Corso di Architettura dei Sistemi a Microprocessore

Introduzione Alle Reti Logiche

Luigi Coppolino

Contact info

Prof. Luigi Coppolino
luigi.coppolino@uniparthenope.it

Università degli Studi di Napoli "Parthenope"
Dipartimento di Ingegneria

Centro Direzionale di Napoli, Isola C4
V Piano lato SUD - Stanza n. 512

Tel: +39-081-5476702
Fax: +39-081-5476777

Analisi e Sintesi di un sistema

Per *analisi* di un sistema si intende l'individuazione delle relazioni di causa/effetto tra i segnali di ingresso e uscita, attraverso l'esame di una rappresentazione schematica dei suoi componenti elementari e dei collegamenti che li interconnettono, ovvero:

- *data la rappresentazione schematica del sistema, individuarne il comportamento*.

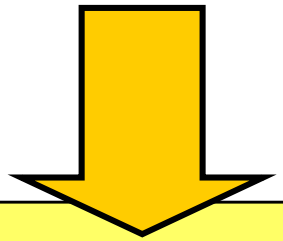
Per *sintesi* di un sistema si intende l'individuazione dei componenti e delle interconnessioni necessarie per realizzarlo seguendo la preassegnata specifica funzionale:

- *data la specifica funzionale individuarne la struttura.*

Analisi e sintesi

Analisi

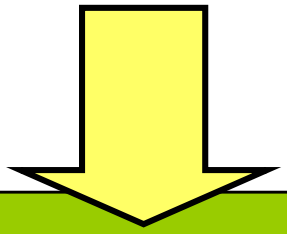
Data la descrizione
della
STRUTTURA
(come è fatta)



Determinarne il
COMPORTAMENTO
(cosa fa)

Sintesi

Data la descrizione
del
COMPORTAMENTO
(cosa deve fare)



Determinarne la
STRUTTURA
(come è fatta)

L'algebra di Boole - richiami

Operazioni fondamentali sui bit:

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

Congiunzione
x AND y si indica anche con: $x \cdot y$

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

Disgiunzione
x OR y si indica anche con $x + y$

x	NOT x
0	1
1	0

Negazione
NOT x si indica anche con $\overline{x}$

L'algebra di Boole - alcune proprietà (1)

- Proprietà commutativa:

  x AND y = y AND x x OR y = y OR x

- Proprietà associativa:

  (x AND y) AND z = x AND (y AND z)

  (x OR y) OR z = x OR (y OR z)

  per la propr. associativa si posso definire AND e OR a più di due operandi (es. x AND y AND z)

- Proprietà di idempotenza e assorbimento:

  x AND x = x x OR x = x

  x AND (x OR y) = x x OR (x AND y) = x

L'algebra di Boole - alcune proprietà (2)

- Proprietà distributiva
  x AND (y OR z) = (x AND y) OR (x AND z)
  x OR (y AND z) = (x OR y) AND (x OR z)

- Proprietà di convoluzione
  NOT (NOT x) = x

- Proprietà del minimo e del massimo:
  x AND 1 = x x AND 0 = 0
  x OR 0 = x x OR 1 = 1

- Leggi di De Morgan:
  NOT (x AND y) = (NOT x) OR (NOT y)
  NOT (x OR y) = (NOT x) AND (NOT y)

Funzioni booleane (o logiche)

$y = f(x_1, x_2, \ldots, x_n)$ è una funzione booleana se ad ogni ennupla di valori booleani $x_1,\ldots,x_n$ associa un valore booleano y

Due esempi:

x_1	x_2	$f(x_1,x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Questa funzione è detta OR esclusivo, o XOR

x_1	x_2	$f(x_1,x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

Questa funzione è detta equivalenza, o EQU

Anche AND, OR e NOT sono funzioni booleane. Esse vengono dette *funzioni fondamentali* dell'algebra

Insiemi funzionalmente completi

Si può dimostrare che qualsiasi funzione booleana può essere calcolata applicando le funzioni AND, OR, e NOT. Ad esempio:

x XOR y = (x AND NOT y) OR (y AND NOT x)

Per questo, l'insieme {AND, OR, NOT} si dice *funzionalmente completo*.

Esistono altri insiemi funzionalmente completi. Si noti che grazie alle leggi di De Morgan si può costruire la AND da {OR, NOT}, oppure la OR da {AND, NOT}. Quindi anche {AND, NOT} e {OR, NOT} sono insiemi funzionalmente completi.

Reti logiche

I valori booleani possono essere rappresentati da grandezze elettriche.
Ad esempio:

0 <=> tensione di 0 Volt
1 <=> tensione di +5 Volt

In tal caso le funzioni booleane possono essere realizzate mediante circuiti elettronici detti *reti logiche*.

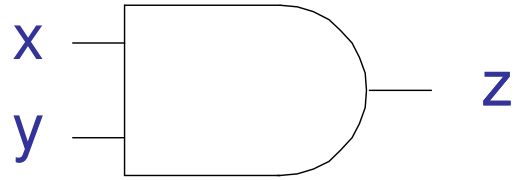
Nelle reti logiche *unilaterali*, le uscite della rete corrispondono a valori di grandezze elettriche misurate in opportuni punti del circuito; il flusso dell'elaborazione procede fisicamente in un'unica direzione, dai segnali di ingresso verso i segnali di uscita.

Nelle reti logiche *bilaterali*, invece, l'uscita della rete è determinata dalla presenza o dall'assenza di "contatto" tra due punti della rete.

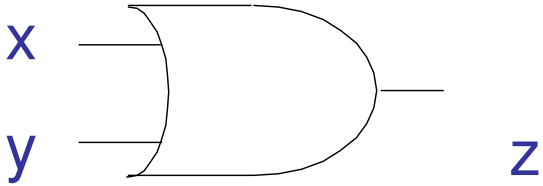
Porte logiche (gates)

Circuiti logici elementari che realizzano le operazioni fondamentali. Le reti logiche si costruiscono connettendo più porte logiche.

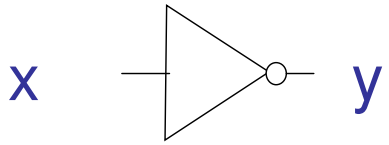
Simboli delle principali porte logiche:



z = x AND y



z = x OR y



y = NOT x

(se connesso a un'altra porta, il NOT si indica talora con un semplice pallino)

Operatori logici generalizzati (1)

Dato un vettore di variabili booleane $X = (x_1, x_2, \ldots, x_n)$, e una variabile booleana α, indicheremo con la notazione:

$Y = \alpha \; OP \; X$ (dove *OP* è un operatore booleano)

l'operazione che produce il vettore booleano Y così definito:

$Y = (y_1, y_2, \ldots, y_n)$ con

$y_1 = \alpha \; OP \; x_1$

. . . . . .

$y_n = \alpha \; OP \; x_n$

Un vettore di bit viene indicato anche come **parola** (**word**)

Esempio:

α AND X ha come risultato il vettore formato da:

(α AND x_1, α AND x_2, …, α AND x_n)

Operatori logici generalizzati (2)

Dati due vettori di variabili booleane $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_n)$ indicheremo con la notazione:

$Z = X \; OP \; Y$ (dove *OP* è un operatore booleano)

l'operazione che produce il vettore booleano Z così definito:

$Z = (z_1, z_2, \ldots, z_n)$ con

$z_1 = x_1 \; OP \; y_1$

. . . . . .

$z_n = x_n \; OP \; y_n$

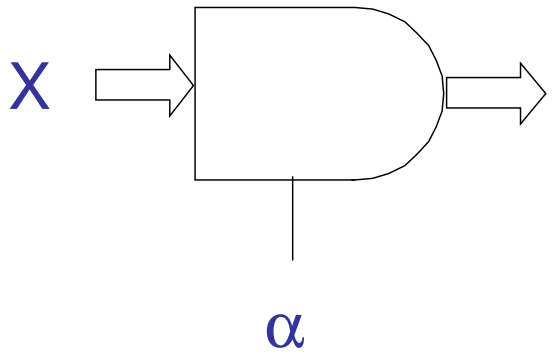
Esempio:

X OR Y ha come risultato il vettore formato da:

$(x_1 \text{ OR } y_1, x_2 \text{ OR } y_2, \ldots, x_n \text{ OR } y_n)$

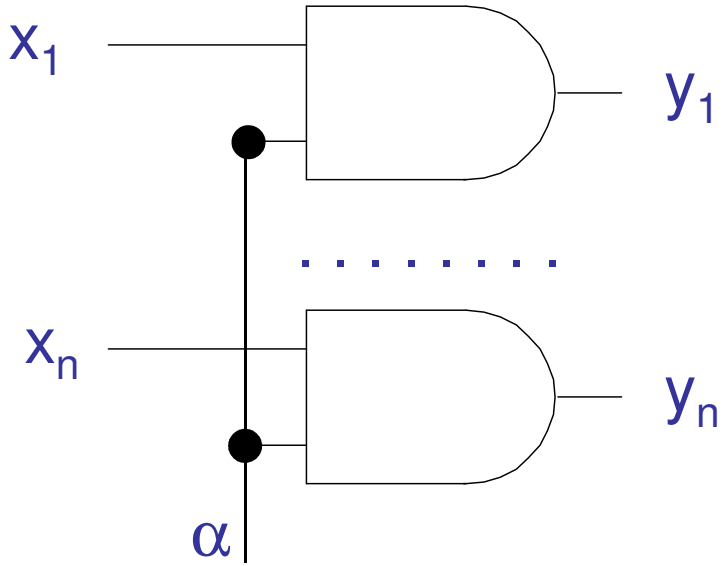
Porte logiche generalizzate (1)

Rappresentazione simbolica:



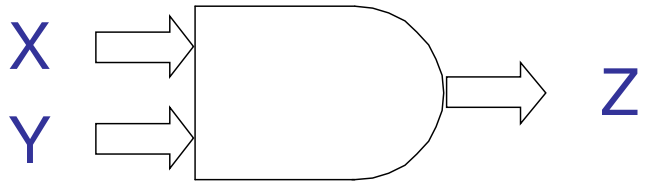
$Y = \alpha \text{ AND } X$

Circuito equivalente:



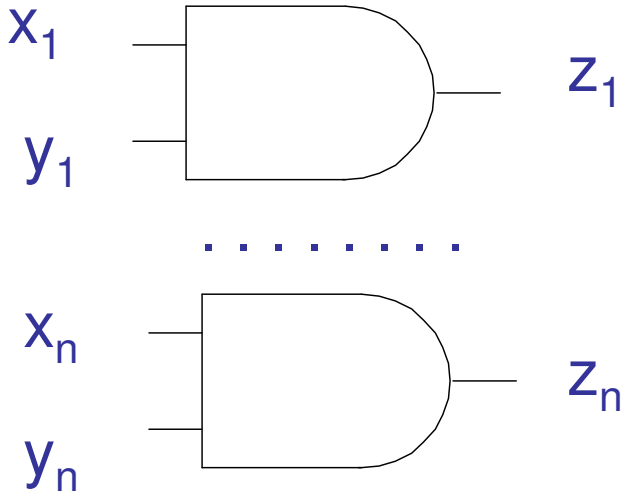
Porte logiche generalizzate (2)

Rappresentazione simbolica:



Z = X AND Y

Circuito equivalente:



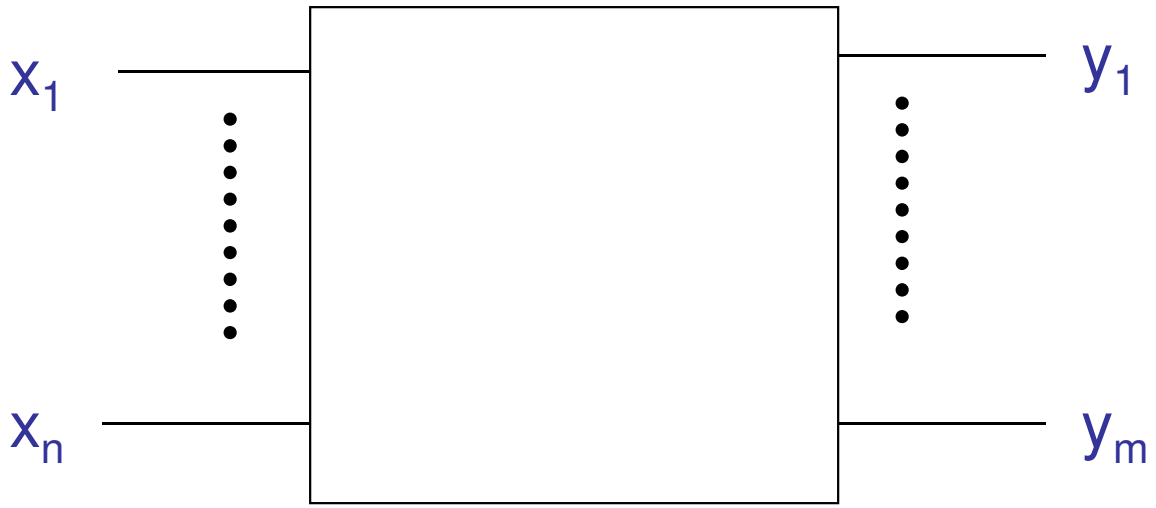
Tassonomia dei circuiti digitali

- I circuiti digitali possono essere classificati in due categorie
- Circuiti combinatori
  - Il valore delle uscite ad un determinato istante dipende unicamente dal valore degli ingressi in quello stesso istante.
- Circuiti sequenziali
  - Il valore delle uscite in un determinato istante dipende sia dal valore degli ingressi in quell'istante sia dal valore degli ingressi in istanti precedenti
  - Per definire il comportamento di un circuito sequenziale è necessario tenere conto della storia passata degli ingressi del circuito
- La definizione di circuito sequenziale implica due concetti:
  - Il concetto di tempo
  - Il concetto di stato

Macchine combinatorie (1)

Reti logiche con *n* ingressi $x_1, x_2, \ldots, x_n$ e *m* uscite $y_1, y_2, \ldots, y_m$ che realizzano la corrispondenza:

$$y_1 = f_1(x_1, x_2, \ldots, x_n)$$
$$\ldots\ldots$$
$$y_m = f_m(x_1, x_2, \ldots, x_n)$$



Macchine combinatorie (2)

In una macchina combinatoria i valori delle uscite dipendono esclusivamente dai valori degli ingressi

In una macchina combinatoria ideale tale dipendenza è istantanea

In una macchina reale c'è sempre un ritardo tra l'istante in cui c'è una variazione in uno degli ingressi e l'istante in cui l'effetto di questa variazione si manifesta sulle uscite

Sintesi di funzioni logiche

- Possiamo sintetizzare una funzione logica a partire dalla sua tabella di verità nella forma *somma di prodotti* o *prodotti di somme.*

x_1	x_2	x_3	f_1	f_2
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

- Somma di prodotti:
  - Individuare le righe della tabella di verità a valore 1
  - Per ognuna di esse, rappresentare un termine come il prodotto delle variabili in input, prese come x_i se $x_i = 1$, altrimenti (NOT x_i) se $x_i = 0$.
  - Sommare tutti i termini così ottenuti

Minimizzazione delle funzioni logiche

- Costo di una funzione logica: numero di gates + numero di input per ogni porta
- Minimizzare: rendere minimo il costo della rete logica associata ad una funzione logica
- Possiamo minimizzare effettuando operazioni algebriche, tenendo a mente le regole di logica binaria viste in precedenza:
  - Distributività, Idempotenza, Complemento, …

$$f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_2 \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot x_3$$

$$f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x_3} + x_1 \cdot x_2 \cdot x_3$$

$$f = \overline{x_1} \cdot \overline{x_2}(\overline{x_3} + x_3) + x_1 \cdot \overline{x_2}(\overline{x_3} + x_3) + \overline{x_1} \cdot \overline{x_3}(\overline{x_2} + x_2)$$

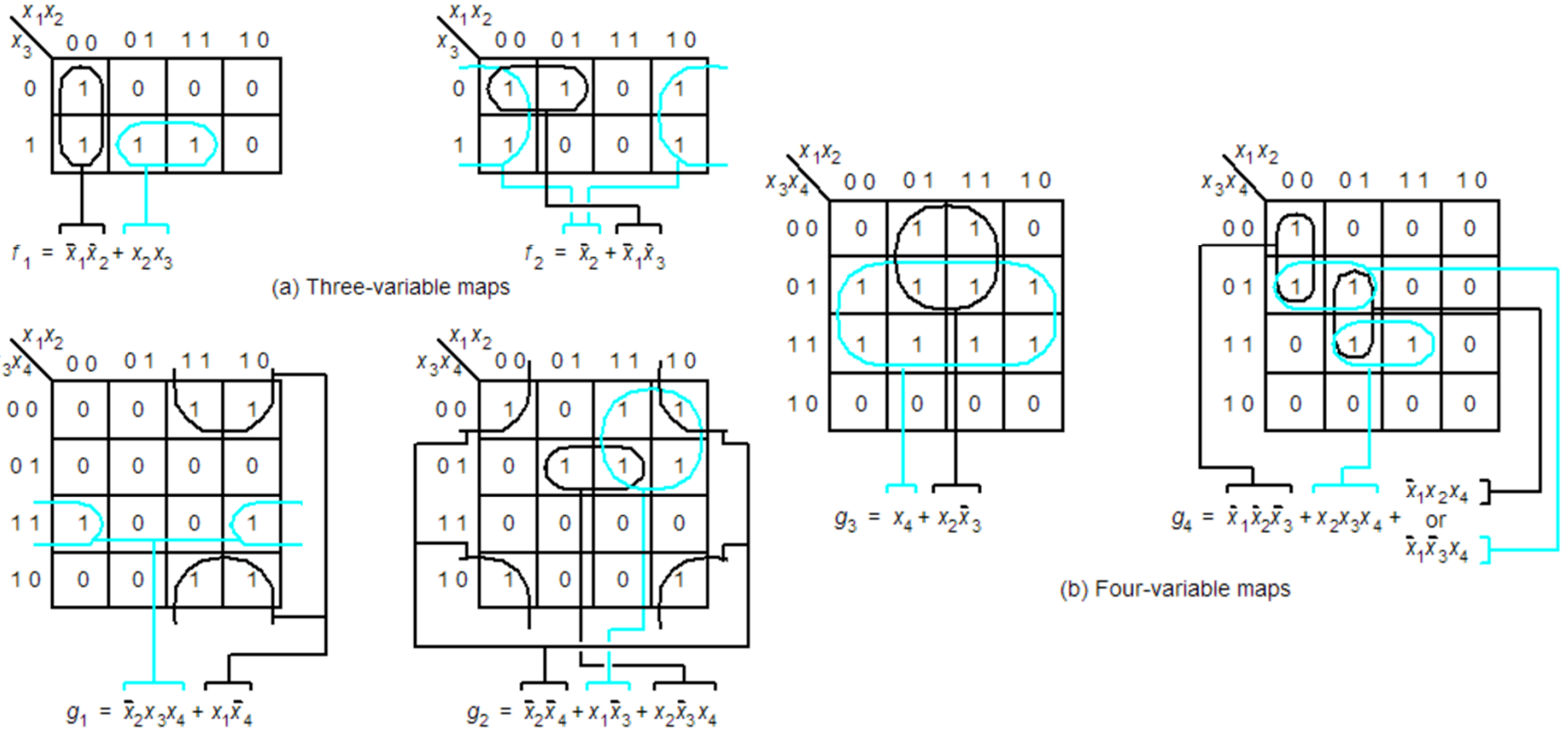
$$= \overline{x_1} \cdot \overline{x_2} + x_1 \cdot \overline{x_2} + \overline{x_1} \cdot \overline{x_3} = \overline{x_2} + \overline{x_1} \cdot \overline{x_3}$$

Minimizzazione con le mappe di Karnaugh (1/2)

- Tecnica geometrica per derivare rapidamente la rappresentazione minima di una funzione logica
- Si riportano su due assi le variabili in input (con una variazione alla volta) e si indica all'interno dei quadrati della mappa il valore della funzione logica corrispondente
- La funzione logica può essere agevolmente rappresentata fino a 4 variabili di input (cioè basta una sola mappa di Karnaugh piana)
- Si raccolgono i quadrati in gruppi di 2^k e si riportano i termini prodotto, indicando le sole variabili che non cambiano.

Minimizzazione con le mappe di Karnaugh (2/2)

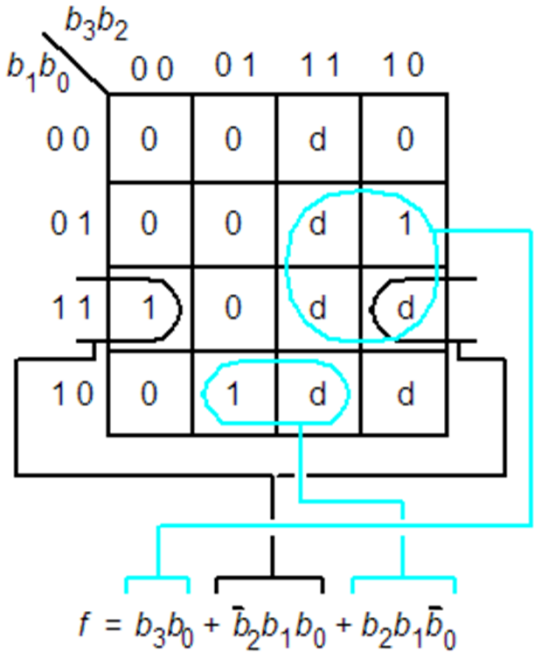
- Esempi:



Variabili Don't Care

- *Don't Care*: variabili che possono assumere indifferentemente valore 0 o 1. Sono indicate con d nella tavola di verità.
- Esempio: BCD a 4 bit

Decimal digit represented	Binary coding b_3 b_2 b_1 b_0	f
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	0
6	0 1 1 0	1
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	1
unused	1 0 1 0	d
	1 0 1 1	d
	1 1 0 0	d
	1 1 0 1	d
	1 1 1 0	d
	1 1 1 1	d



Esercizio

- Trovare l'espressione delle seguenti funzioni logiche nella forma somma di prodotti
- Minimizzarla mediante tavola di Karnaugh
- Verificare il risultato ottenuto con LogiSim

x_1	x_2	x_3	f_1	f_2	f_3	f_4
0	0	0	1	1	d	0
0	0	1	1	1	1	1
0	1	0	0	1	0	1
0	1	1	0	1	1	d
1	0	0	1	0	d	d
1	0	1	0	0	0	d
1	1	0	1	0	1	1
1	1	1	1	1	1	0

Figure PA.1 Logic functions for Problem A.

NAND e NOR gates

- Implementazione elettronica più vantaggiosa

x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	0

$f = x_1 \uparrow x_2 = \overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

(a) NAND

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	0

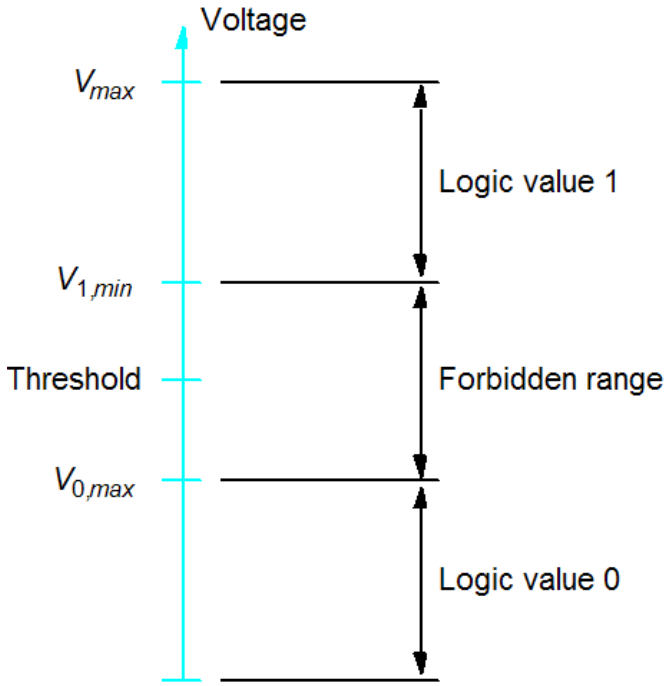
$f = x_1 \downarrow x_2 = \overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

(b) NOR

- Una NAND (o NOR) è un insieme funzionalmente completo
  - Posso rappresentare una funzione somma di prodotti con sole NAND
- Non è applicabile la proprietà associativa

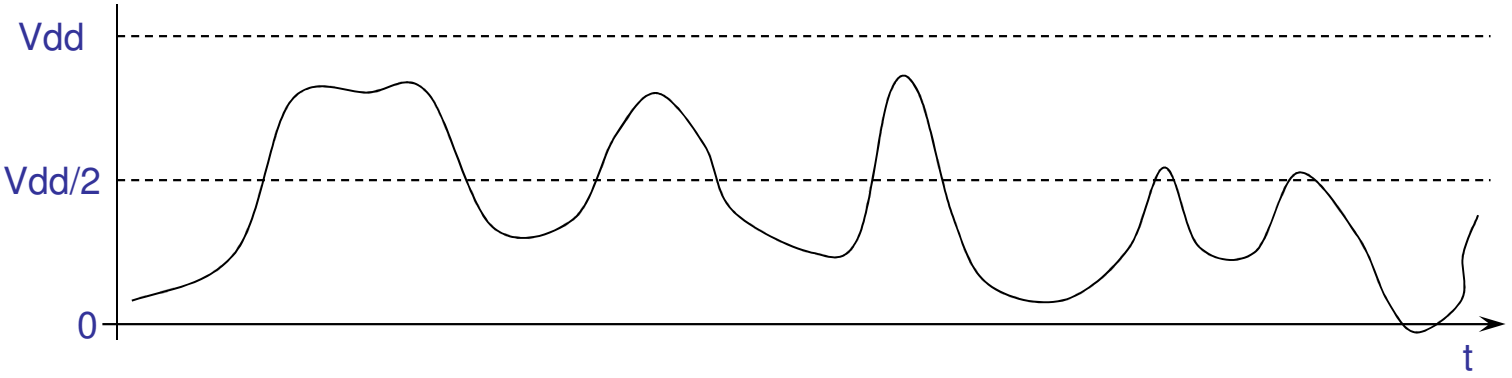
Implementazione reale delle porte logiche

- I valori logici sono rappresentati mediante correnti o tensioni
  - I due valori sono separati da una soglia (threshold)
  - In realtà vi è una regione interdetta ai valori logici (forbidden range) per contemplare eventuali disturbi elettrici (noise)

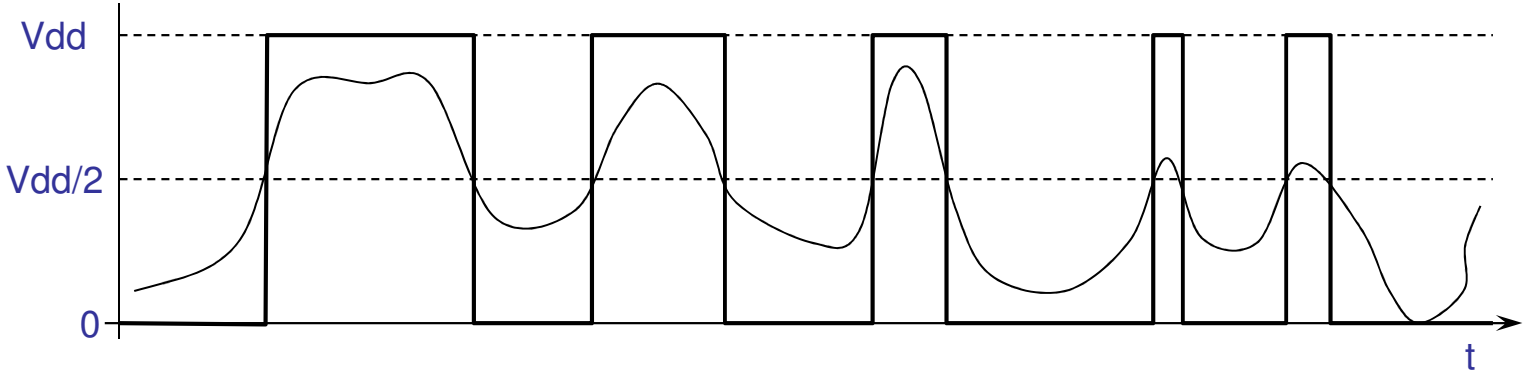


Il concetto di tempo

- Un segnale elettrico è una tensione variabile nel tempo

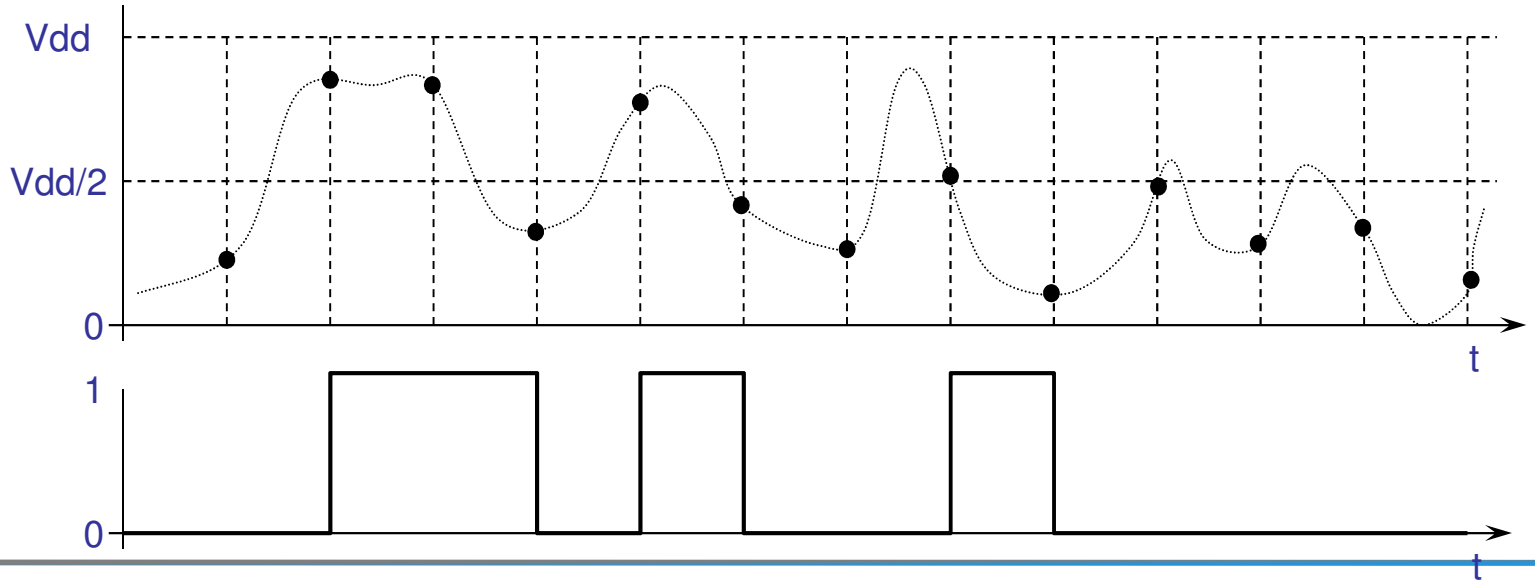


- I segnali binari sono rappresentati tipicamente mediante due livelli di tensione di un segnale elettrico.



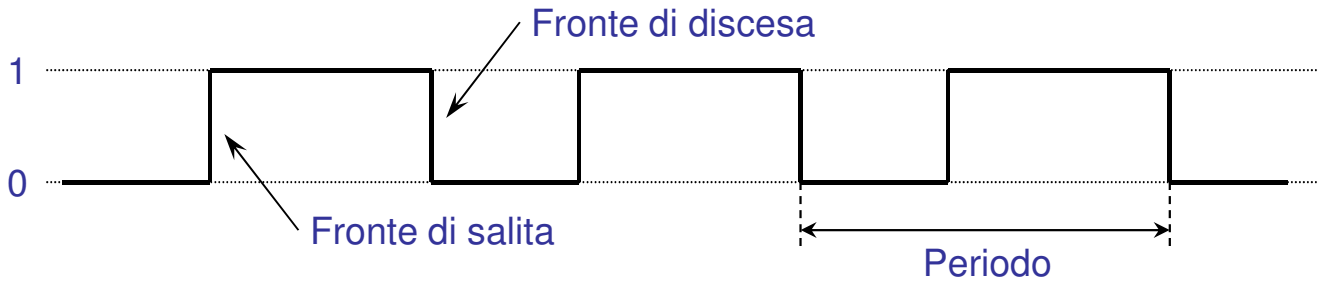
Il concetto di tempo

- Il segnale binario è un segnale variabile con continuità
- In un intervallo di tempo $t=t_1-t_0$ il segnale assume infiniti valori, corrispondenti agli infiniti istanti tra t_0 e t_1
- Si ricorre al concetto di *tempo discreto* in cui il numero di istanti discreti in un intervallo $t=t_1-t_0$ è finito



Il concetto di tempo

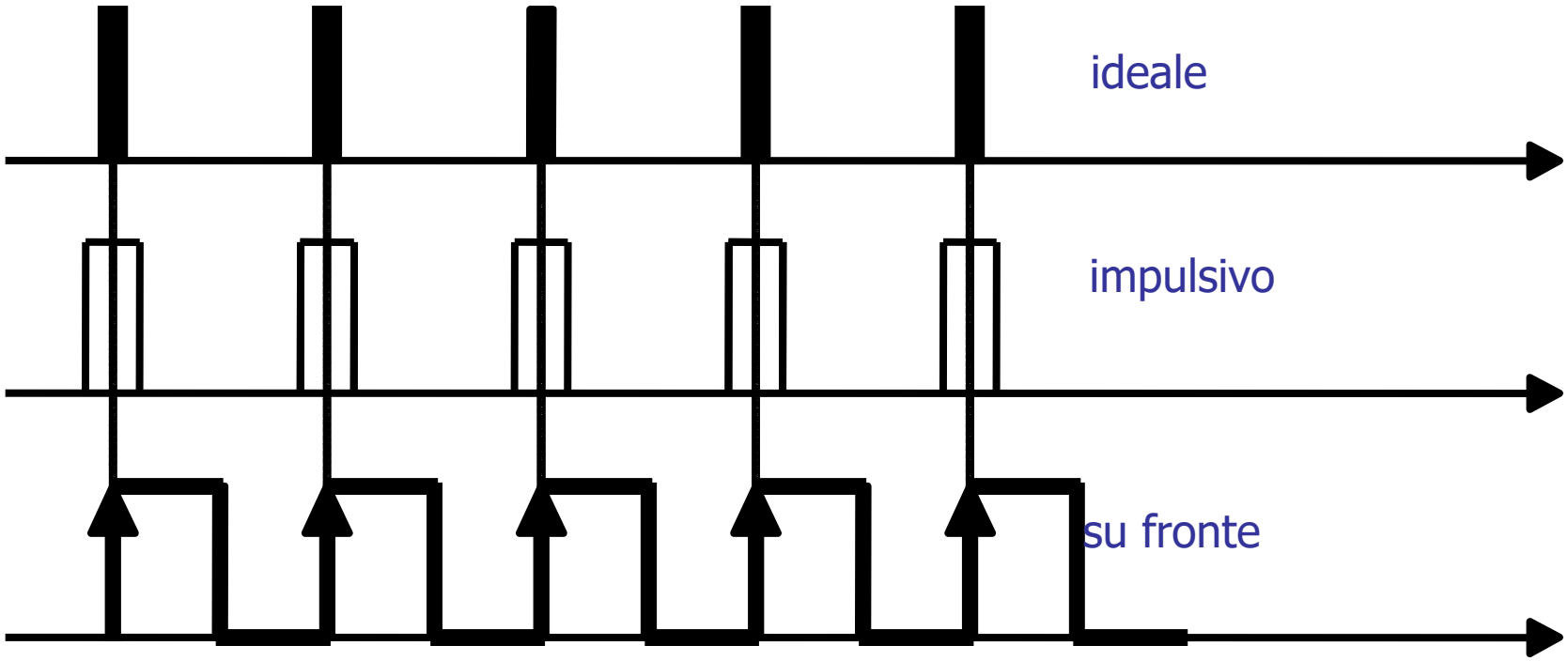
- Il valore del segnale elettrico viene letto o *campionato* in istanti determinati
- Gli istanti in cui deve essere *campionato* il segnale elettrico sono scanditi da un apposito segnale detto *clock*
- Un *clock* ha le seguenti caratteristiche:
  - E' un segnale binario
  - E' un segnale periodico



Il concetto di tempo

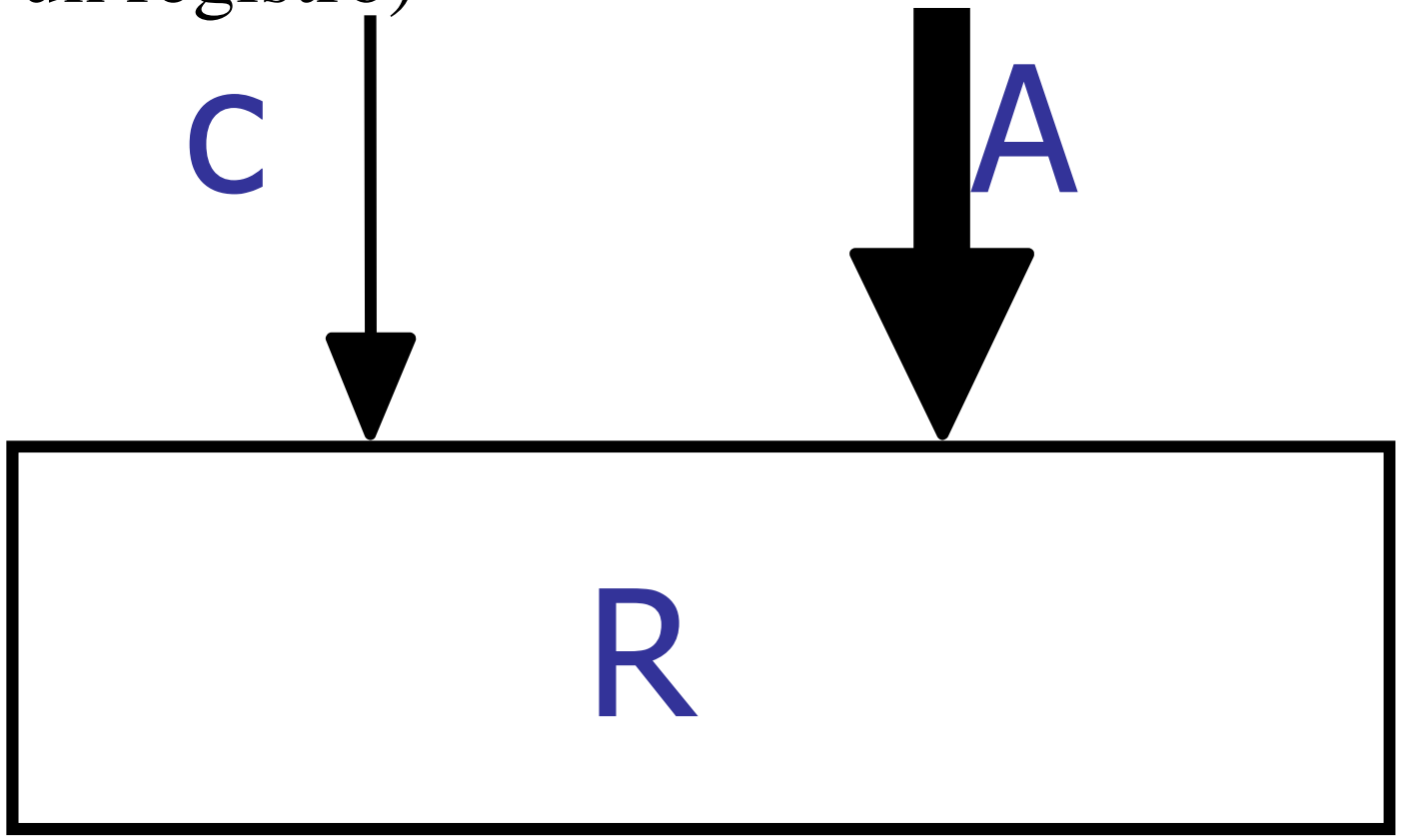
- Nel periodo T_{CK}, o *ciclo di clock*, il segnale assume:
  - Il valore logico 1 per un tempo T_H
  - Il valore logico 0 per un tempo T_L
- Il rapporto T_H / T_{CK} è detto *duty-cycle*
- Il passaggio dal valore 0 al valore 1 è detto *fronte di salita*
- Il passaggio dal valore 1 al valore 0 è detto *fronte di discesa*

Clock



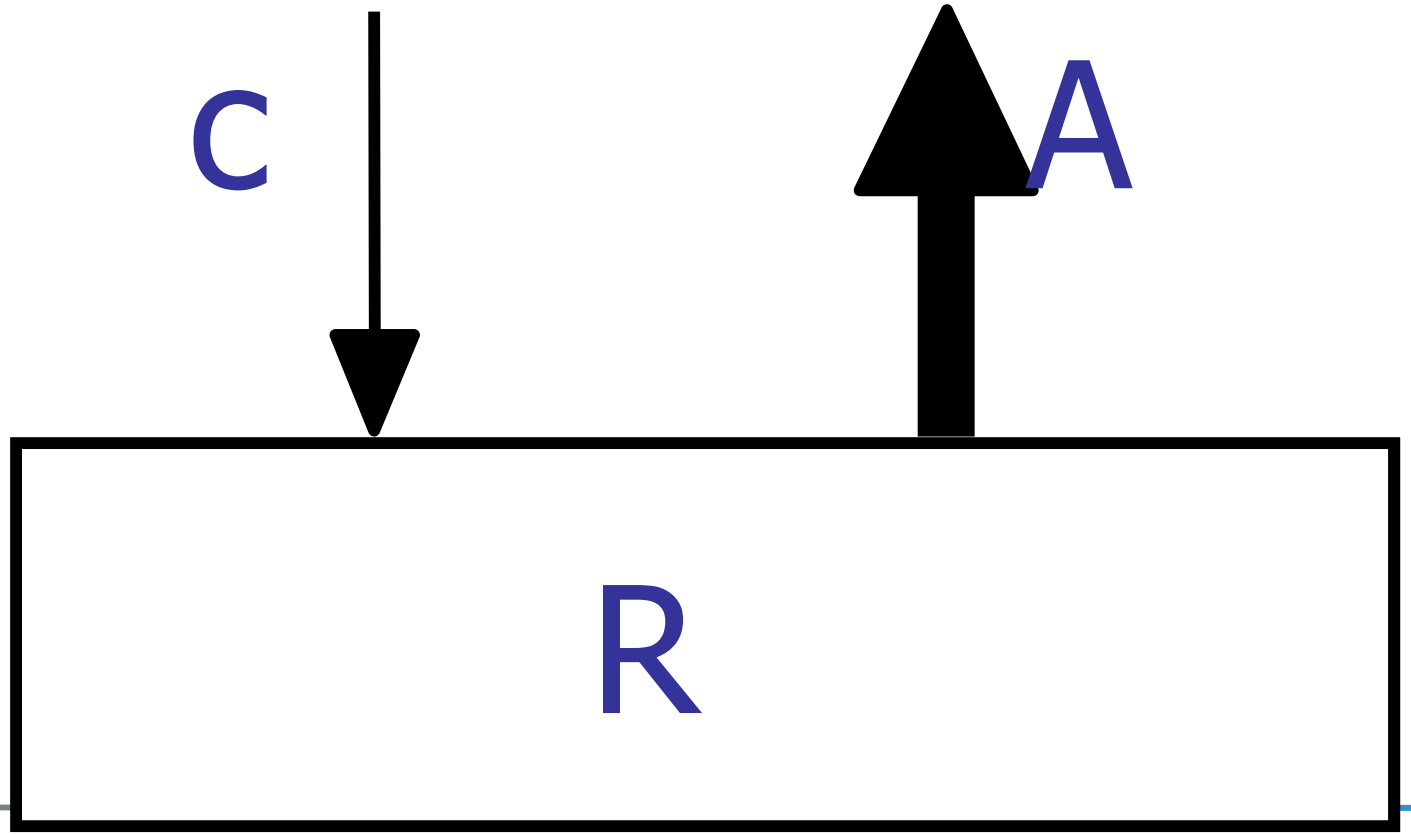
Trasferimento dati su bus unico -1/2

- Trasferimento da bus a registro (caricamento di un registro)

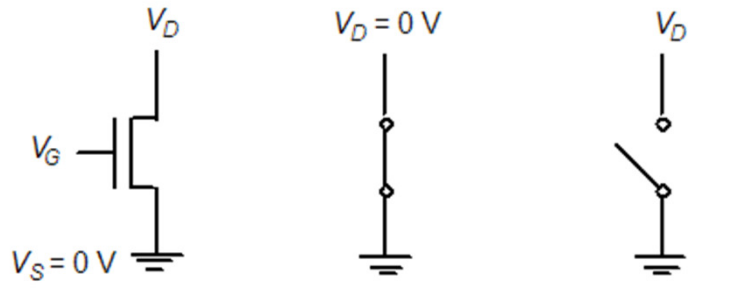


Trasferimento dati su bus unico -2/2

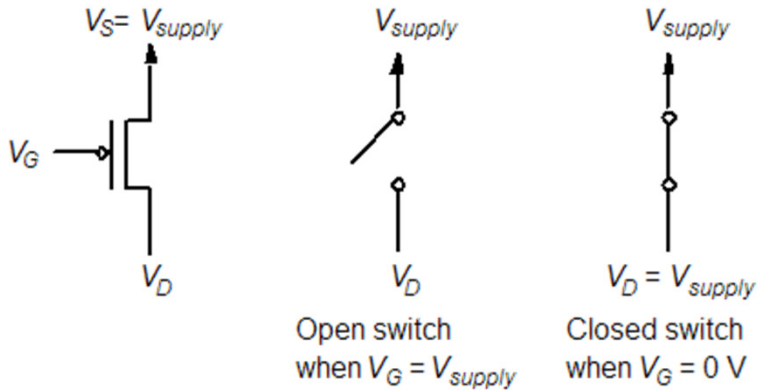
- Trasferimento da registro a bus (caricamento da un registro)



Transistor NMOS e PMOS

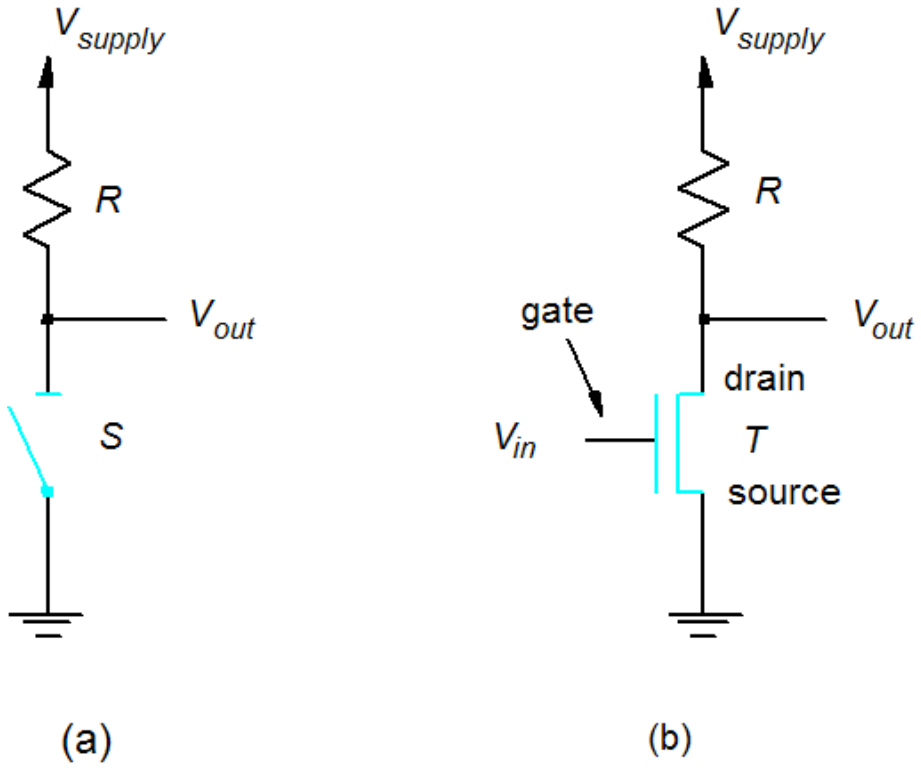


(a) NMOS transistor



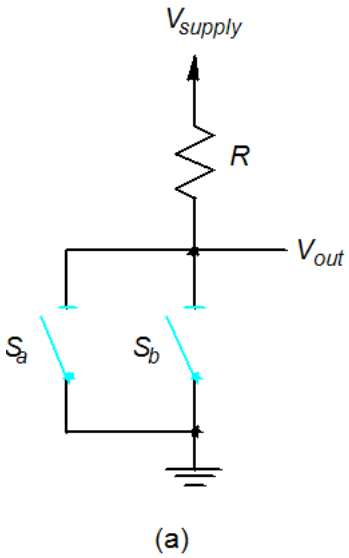
(b) PMOS transistor

Circuito Invertitore

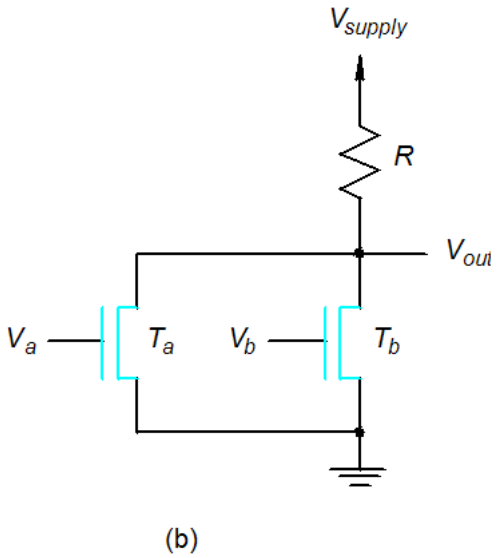


Se Vin=0 allora Vout=Vsupply
Se Vin=Vsupply allora Vout=0

Circuiti per NAND e NOR

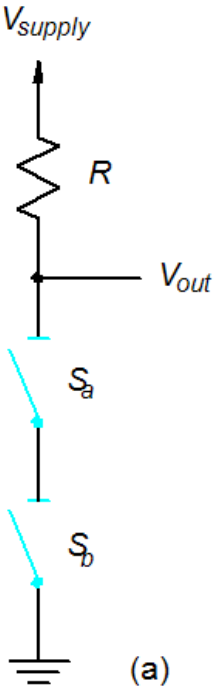


(a)

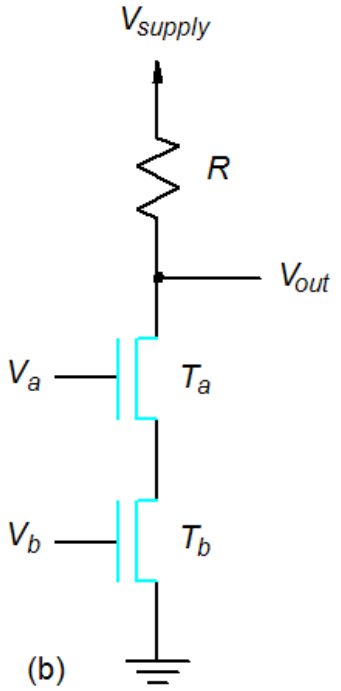


(b)

NOR



(a)

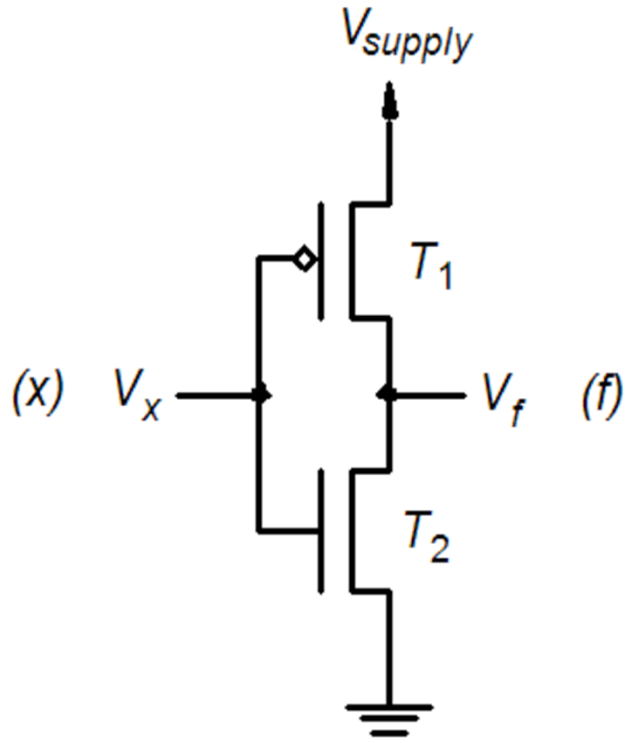


(b)

NAND

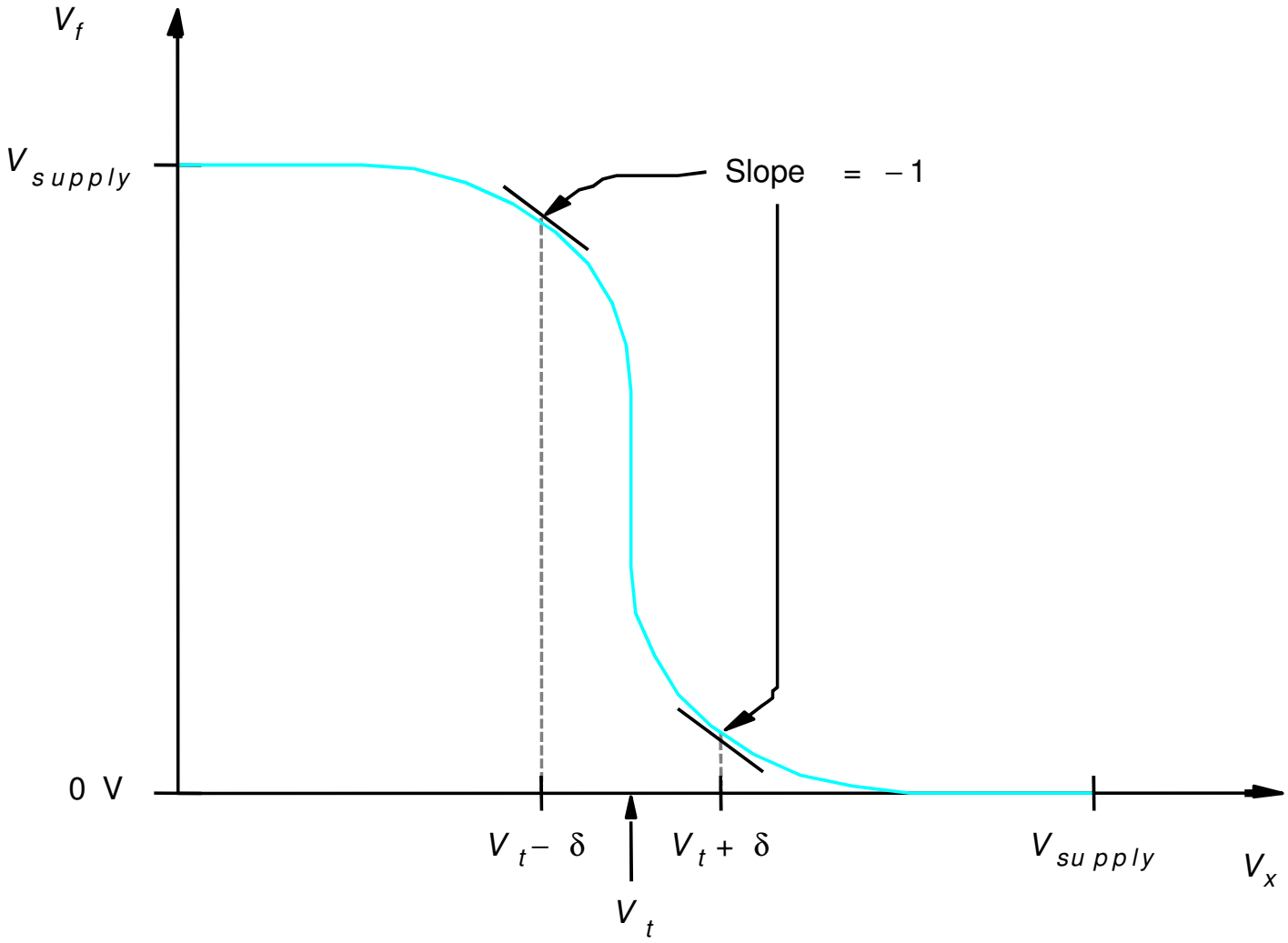
Invertitore CMOS

Problema energetico: quando il percorso verso terra è chiuso, la corrente passa attraverso il resistore, che dissipa energia sotto forma di calore.
Soluzione: sostituire il resistore con un transistor PMOS. In questo modo non c'è mai un percorso chiuso per la corrente verso terra, tranne in fase di transizione degli stati (nota: la potenza dissipata dipende dal numero di transizioni di stato in questo caso)



x	V_x	T_1	T_2	V_f	f
0	low	on	off	high	1
1	high	off	on	low	0

Caratteristica di trasferimento del CMOS



Circuiti CMOS

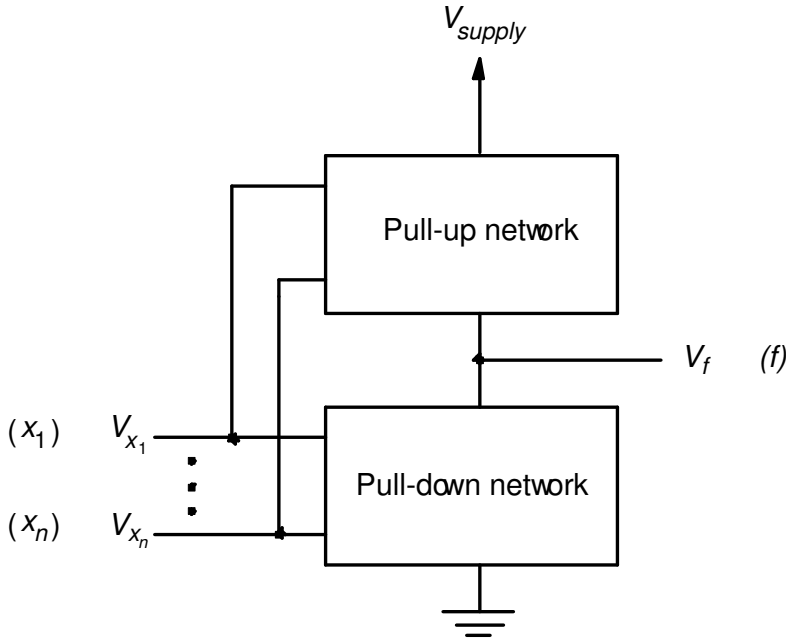
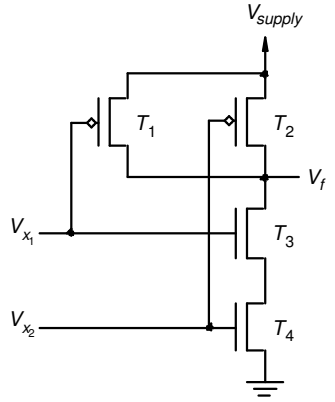


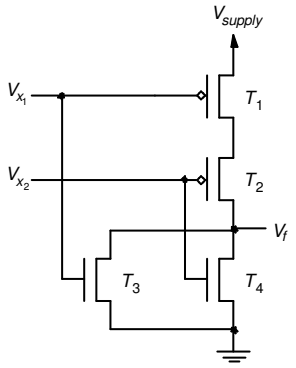
Figure A.16. Structure of a CMOS circuit.



(a) Circuit

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

(b) Truth table and transistor states



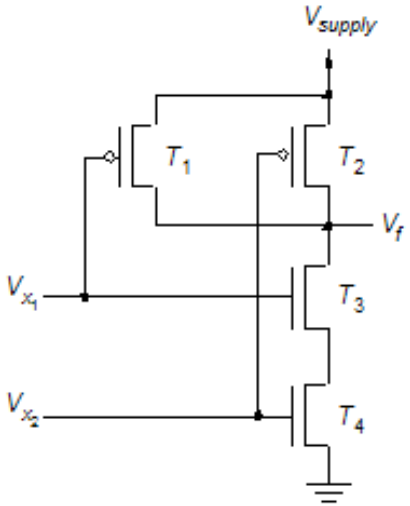
(a) Circuit

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

(b) Truth table and transistor states

Figure A.18. CMOS realization of a NOR gate.

NAND vs AND

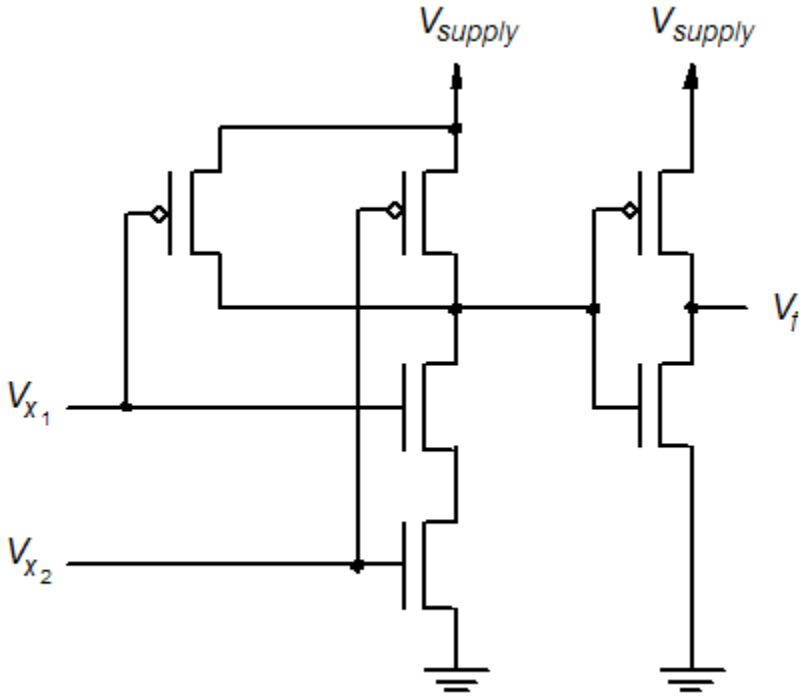


(a) Circuit

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

(b) Truth table and transistor states

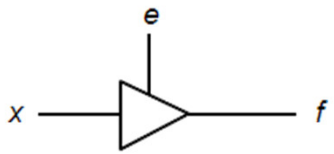
NAND



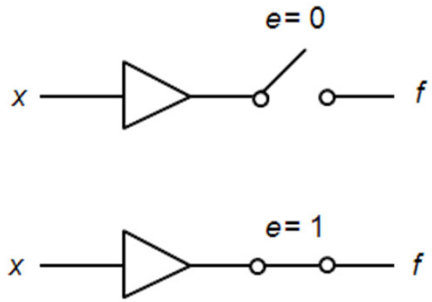
AND

Buffer a tre stati

- Si introduce un terzo stato (high-impedance) per disconnettere elettricamente un dispositivo dal circuito
- È la tecnica più diffusa per realizzare il collegamento di più registri "sorgenti" verso un bus comune



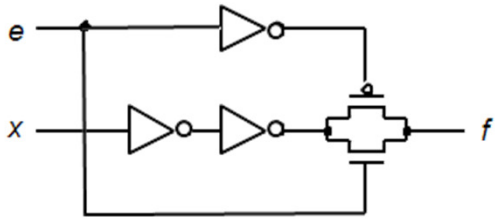
(a) Symbol



(b) Equivalent circuit

e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

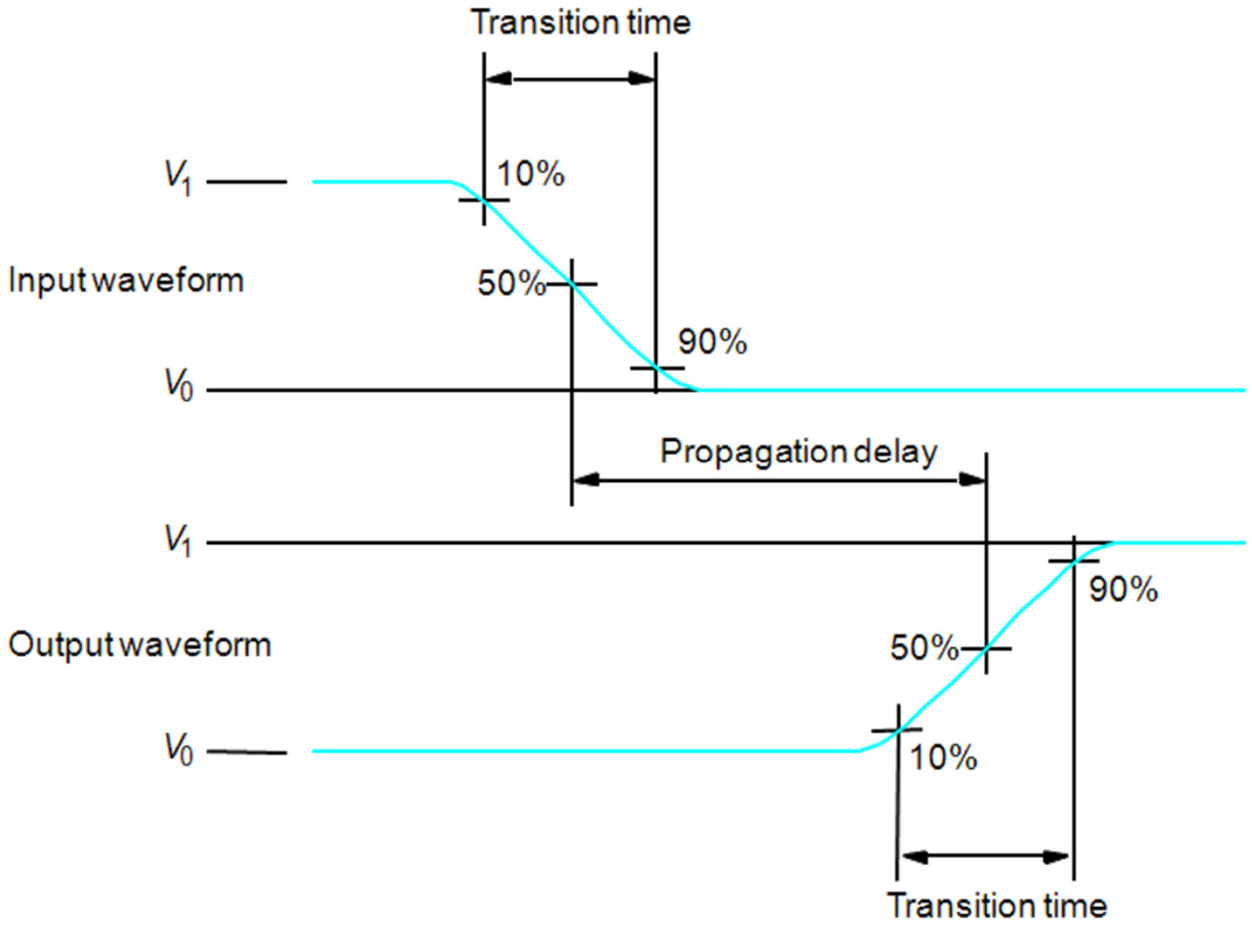
(c) Truth table



(d) Implementation

Ritardo di propagazione

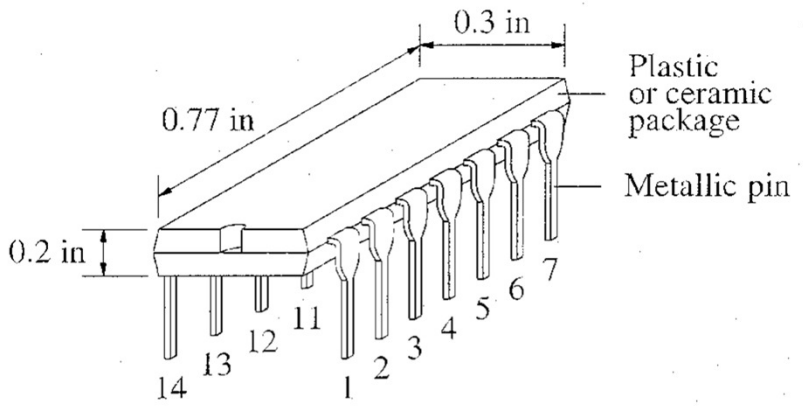
Maggiore è il ritardo di propagazione, minore è la velocità massima del circuito logico



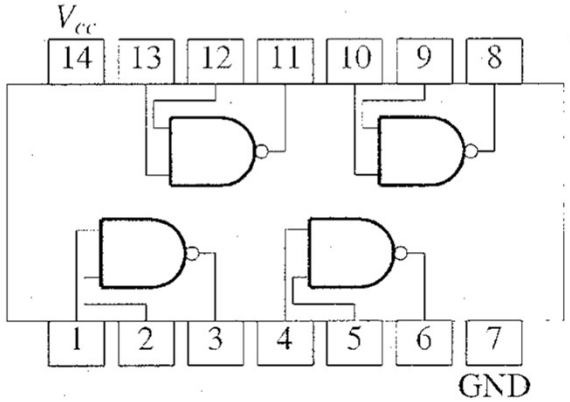
Fan-In e Fan-Out

- Fan-In: numero di ingressi alle porte logiche di un circuito
  - Maggiore è il Fan-In, maggiore è il ritardo di propagazione del circuito logico
  - Nel caso di CMOS un nuovo ingresso implica un nuovo PMOS e un nuovo NMOS
- Fan-Out: numero di uscite che pilotano ulteriori ingressi
- Solitamente il Fan-In e il Fan-Out sono in numero limitato per non far decrescere la velocità del circuito.
  - Per successivi ingressi si adopera un nuovo gate

Package di Circuiti Integrati (IC)



(a) Physical appearance



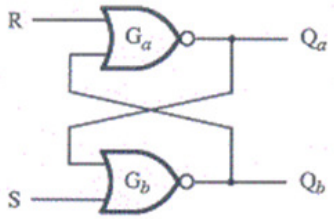
(b) Schematic of an integrated circuit providing four 2-input NAND gates

Elementi Bistabili

- Come detto, lo stato di un circuito deve essere memorizzato. A tale scopo si utilizzano degli elementi di memoria detti *bistabili.*
- Il termine *bistabili* deriva dal fatto che tali elementi possono assumere solo due valori: 0 e 1
- Esistono diversi tipi di bistabili che differiscono per il numero di ingressi e per il comportamento
- I flip-flop vengono utilizzati sia come elemento fondamentale per la costituzione dei registri sia come elemento ausiliario per la costruzione di macchine sequenziali più complesse.

SR latch

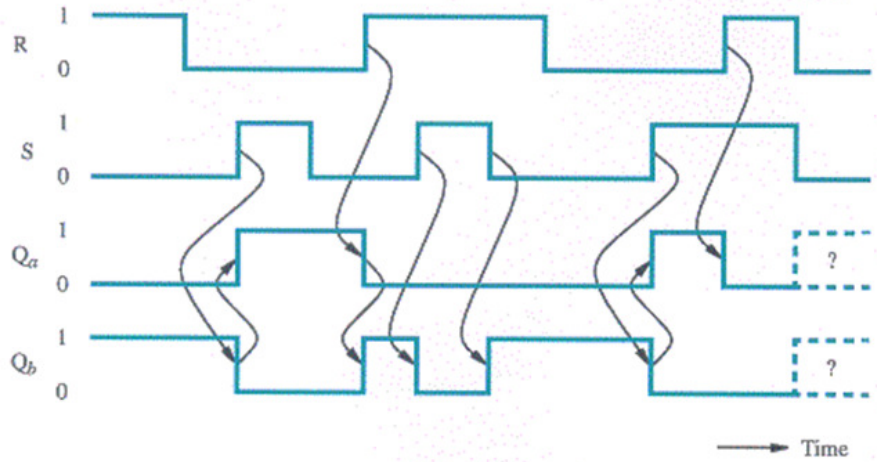
- Un *latch* è un dispositivo per memorizzare un dato binario.
- E' un circuito sequenziale



(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	

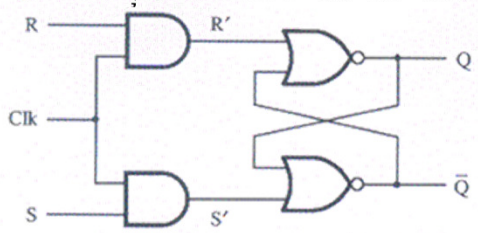
(b) Truth table



(c) Timing diagram

Gated SR latch

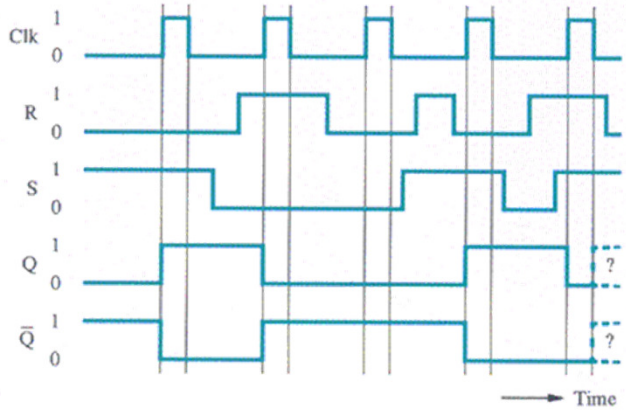
- Si introduce un input di controllo detto Clock (Clk)
  - Se Clk=1, il circuito segue il comportamento dettato da S e R. Se Clk=0 il circuito non cambia stato.
  - In presenza di abilitazione si parla di *gated latch*



(a) Circuit

Clk	S	R	Q(t + 1)
0	x	x	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Truth table



(c) Timing diagram

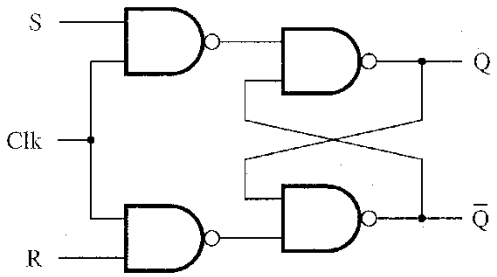
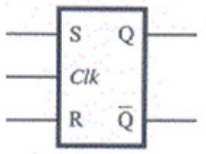


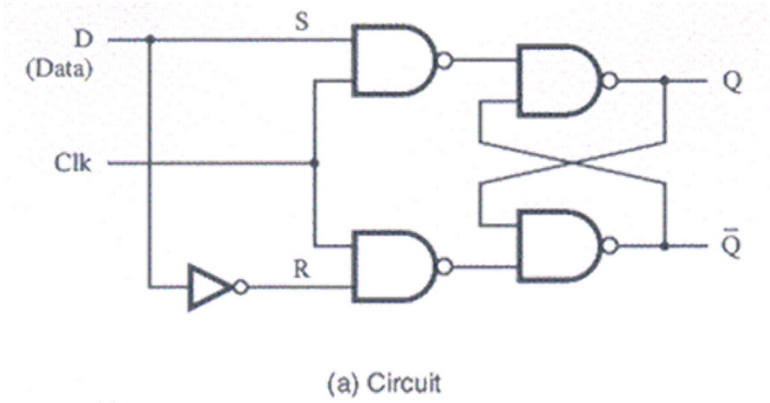
Figure A.26. Gated SR latch implemented with NAND gates.



(d) Graphical symbol

Gated D latch

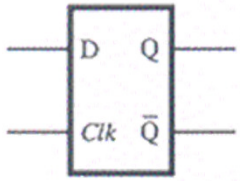
- S ed R sono comandati da un solo input, detto D (più il Clock).
- S ed R non varranno mai contemporaneamente 0,0 o 1,1
- L'uscita segue sempre il valore di D (se Clk=1).



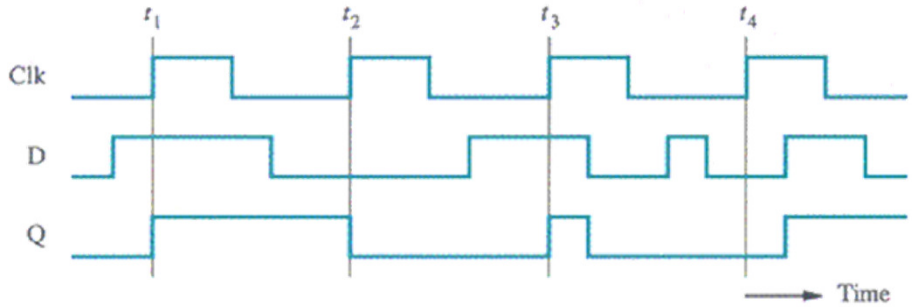
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Truth table



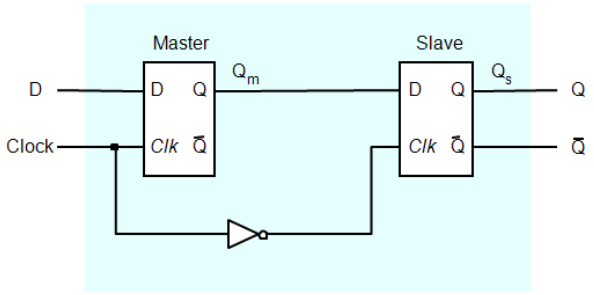
(c) Graphical symbol



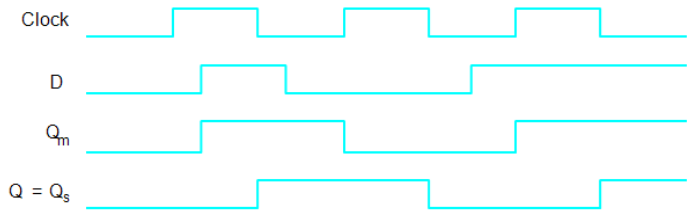
(d) Timing diagram

Flip-Flop D Master-Slave

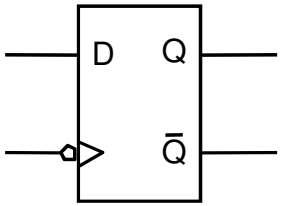
- Flip-Flop: elemento di memoria che cambia stato (cioè si abilita) sul fronte di un clock
- Vogliamo introdurre un ritardo tra l'arrivo di un nuovo input e l'uscita del circuito -> Circuito Master-Slave
- Se Clk=1 il Master segue D e lo Slave non cambia stato. Se Clk=0, Q=D.
- L'uscita cambia sul fronte di discesa del clock (fronte negativo) -> Inverto il clock (fronte positvo)



(a) Circuit



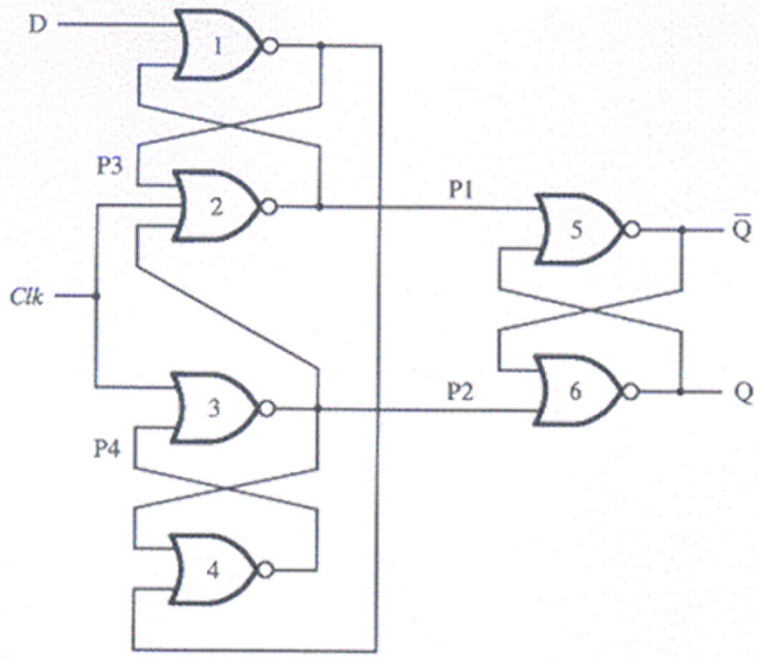
(b) Timing diagram



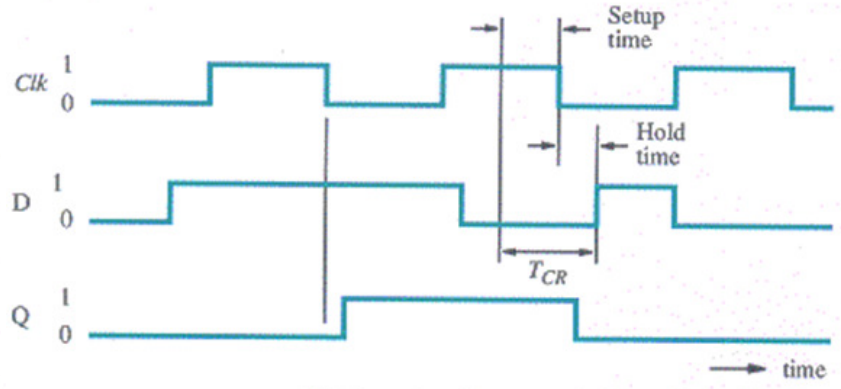
(c) Graphical symbol

Flip-Flop D edge triggered

- I Flip-Flop cambiano stato solo durante le transizioni del clock (fronte positivo o negativo). Il tempo di transizione deve essere breve.
- La tempistica deve essere ben definita. Tcritical = Hold Time + Setup Time
- Tcr è l'intervallo in cui D non deve cambiare il suo valore



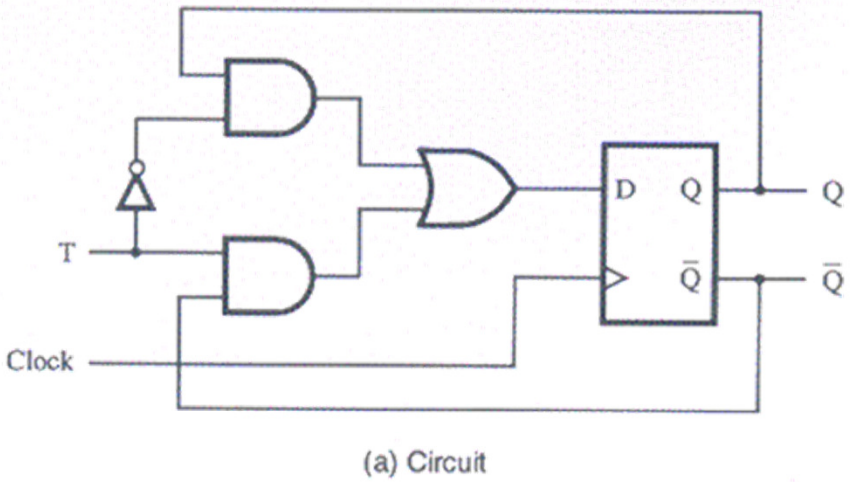
(a) Network



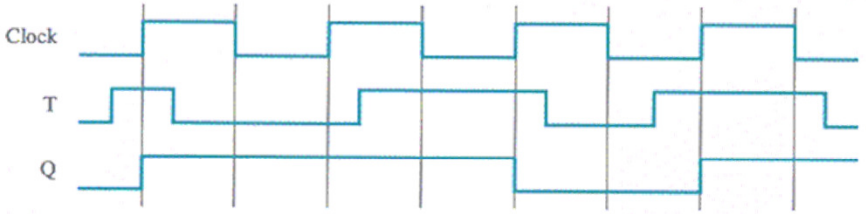
(b) Example of timing

Flip-Flop T

- Se Clk=1 e T=1, il flip-flop cambia stato.
- E' usato per i circuiti contatori



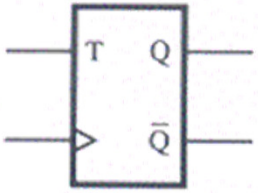
(a) Circuit



(d) Timing diagram

T	Q(t + 1)
0	Q(t)
1	$\bar{Q}(t)$

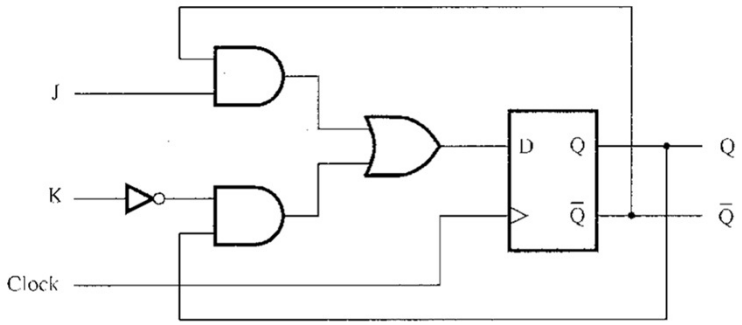
(b) Truth table



(c) Graphical symbol

Flip-Flop JK

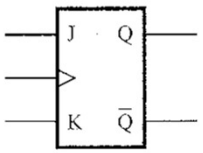
- Ha un comportamento simile all'SR e al T e quindi può essere utilizzato sia come memoria, sia per i counter



(a) Circuit

J	K	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Truth table

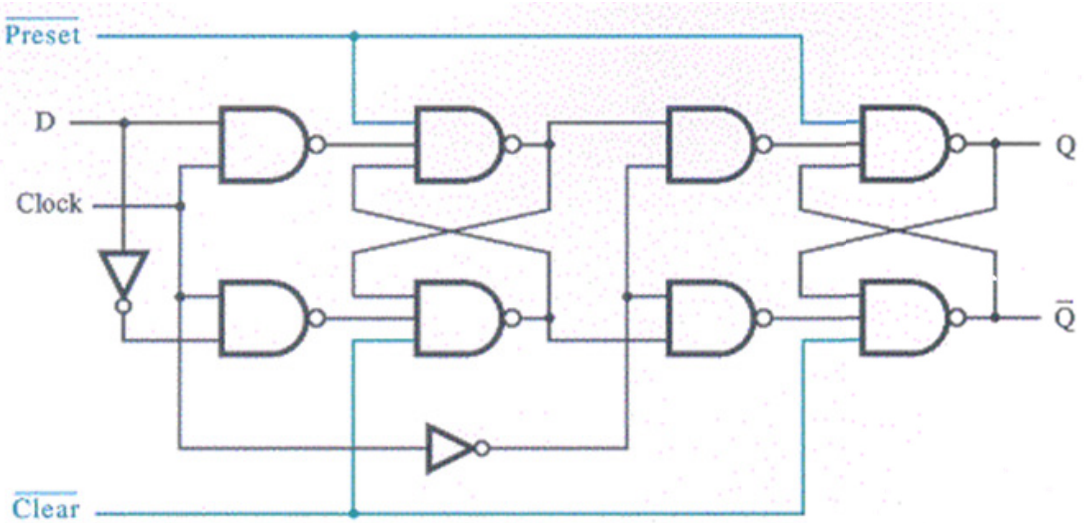


(c) Graphical symbol

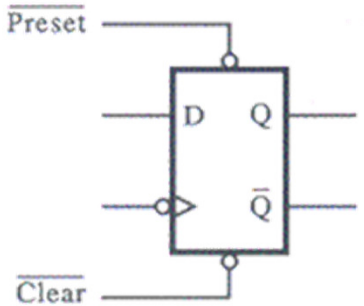
Figure A.31. JK flip-flop.

Flip-Flop con Preset e Clear

- Due input aggiuntivi che forzano lo stato del Flip-Flop a 0 o 1
  - Es. all'accensione del computer tutti i f-f dovrebbero essere a 0
  - Se P,C =1 il f-f è controllato da Clk e D. Se P=0,Q=1. Se C=0,Q=0.



(a) Circuit

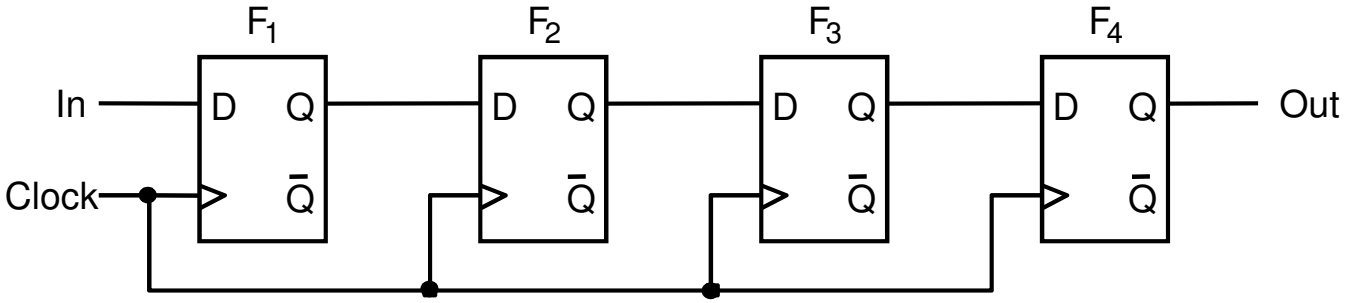


(b) Graphical symbol

Figure A.32. Master-slave D flip-flop with *Preset* and *Clear*.

Registri e Registri a scorrimento

- Memorizziamo una parola in un'unica struttura: registro
  - Letture e scritture sincronizzate in contemporanea dallo stesso Clk
- Realizziamo lo scorrimento di un bit alla volta (a dx o sx) con una catena di f-f
  - Se l'out va in input effettuiamo la rotazione del registro



Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

• Bisognerebbe avere un solo shift a clock: con i soli f-f D latch non controllo il numero di shift, che dipendono dalla durata del clock e dal delay di ogni f-f
->Usiamo le configurazioni Master Slave o Edge Triggered

Registro Seriale/Parallelo con Flip-Flop D

- Nella configurazione parallela:
  - se Load=0, l'input è Serial e lo shift è seriale
  - se Load=1, la lettura/scrittura è parallela

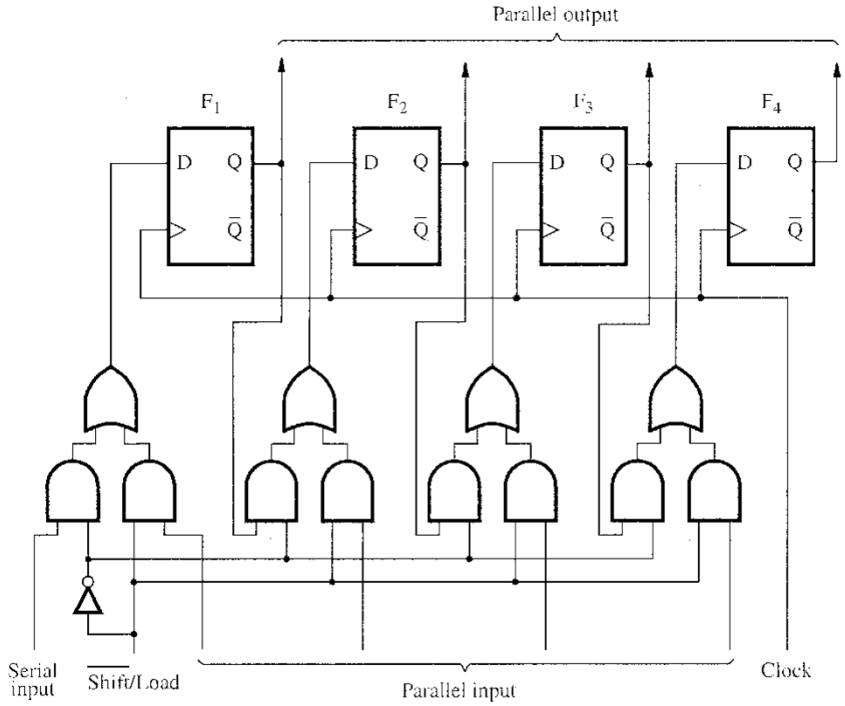


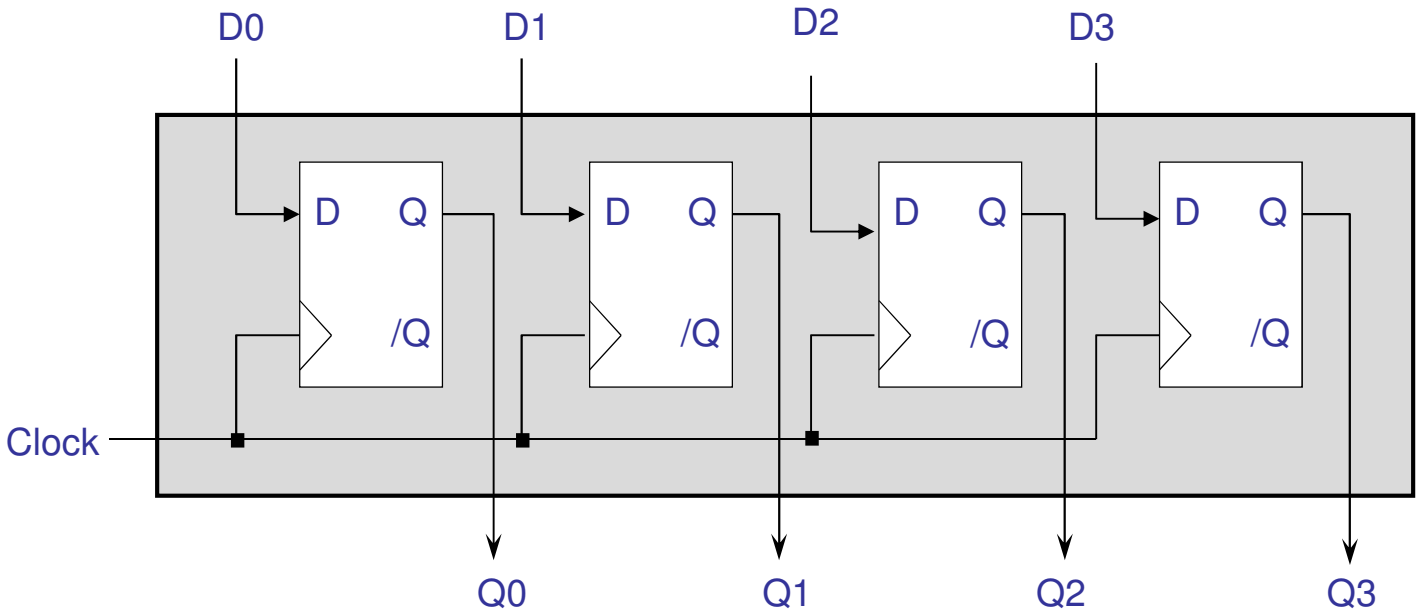
Figure A.34. Parallel-access shift register.

Registri

- I registri si distinguono sulla base dei seguenti aspetti:
- Modalità di caricamento dati
  - Parallelo
  - Seriale
- Modalità di lettura dati
  - Parallelo
  - Seriale
- Operazioni sui dati:
  - Scorrimento a destra
  - Scorrimento a sinistra
  - Scorrimento circolare
  - Scorrimento con immissione di un valore

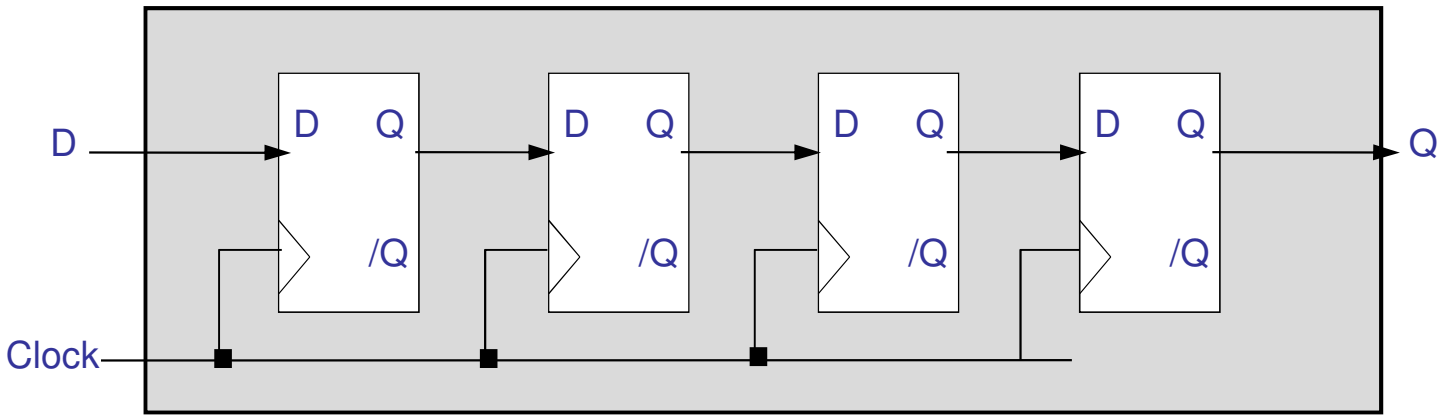
Registri

- Registro *parallelo-parallelo* a 4 bit



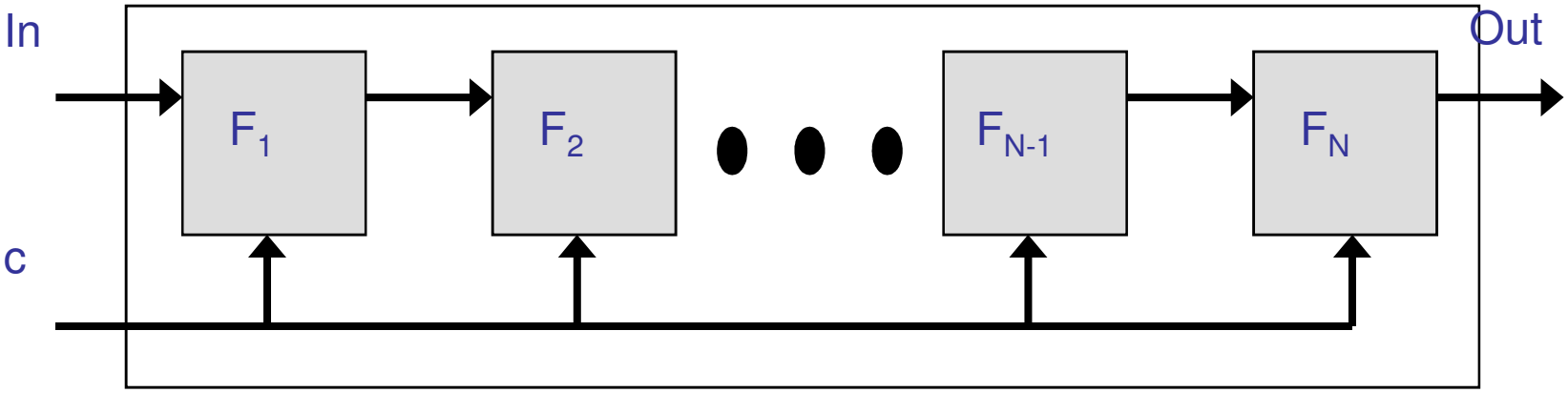
Registri

- Registro *serie-serie* a 4 bit (*Shift Register*)

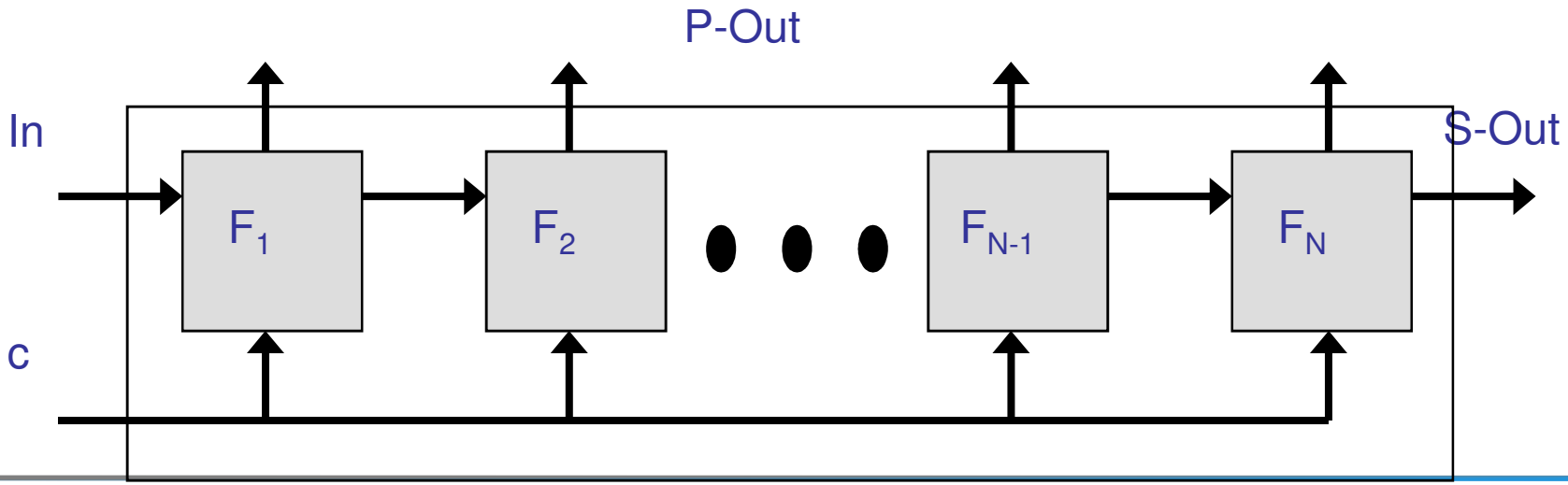


Registri a scorrimento

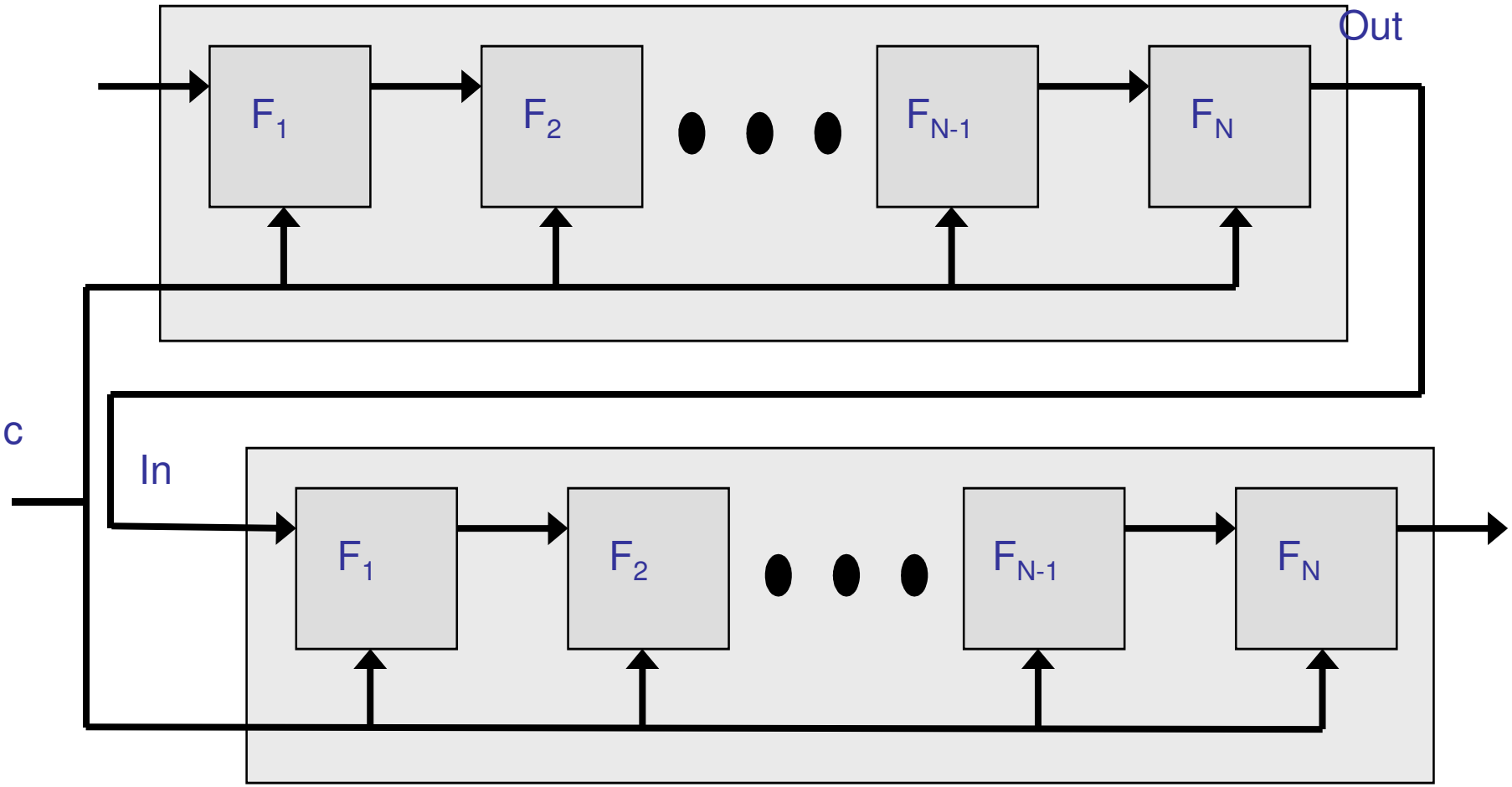
Input: Serie -> Output: Serie



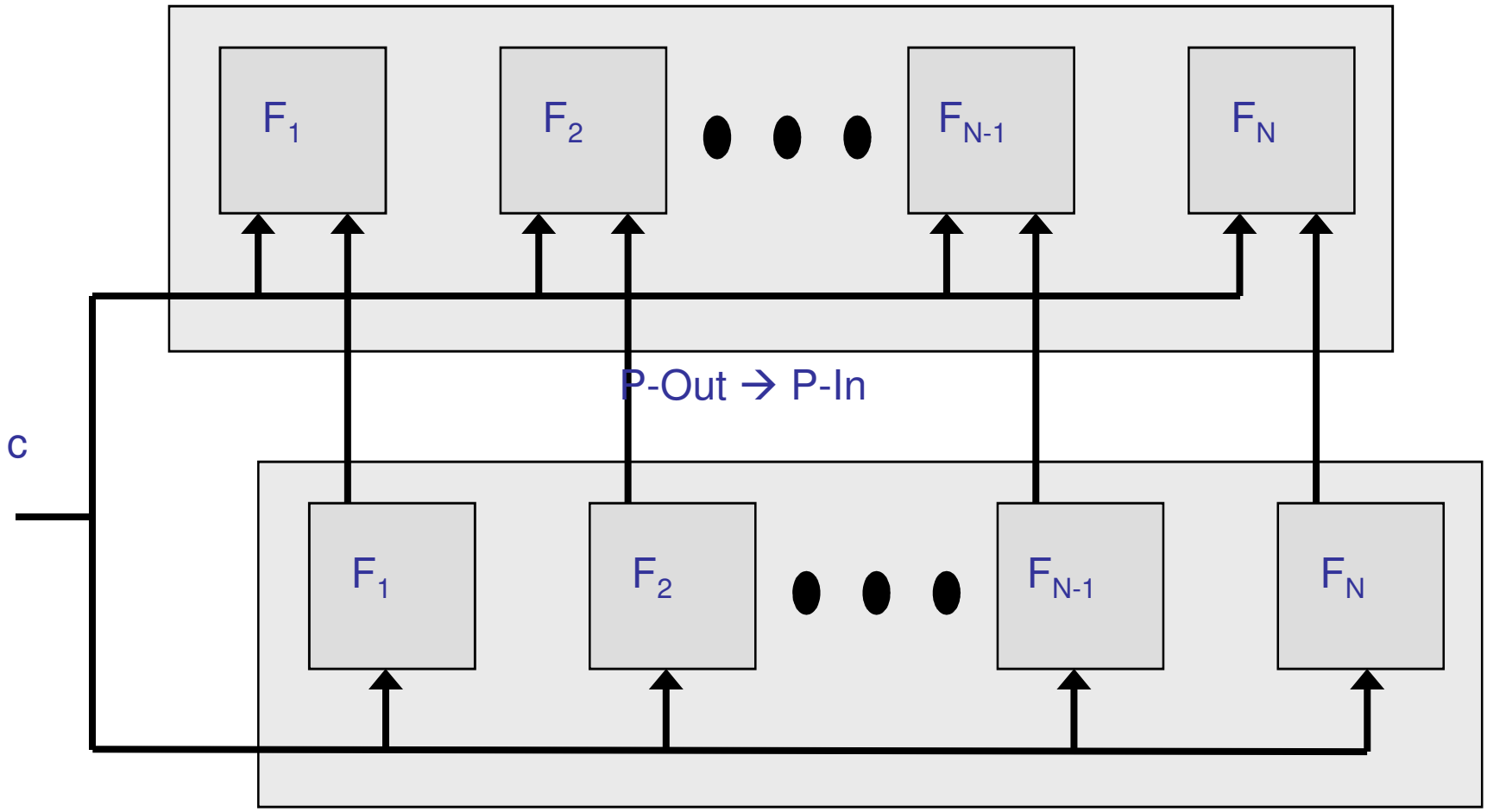
Input: Serie -> Output: Serie/Parallelo



Trasferimento seriale

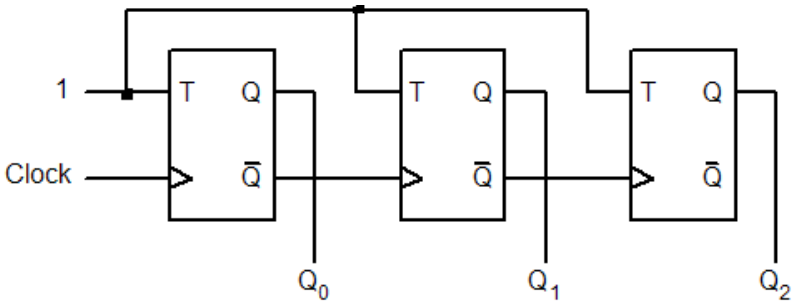


Trasferimento parallelo

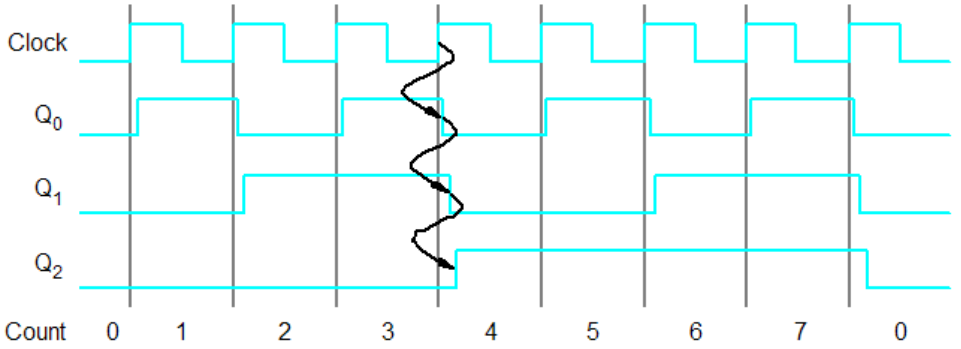


Counter

- Realizzato con f-f T per diversi scopi:
  - Scaler: genera clock a frequenze sottomultiple di quella del clock principale
- Gli stati dei f-f effettuano un conteggio binario
- Il Clk può essere unico o comandare tutti i f-f (asincrono, sincrono)



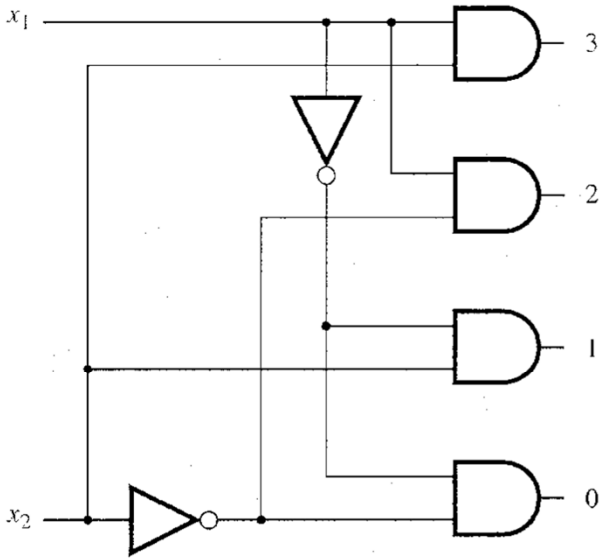
(a) Circuit



(b) Timing diagram

Decoder

- Traduce un input di n bit (input codificato), in uno stato di uscita (output decodificato)
  - Es.: n bit input -> 2^n stati di uscita



x_1	x_2	Active output
0	0	0
0	1	1
1	0	2
1	1	3

Figure A.36. A two-input to four-output decoder.

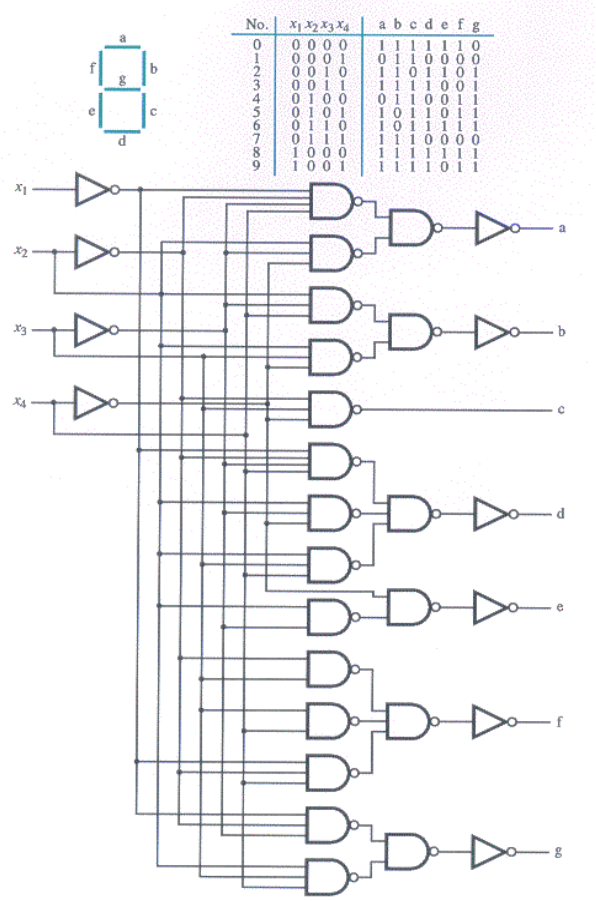
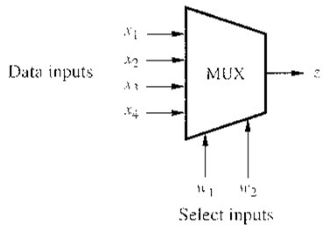


Figure A.37. A BCD to seven-segment display decoder.

Multiplexer

- Circuito che sceglie uno degli input da portare in output
  - Gli input di selezione sono detti appunto “select”
  - Es.: per 2^k input servono k select



w_1	w_2	z
0	0	x_1
0	1	x_2
1	0	x_3
1	1	x_4

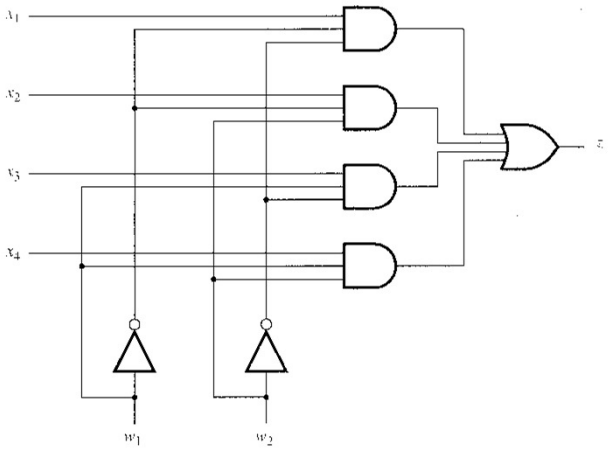


Figure A.38. A four-input multiplexer.

Può essere utilizzato per implementare una qualunque funzione:

e.g.
4 input mux -> f(x1,x2,x3)

8 input mux-> F(x1,x2,x3,x4)
...

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

⇒

x_1	x_2	f
0	0	0
0	1	x_3
1	0	1
1	1	$\bar{x}_3$

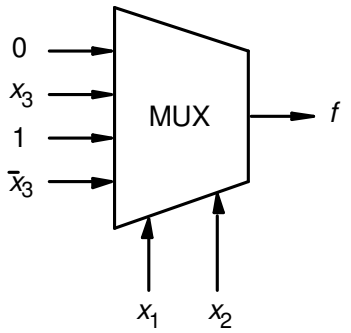


Figure A.39. Multiplexer implementation of a logic functi

Dispositivi programmabili (PLD)

- Programmable Logic Device (PLD): array di elementi programmabili che implementano funzioni in forma di somme di prodotti

• PLA: programmabile sia sulle AND che sulle OR
• PAL: programmabile sugli input alle AND ma non sulle OR
• CPLD: interconnessione di più PAL
• FPGA: realizzano circuiti logici più complessi (RAM, ROM, sistemi embedded) e veloci. Sono molto facili da sviluppare su VLSI, rispetto a quelli custom.

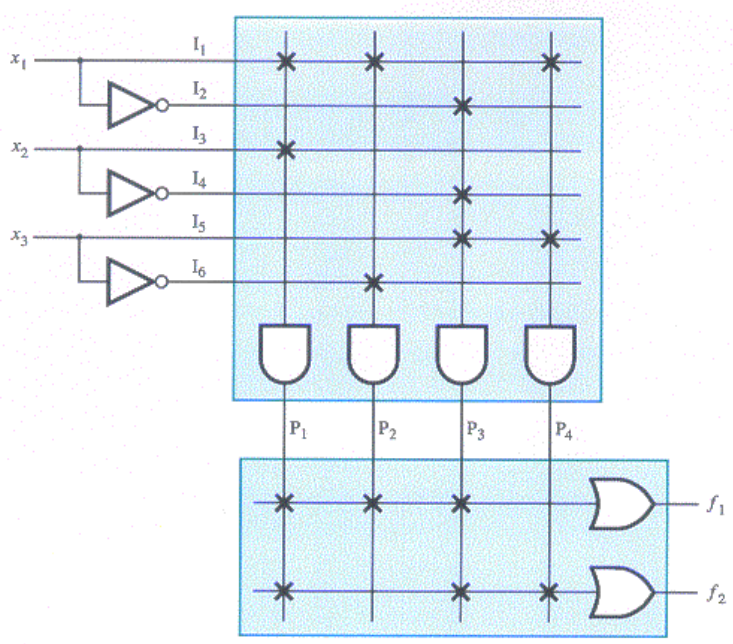


Figure A.42. A simplified sketch of the PLA in Figure A.41.

Circuiti Sequenziali

- L'output dipende dall'input attuale e dagli input passati
- Tali circuiti sono definiti da *stati*
  - I contatori e i registri a scorrimento ne sono un esempio
- Diagramma a stati:
  - rappresentazione grafica degli stati con nodi
  - transizioni di stato con frecce
  - un'etichetta sulle frecce indica gli input che causano la transizione e il valore degli output del circuito

Esempio: contatore mod 4

- Uscita z=1 se il contatore conta 2.
- L'input x determina se il conteggio va in avanti o all'indietro (0 o 1)
- Il Clock scandisce il conteggio (per es. fronte negativo)
- Le variabili per rappresentare gli stati sono dette *variabili di stato*

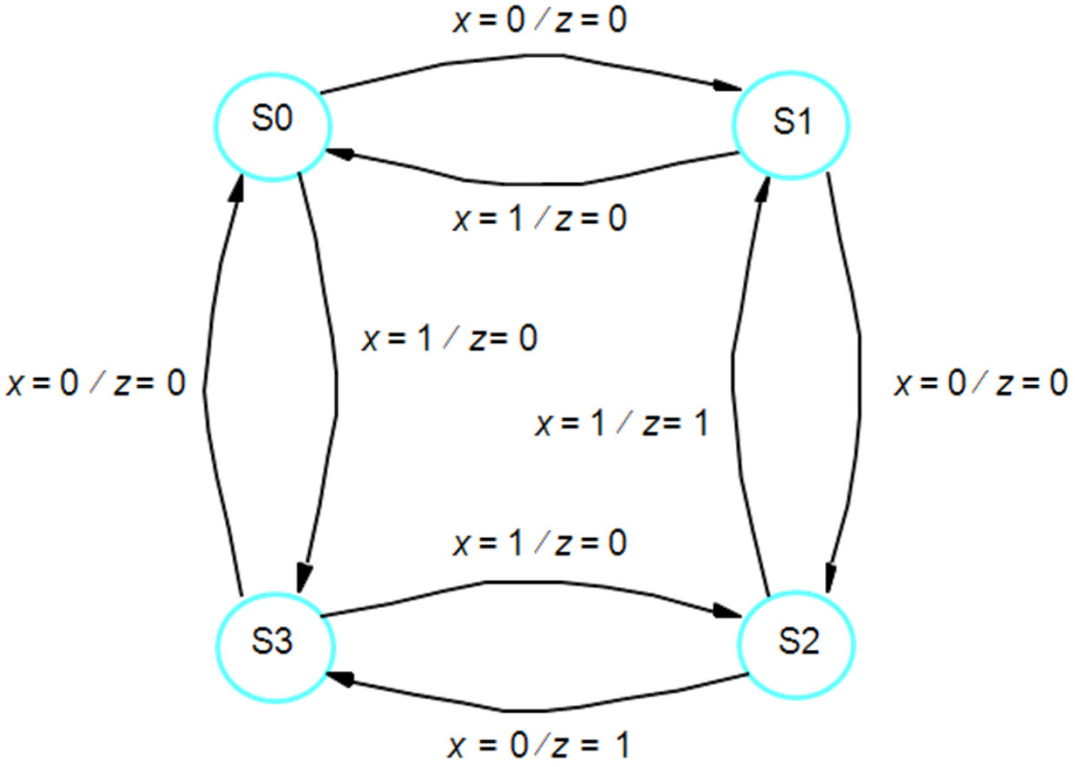


Tabelle degli stati e circuito

Present state	Next state		Outputz	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
S0	S1	S3	0	0
S1	S2	S0	0	0
S2	S3	S1	1	1
S3	S0	S2	0	0

Present state	Next state		Output z	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$y_2 y_1$	$Y_2\ Y_1$	$Y_2\ Y_1$		
0 0	0 1	1 1	0	0
0 1	1 0	0 0	0	0
1 0	1 1	0 1	1	1
1 1	0 0	1 0	0	0

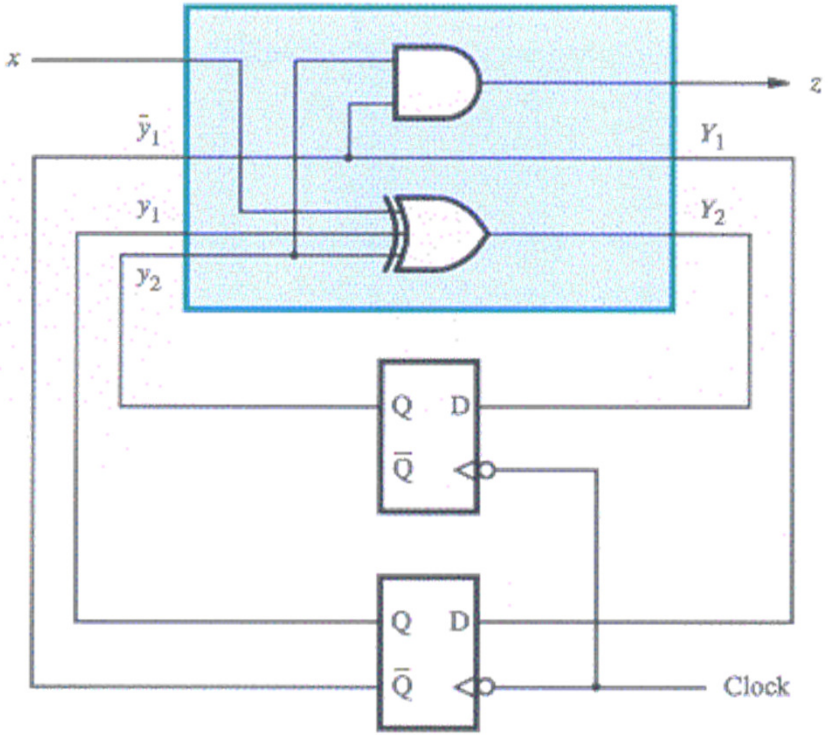


Figure A.50. Implementation of the up/down counter.

Diagramma di temporizzazione e FSM

- Osserviamo che la rete combinatoria non introduce ritardo nella variazione delle variabili di stato, a differenza della rete sequenziale (in realtà sui gates c'è, ma è trascurabile rispetto ai flip flop)
  - *Macchina a stati finiti (FSM)*
- Se le variabili del circuito sono comandate dallo stesso clock si parla di *circuiti sequenziali sincroni*

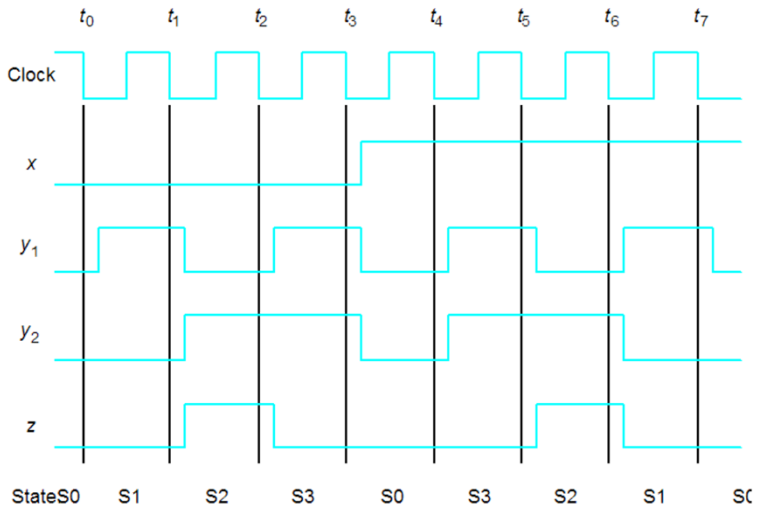


Figure A.51. Timing diagram for the circuit in Figure A.50.

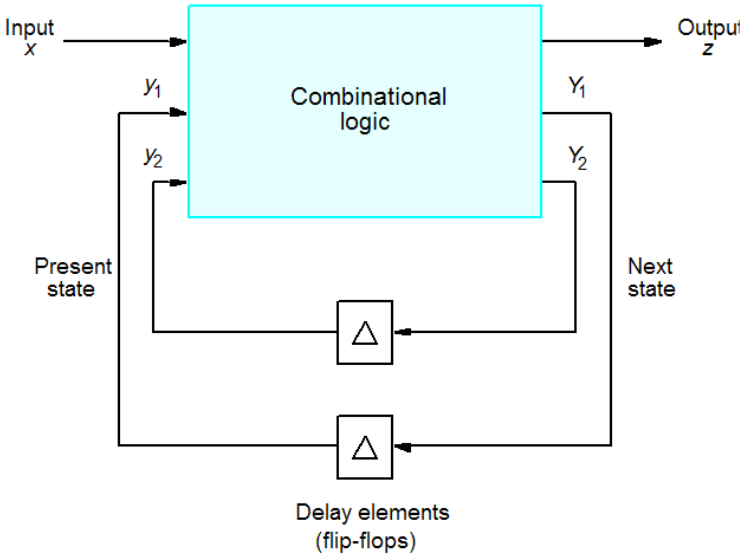


Figure A.52. A formal model of a finite state machine.

Sintesi di una macchina sequenziale

- La sintesi si svolge nei seguenti passi:
  1. Realizzazione del *diagramma degli stati* a partire dalle specifiche informali del problema
  2. Costruzione della *tabella degli stati*
  3. Riduzione del numero degli stati: *ottimizzazione*
  4. Assegnamento degli stati: *codifica*
  5. Costruzione della *tabella di verità*
  6. Sintesi della rete combinatoria che realizza la funzione stato prossimo
  7. Sintesi della rete combinatoria che realizza la funzione d'uscita
  8. Implementazione della parte combinatoria del circuito

Codifica degli Stati (1/2)

- La minimizzazione del numero di stati consente di ridurre il numero di elementi di memoria necessari a codificare gli stati stessi
  - A valle della minimizzazione realizzo la tabella degli stati
- E' necessario codificare gli stati, cioè indicare come sono rappresentati sui flip-flop:
  - Tabella delle transizioni: per un dato input indica le codifica dello stato successivo
  - Tabella delle eccitazioni: per un dato input indica l'ingresso ai flip-flop che codificano lo stato prossimo
  - Nota: se uso Flip-Flop D le due tabelle coincidono

E.g. per input x=0

transizione S1(0,1)->S2(1,0)

Quali flip-flop sto usando
nelle tabelle di eccitazione quì indicate?

Stato	X=0
(0,1)	(1,0)
(1,0)	...

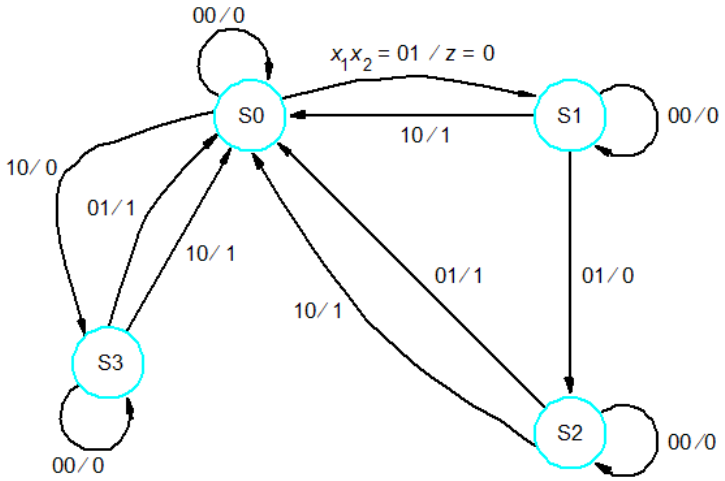
Stato	X=0
(0,1)	(1,1)
(1,0)	...

Codifica degli Stati (2/2)

- La codifica influenza la complessità della rete combinatoria perchè lo stato corrente è input alla rete e, inoltre, l'uscita della rete è eccitazione dei flip-flop per lo stato successivo
- Scegliere la lunghezza del codice -> numero di flip-flop
  - Il minimo è l'intero superiore di $log_2 |S|$, con |S| numero degli stati
- *Codifica sparsa* con un bit alto per stato -> tanti bit quanti sono gli stati (|S|=4): 0001, 0010, 0100, 1000
- Criteri di codifica a conteggio binario con numero di bit minimo :
  - E.g. (|S|=5) S0=000, S1=001, S2=010, S3=011, S4=100, S5=101
- Criteri a minima distanza (adiacenza):
  - Due stati con uguale stato prossimo a seguito dello stesso ingresso, devono avere codifiche adiacenti (1bit di differenza)
  - Stati prossimi dello stesso stato con ingressi adiacenti devono essere codificati con codifica adiacente
  - Stati con uscite uguali per il medesimo ingresso devono avere codifiche adiacenti

Esempio di FSM (1/2)

- Distributore automatico a 30 cents (erogazione prodotto, z). Accetta 25 (x1) e 10 (x2) cents (mai assieme) e non dà resto



$x_1 = 1$ ~ quarter deposited

$x_2 = 1$ ~ dime deposited

$z = 1$ ~ dispense merchandise (i.e., a total of 30 cents deposited)

Input combination $x_1 x_2$ = 11 cannot occur

Present state	Next state				Outputz			
	$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 10$	$x_1x_2 = 11$	$x_1x_2 = 00$	$x_1x_2 = 01$	$x_1x_2 = 10$	$x_1x_2 = 11$
y_2y_1	$Y_2\,Y_1$	$Y_2\,Y_1$	$Y_2\,Y_1$	$Y_2\,Y_1$				
S0 0 0	0 0	0 1	1 1	-	0	0	0	-
S1 0 1	0 1	1 0	0 0	-	0	0	1	-
S2 1 0	1 0	0 0	0 0	-	0	1	1	-
S3 1 1	1 1	0 0	0 0	-	0	1	1	-

Figure A.54. Assigned state table for the vending machine example.

Esempio di FSM (2/2)

x_1	x_2	y_2	y_1	Y_2	Y_1	z
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	0	1
1	1	0	0	d	d	d
1	1	0	1	d	d	d
1	1	1	0	d	d	d
1	1	1	1	d	d	d