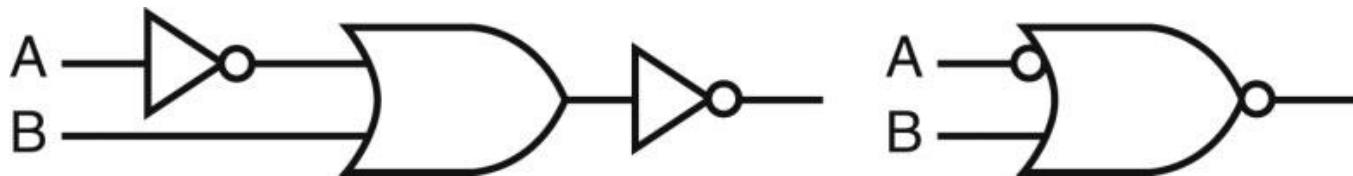
OR



NOT

Porte logiche

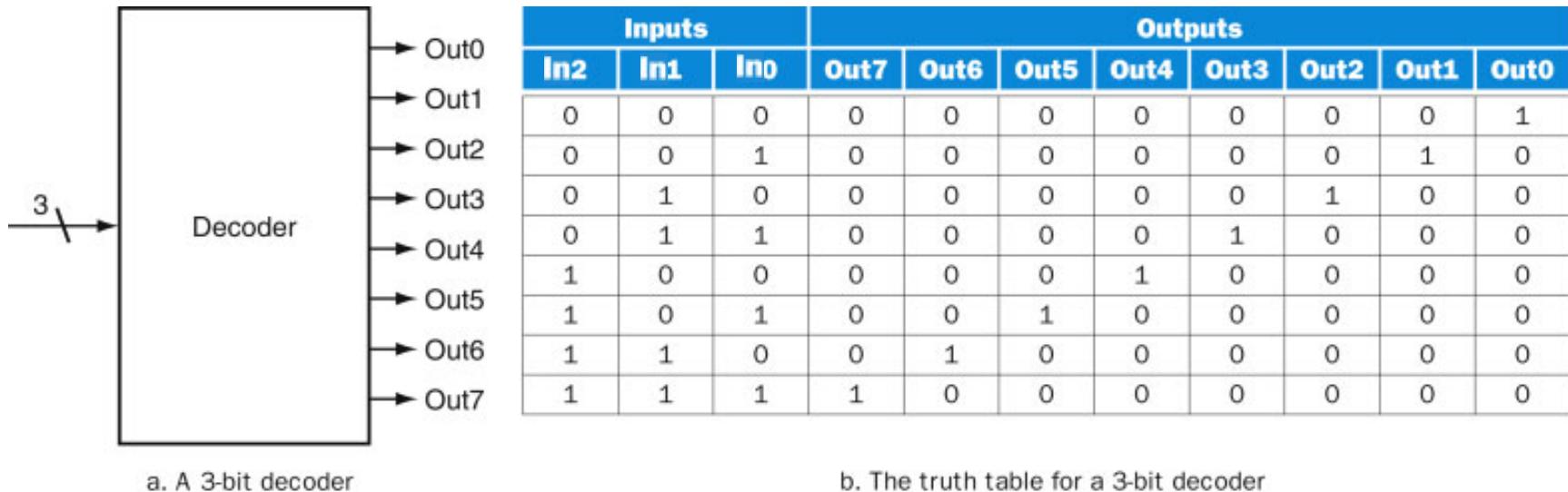
- Le porte si possono combinare tra di loro (con il NOT che viene indicato di solito tramite un cerchio sull'input corrispondente)



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

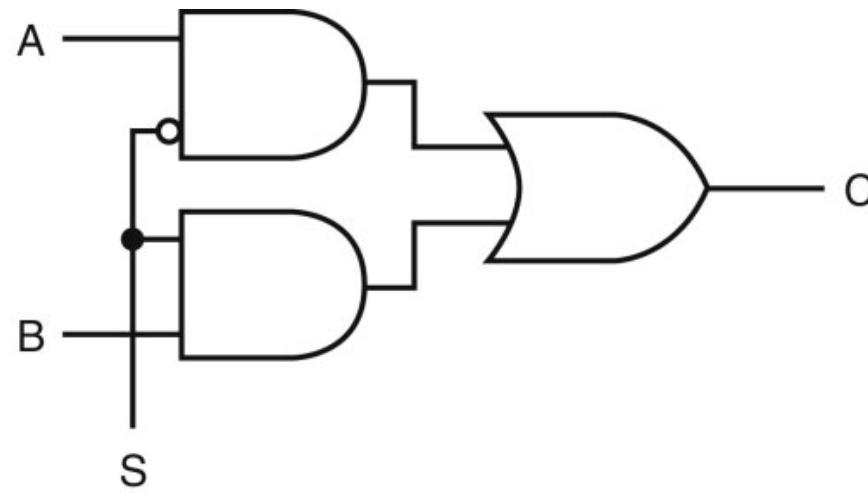
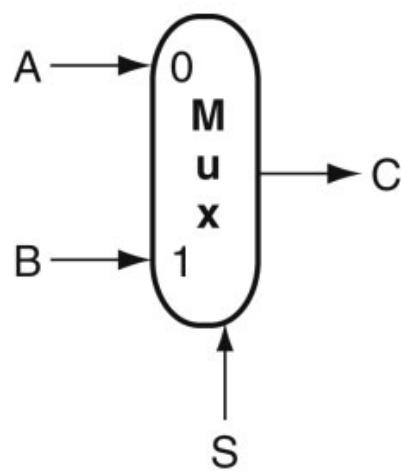
Alcuni circuiti

- Decoder



Alcuni circuiti

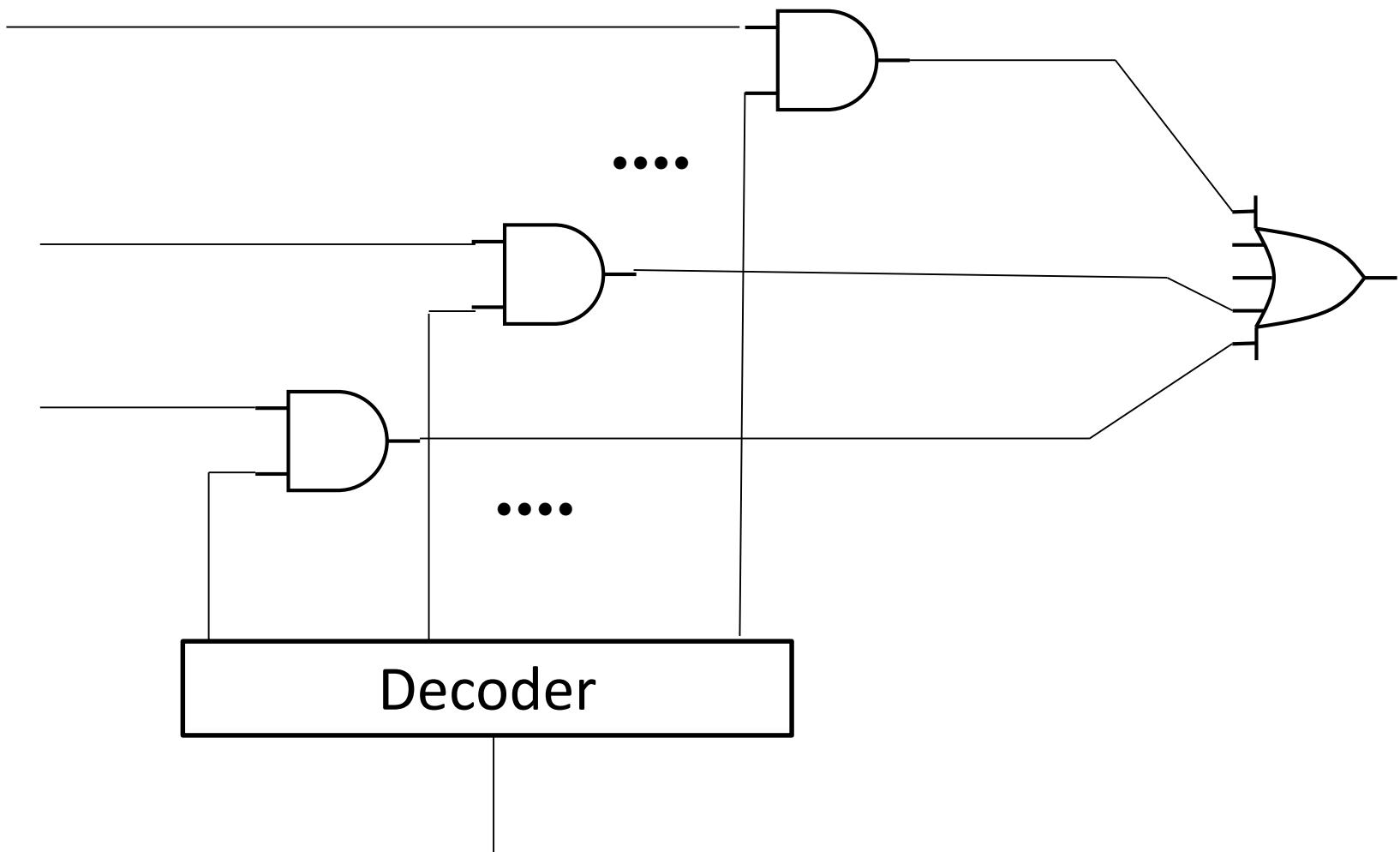
- Multiplexer



- Deviatore che sulla base di un input di controllo, determina quale degli input passa

Alcuni circuiti

- Multiplexer a N vie



Forme canonica SP

- Abbiamo visto calcolare l'espressione logica equivalente ad una tabella di verità è semplice
- Basta prendere ciascuna riga uguale a 1 e scrivere un termine di prodotto logico (AND) dettato dalla configurazione degli ingressi
- A quel punto si può fare la somma (OR) di tutti i prodotti individuati

Altro esempio

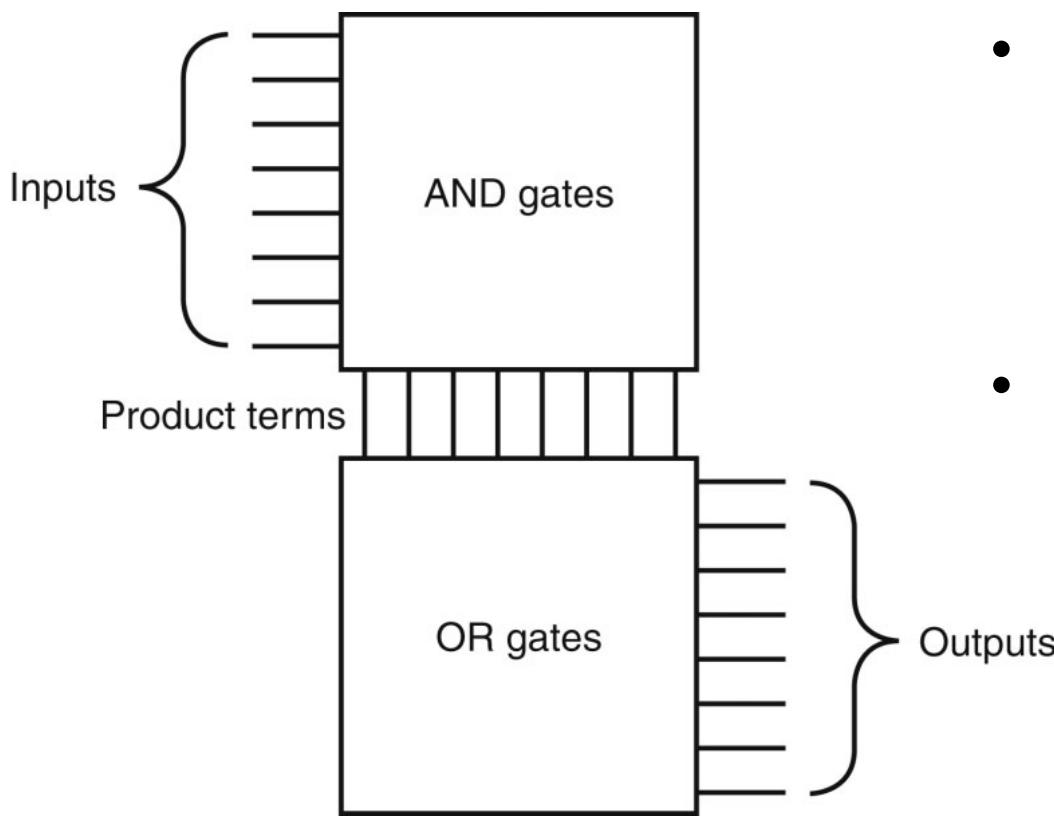
- Consideriamo come ulteriore esempio:

INPUT			OUTPUT
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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

Programmable Logic Array (PLA)

- La struttura che abbiamo visto si compone di due stadi: la prima è una barriera di AND (cosiddetti anche «mintermini»), la seconda è una barriera di OR



- La dimensione totale del PLA è data dalla somma del piano AND (numero di mintermini) e del piano OR (numero di uscite)
- Caratteristiche importanti:
 - Ci sono porte logiche solo per le configurazioni di ingressi che producono 1 in uscita
 - Se un mintermine è condiviso tra varie uscite, lo si può riutilizzare (basta inserirlo una sola volta nel piano AND)

Esempio

- Torniamo all'esempio precedente:

INPUT			OUTPUT		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

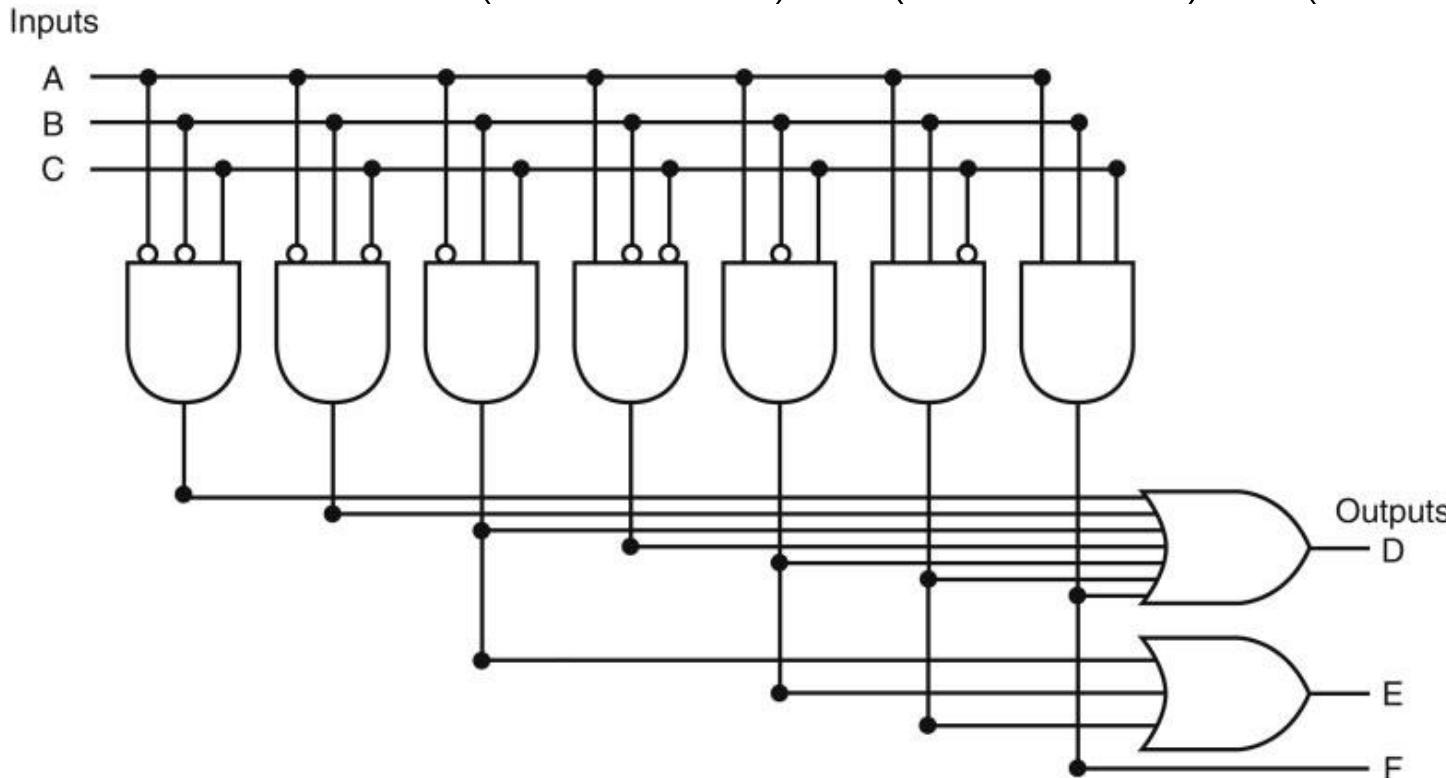
Esempio

- Implementazione tramite porte logiche:

$$D = A + B + C$$

$$F = A \cdot B \cdot C$$

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



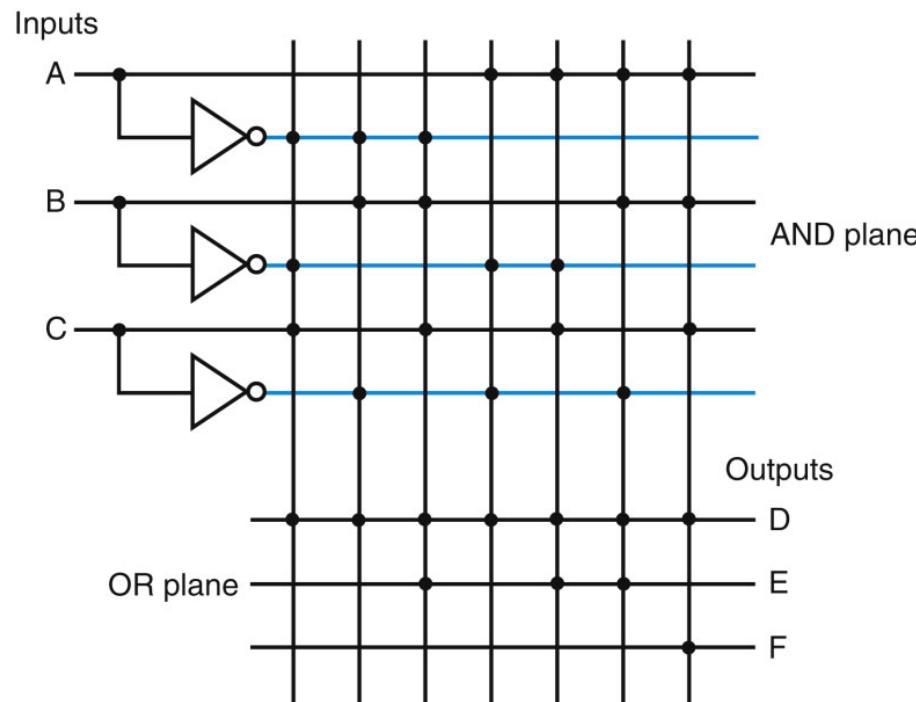
Esempio

- Una diversa rappresentazione

$$D = A + B + C$$

$$F = A \cdot B \cdot C$$

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



Costo

- Le funzioni logiche possono essere implementate in maniera diversa (più o meno efficiente)
- Per **COSTO** di una rete logica si intende normalmente la somma del numero di porte e del numero di ingressi della rete (indipendentemente dal fatto che siano positivi o negativi)
- E' possibile trovare delle implementazioni di una rete che hanno costi diversi

Minimizzazione di funzioni logiche

- La minimizzazione di alcune espressioni logiche è banale, in altri casi è necessario applicare le regole algebriche in modo “furbo”
- Esempio:

$$\begin{aligned}f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 \\&= \bar{x}_1 \bar{x}_2 (\bar{x}_3 + x_3) + x_1 \bar{x}_2 (\bar{x}_3 + x_3) \\&= \bar{x}_1 \bar{x}_2 + x_1 \bar{x}_2 \\&= (\bar{x}_1 + x_1) \bar{x}_2 \\&= \bar{x}_2\end{aligned}$$

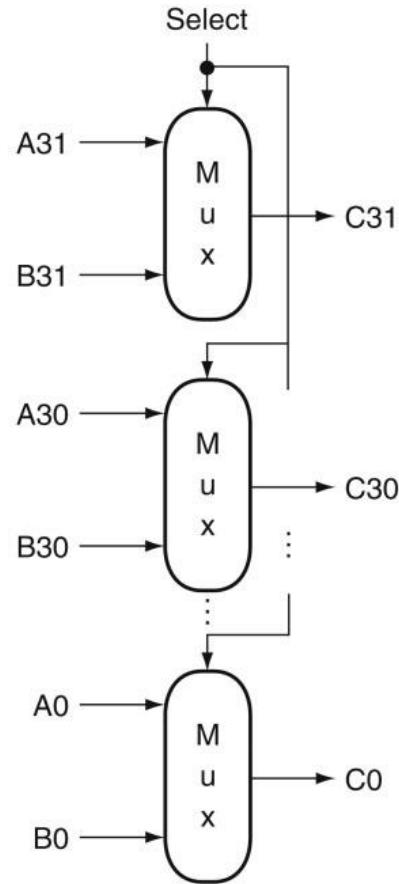
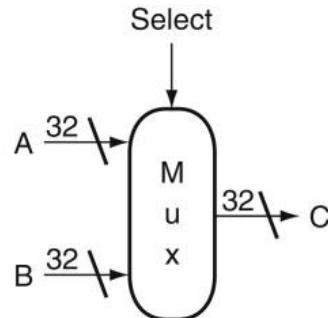
Minimizzazione di funzioni logiche

- Esistono metodi di minimizzazione sistematici basati sull'applicazione iterativa di queste regole
- Altri metodi sono basati su rappresentazioni grafiche (mappe di Karnaugh), ma si applicano solo a casi più semplici
- Questo argomento si chiama “sintesi logica”, ed è coperto nel corso di Reti Logiche

Array di elementi logici

- Molto spesso si costruiscono array di elementi che operano su dati complessi
- Ad esempio come realizzare un multiplexer che opera su un bus a 32 bit utilizzando elementi a un bit
- **BUS**: insieme di «fili di ingresso» (ad esempio 32), che viene visto come un singolo segnale logico

Esempio: Multiplexer a 32 bit



a. A 32-bit wide 2-to-1 multiplexor

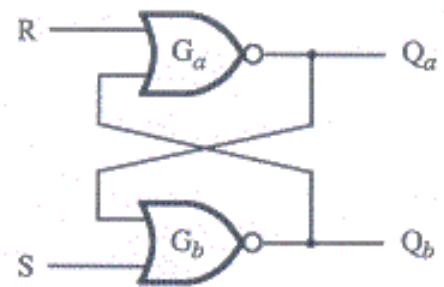
b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

Reti sequenziali

- Le funzioni logiche e le relative reti di implementazione viste fino ad ora sono note come
“reti combinatorie”
- Le reti combinatorie non hanno una nozione “esplicita” del tempo e non hanno memoria del passato: in ogni istante di tempo l’uscita dipende solamente dagli ingressi nell’istante considerato
- Tuttavia, in molte applicazioni è necessario introdurre una «memoria» nel sistema...
- Quasi sempre diamo per scontato che un elaboratore sia in grado di **memorizzare** informazioni

Reti sequenziali

- La memoria in una rete logica si ottiene con una “reazione” (o retroazione), ovvero ridirezionando l’uscita di alcune porte in ingresso ad altre porte del **medesimo stadio** (considerando che la rete logica può essere organizzata in stadi, o strati, successivi), in modo da formare un “anello ” in cui gli ingressi dipendono dalle uscite (e viceversa)
- La reazione complica in modo significativo l’analisi e la sintesi di una rete logica
- La memoria deriva dal fatto che gli ingressi “ricordano” il passato della rete attraverso il valore delle uscite passate



(a) Network

S	R	Q_a	Q_b
0	0	0/1	1/0 (No change)
0	1	0	1
1	0	1	0
1	1	0	0

(b) Truth table

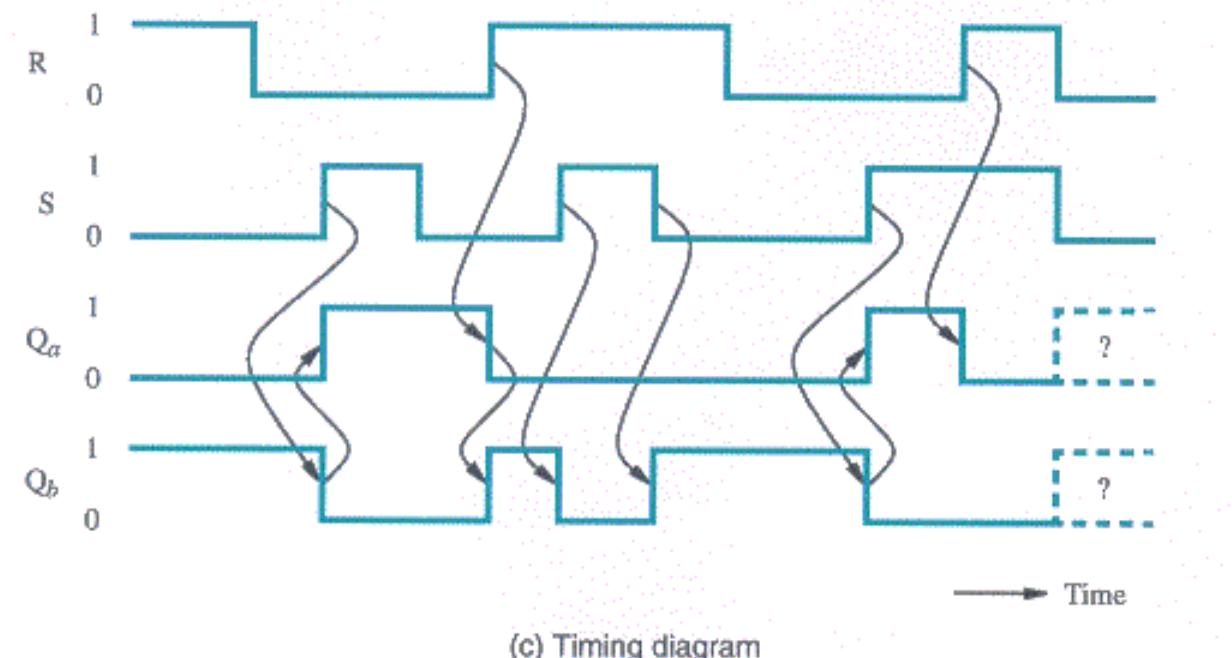


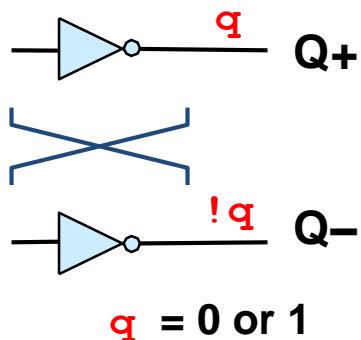
Figure A.24. A basic latch implemented with NOR gates.

Elemento base di memoria (latch)

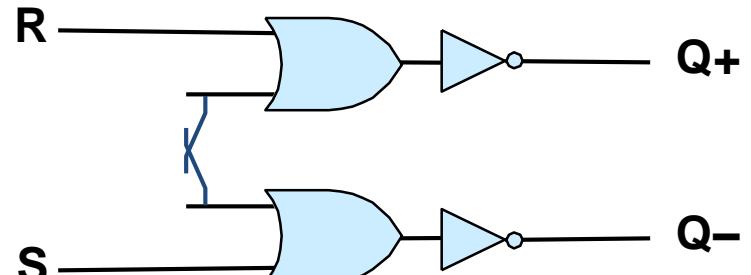
realizzazione con
due porte NOR
e schema di
“temporizzazione”
della tavola di
verità

Memorizzare un bit

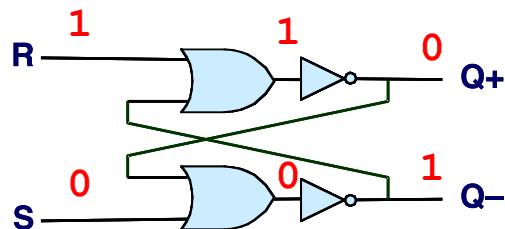
Elemento Bistabile



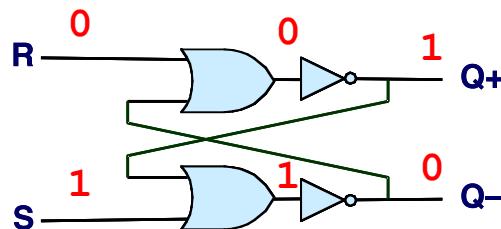
R-S Latch



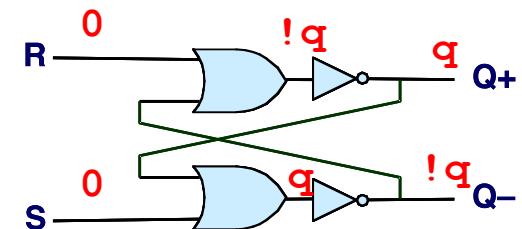
Reset



Set



Memorizzare



Stati indecidibili e temporizzazione

- Dato che i segnali non si propagano in tempo nullo, l'effetto del cambio di un ingresso si propaga in tempo finito sulle uscite
- Se le uscite sono reazionate questo può creare problemi di indecidibilità dello stato della rete logica (con memoria)
- Gli elementi di memoria sono quindi sempre temporizzati, cioè sono governati da un segnale speciale chiamato “clock”
- Un elemento base di memoria temporizzato viene normalmente indicato come “gated latch”

Ingresso di abilitazione

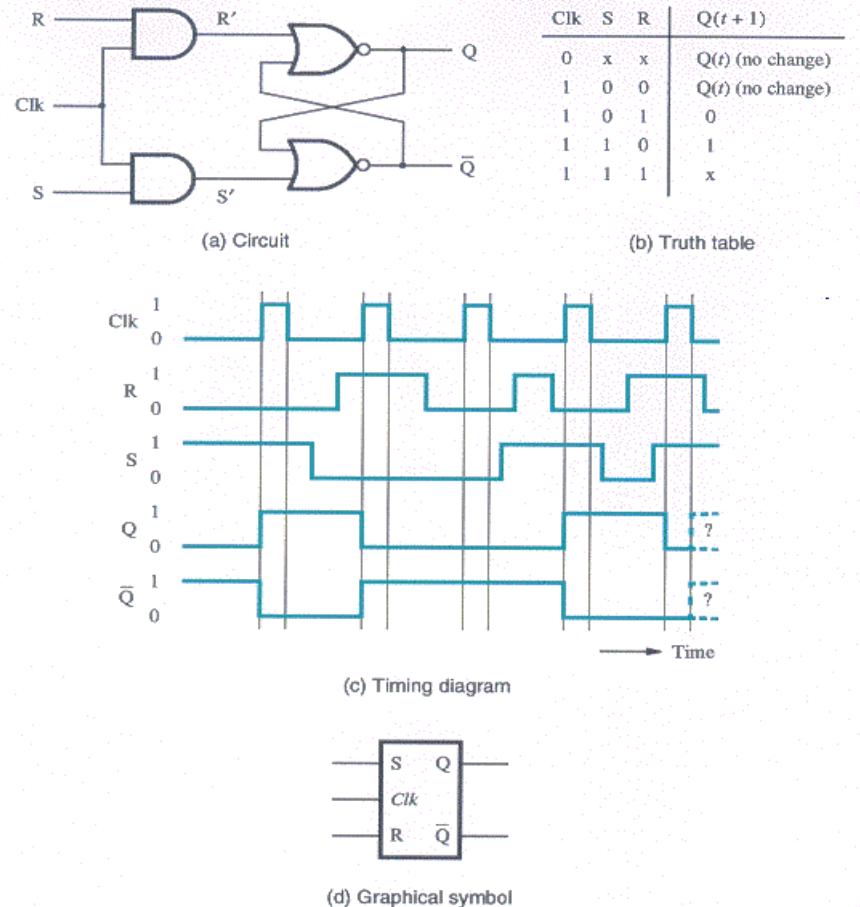


Figure A.25. Gated SR latch.

- Il clock viene inserito come “ingresso di abilitazione” attraverso porte AND: se Clk è a zero la rete reazionata ha gli ingressi forzati a zero e non può cambiare stato
- Quando Clk è a uno gli ingressi della rete reazionata sono gli ingressi R ed S del circuito
- Circuiti di questo tipo hanno rappresentazione grafiche “standard”

Elementi di memoria "reali": Latch tipo "D" e Flip-flop

- Le reti viste prima sono note come latch S-R (Set-Reset)
- Hanno il difetto di avere uno stato indecidibile (cioè l'uscita non può essere nota con certezza) quanto entrambi gli ingressi sono a uno
- In molti casi questo è inaccettabile
- **Si può rimediare?**
 - **latch-D (data)**
 - **flip-flop**

Latch tipo "D"

- Gli ingressi al circuito base sono ottenuti da un'unica variabile (di cui se ne fa il negato)
- Non vi può essere ambiguità, per definizione
- Il circuito è abilitato durante tutta la fase positiva del clock

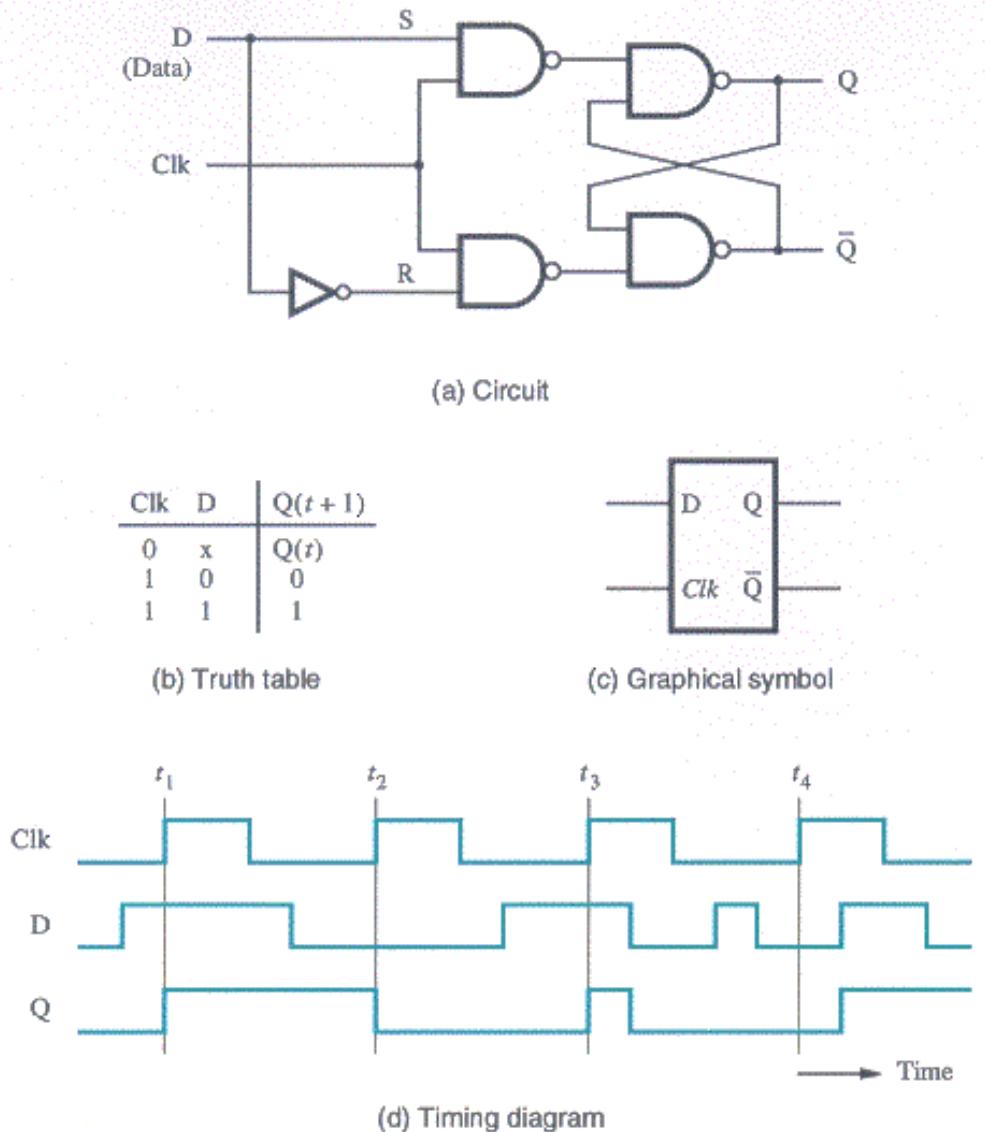
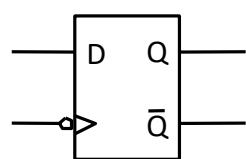
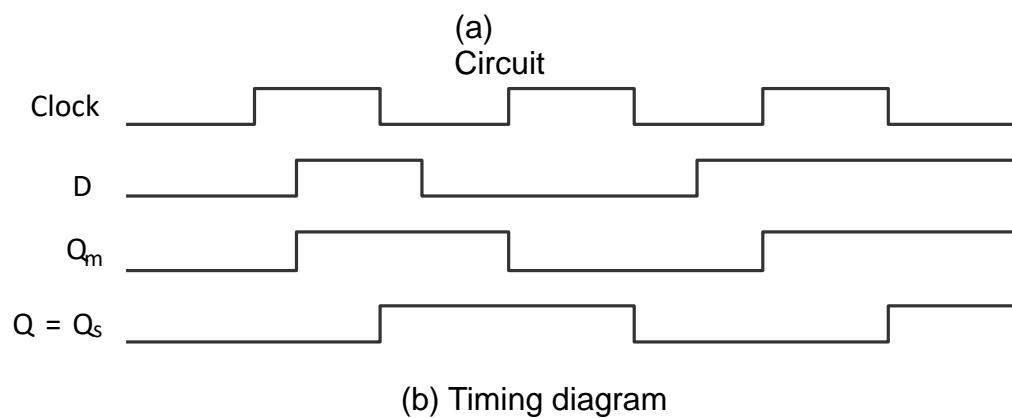
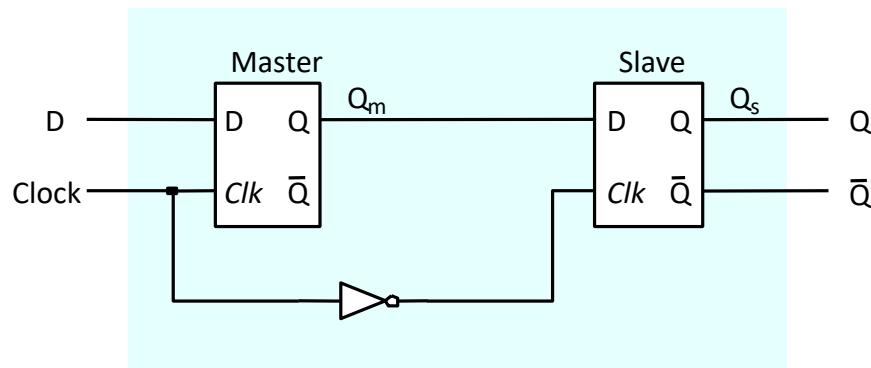


Figure A.27. Gated D latch.

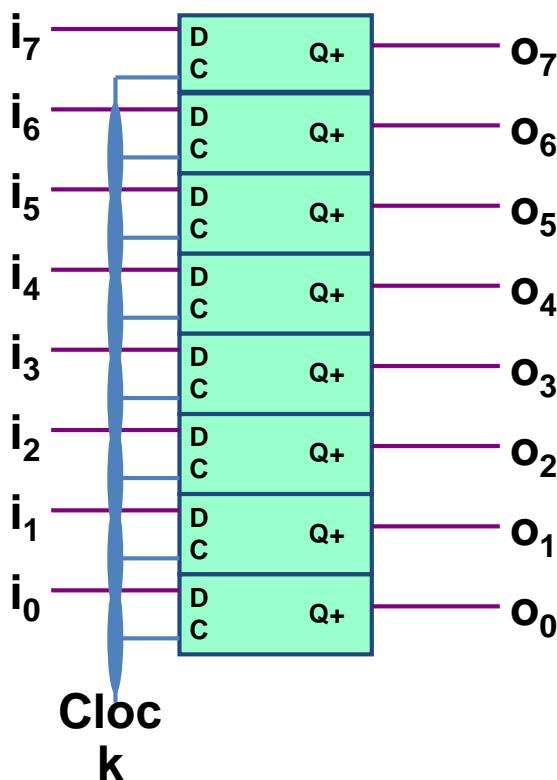
Flip-flop Master-Slave



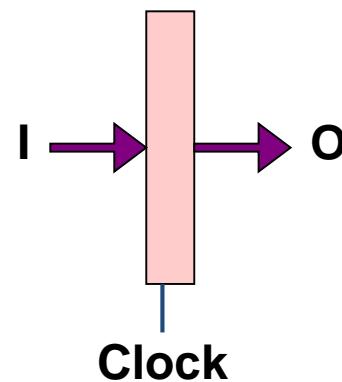
(c) Graphical
symbol

- Configurazioni più complesse (come questa) consentono ad esempio di ottenere che l'uscita del circuito commuti esattamente al termine dell'impulso di clock

Struttura

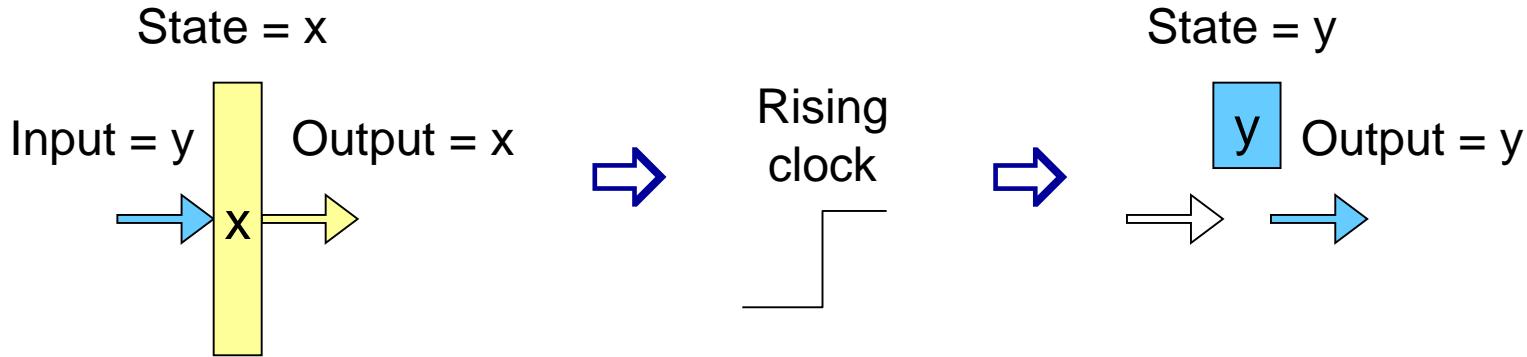


Registri



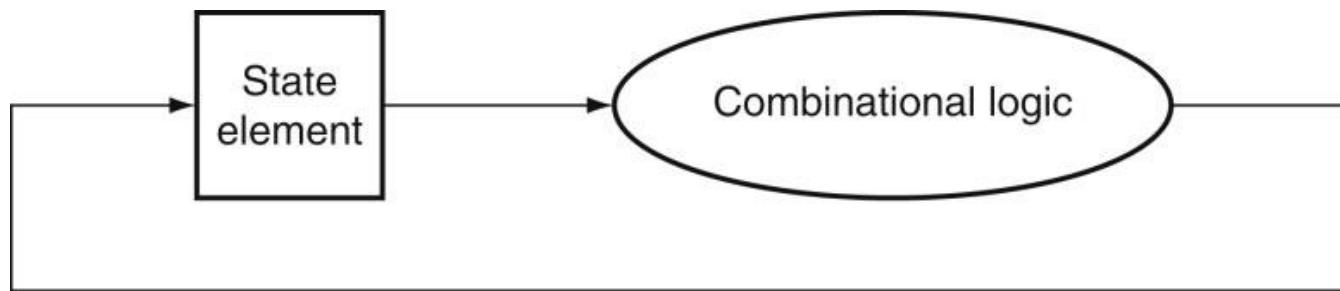
- Impiegati per registrare delle word di dati
- Collezione di latch edge-triggered
- Caricano gli input sul fronte in salita del clock

Operazioni su registri



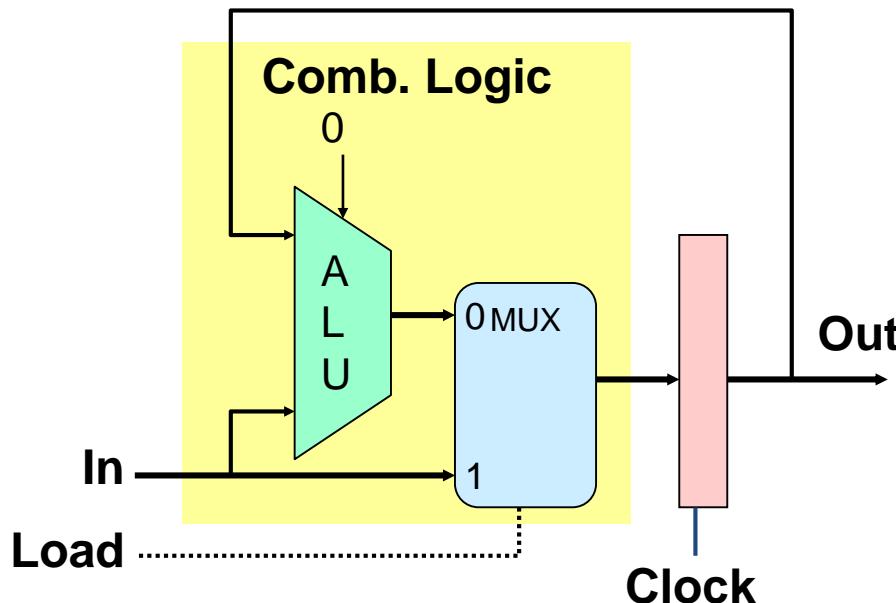
- Memorizzano bit
- La maggior parte delle volte operano come una barriera tra input e output
- Sul fronte in salita del clock memorizzano l'input

Vantaggi dell'edge triggered

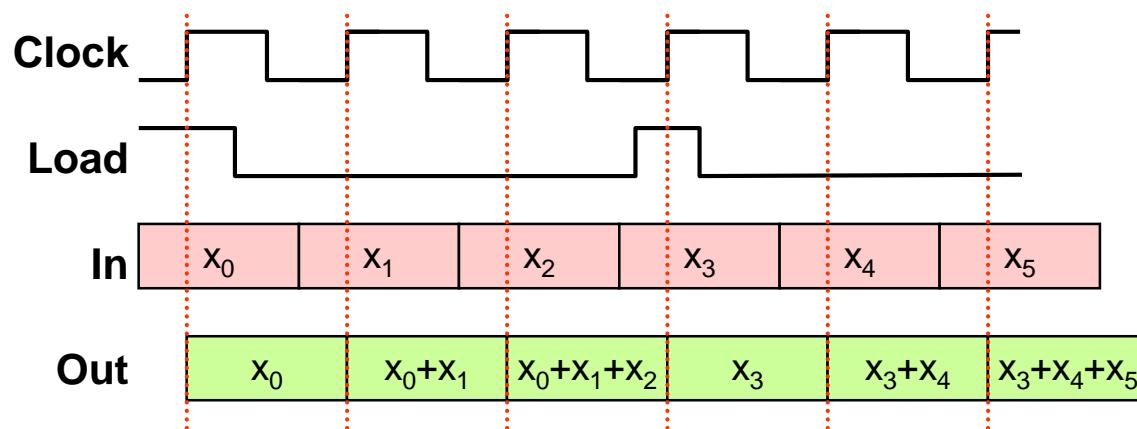


- Una metodologia edge triggered permette di aggiornare lo stato a partire dal quello presente senza creare delle situazioni di corse
- Questo porta alle macchine a stati in cui:
 - Lo stato successivo dipende da quello presente e dall'input
 - L'output dipende dallo stato presente e dall'input (macchina di Mealy), o solo dallo stato presente (macchina di Moore)

Esempio di macchina a stati

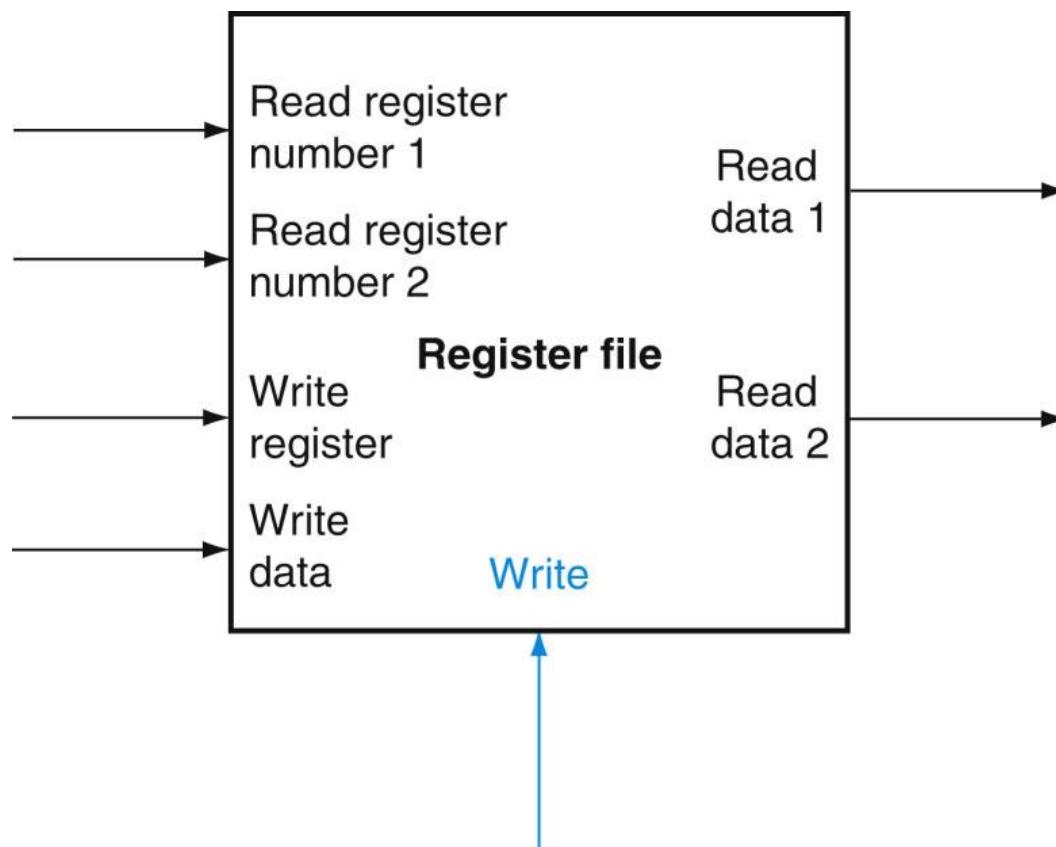


- Circuito accumulatore
- A ogni ciclo carica l'input e lo accumula



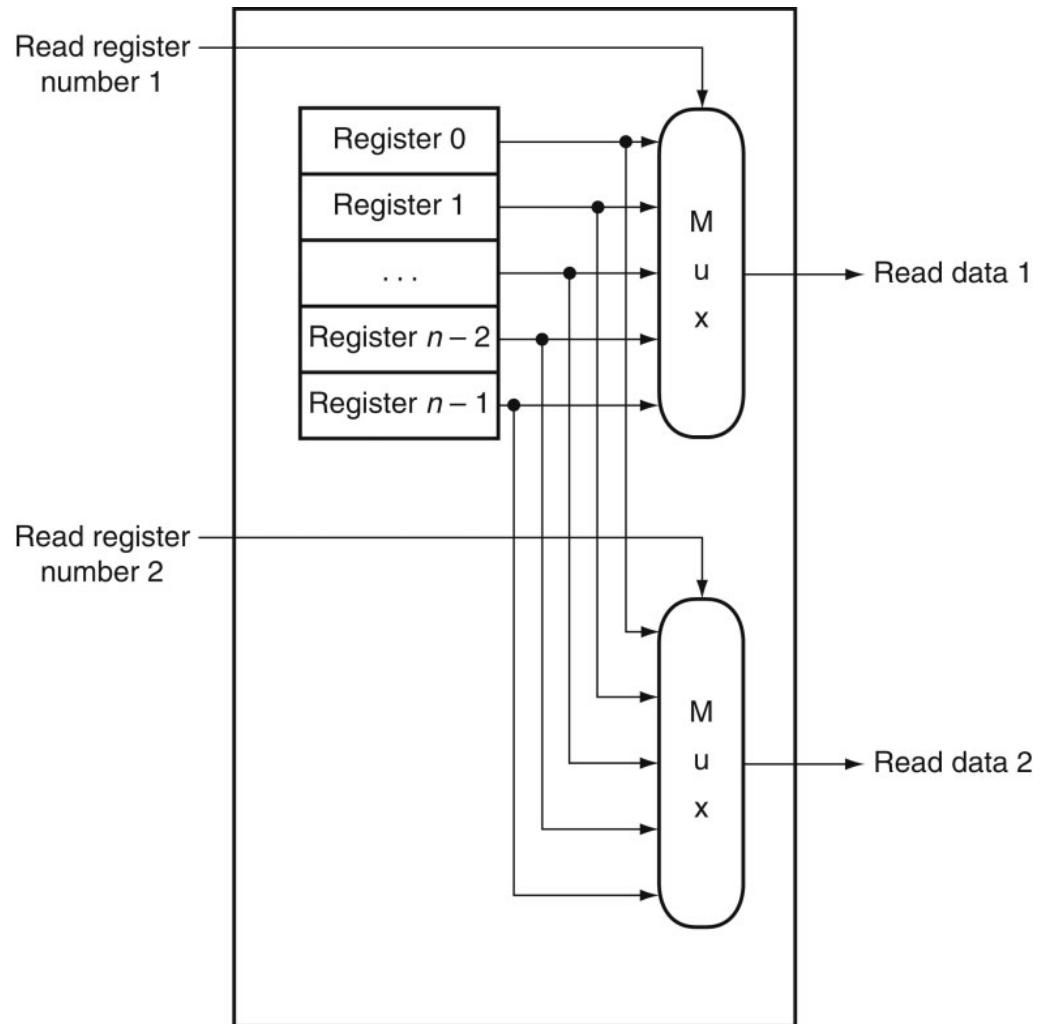
Register file

- I registri possono essere organizzati in un array (memoria) con la possibilità di scrivere o leggere su alcuni di essi



Register file -- lettura

- Possiamo leggere due registri. Per scegliere quali usiamo un MUX



Register file -- scrittura

- Possiamo scrivere in un registro. Per sceglier quale usiamo un decoder

