Algebra booleana e circuiti logici

Salvatore Orlando

Algebra & Circuiti Elettronici

- **I computer operano con *segnali elettrici* con valori di potenziale discreti**
  - **sono considerati significativi soltanto due *potenziali* (high/low)**
  - **i potenziali intermedi, che si verificano durante le transizioni di potenziale, non vengono considerati**
- **L'aritmetica binaria è stata adottata proprio perché i bit sono rappresentabili naturalmente**
  - **tramite elementi elettronici in cui siamo in grado di distinguere i 2 stati del potenziale elettrico (high/low)**

- **Il funzionamento dei circuiti elettronici può essere modellato tramite l'Algebra di Boole**
  - **solo 2 valori:**
    - **valore logico *True* (1 o *asserted*) ⇒ livello di potenziale *alto***
    - **valore logico *Falso* (0 o *deasserted*) ⇒ livello di potenziale *basso***
  - **operazioni logiche Booleane per combinare i valori**

Blocco logico

- **Blocco logico**
  - **circuito elettronico con linee (*fili*) in *input* e *output***
  - **possiamo associare variabili logiche con le varie linee in input/output**
    - **i valori che le variabili possono assumere sono quelli dell'Algebra di Bool**

  - **il circuito calcola una o più funzioni logiche, ciascuna esprimibile tramite la combinazione di operazioni dell'Algebra di Bool sulle variabili in input**

- **Circuito *combinatorio***
  - **senza elementi di *memoria* - produce *output* che dipende funzionalmente solo dall'*input***
- **Circuito *sequenziale***
  - **con elementi di *memoria* - produce *ouput* che dipende non solo dall'*input* ma anche dallo *stato* della memoria**
- **All'inizio ci concentreremo sui *circuiti combinatori***

Tabelle di Verità

- **Funzione logica completamente specificato tramite *Tabella di Verità***

- **Dati n inputs bit, il numero di configurazioni possibili degli input, ovvero il numero di righe della *Tabella di Verità*, è 2^n**
  - **per ogni bit in output, la tabella contiene una colonna, con un valore definito per ognuna delle combinazioni dei bit in input**

- **Esempio di tabella con 3 input A, B e C, e 2 output D ed E**

A	B	C	D	E
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Algebra Booleana

- **Funzione logica** completamente specificato tramite ***Equazione logica***
  - bit in *input* e *output* rappresentati tramite *variabili logiche* (con valori 0 o 1)
  - input *combinati* tramite le operazioni di *somma* (OR), *prodotto* (AND) e *inversione* (NOT) *logica* dell'algebra di Boole
    - OR (A+B): risultato uguale ad 1 (true) se almeno un input è 1 (true)
    - AND (A·B): risultato uguale ad 1 (true) solo se tutti gli input sono 1 (true)
    - NOT (~A): risultato uguale all'inverso dell'input (0→1 oppure 1→0)

- Tabelle di verità delle operazioni di NOT, AND, OR:

A	X
0	1
1	0

X = ~A

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

X = A · B

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

X = A + B

Proprietà dell'algebra di Boole

PROPRIETÀ

• Identità:	A+0=A	A·1=A
• Nullo:	A+1=1	A·0=0
• Idempotente:	A+A=A	A·A=A
• Inverso:	A+(~A)=1	A·(~A)=0
• Commutativa:	A+B=B+A	A·B=B·A
• Associativa:	A+(B+C)=(A+B)+C	A·(B·C)=(A·B)·C
• Distributiva:	A·(B+C)=(A·B)+(A·C)	A+(B·C)=(A+B)·(A+C)
• DeMorgan:	~(A+B)=(~A)·(~B)	~(A·B)=(~A)+(~B)

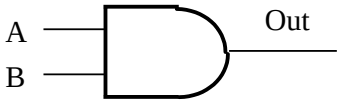
- Ad esempio, gli output D ed E della precedente *Tabella di verità* possono essere espresse come *Equazioni logiche,* semplificabili applicando le proprietà di sopra

A	B	C	D	E
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

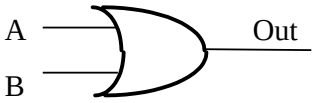
```
D = (~A ~B ~C) + (~A B ~C) + (A B ~C) =
  = (~A ~B ~C) + B ~C (~A + A) =
  = (~A ~B ~C) + (B ~C)

E = (~A ~B C) + (A B ~C)
```

Dalle equazioni logiche ai circuiti combinatori

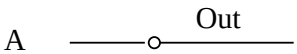
- **Porte logiche**
  - ***AND*:** **A · B**

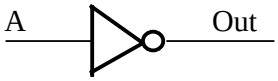


  - ***OR*:** **A + B**

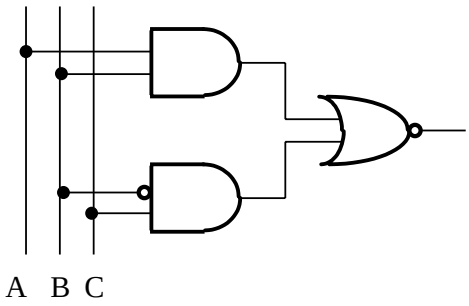


  - ***NOT*:** **~A**



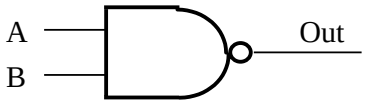


- **Esempio di equazione e corrispondente circuito:**
  - **~((AB) + (~BC))**



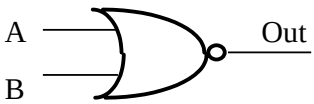
Operazioni NAND o NOR

NAND: porta e tabella di verità



A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

NOR: porta e tabella di verità

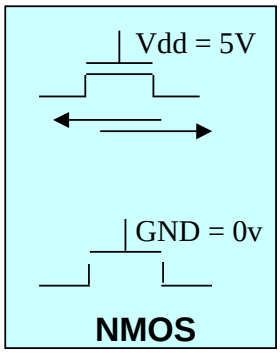


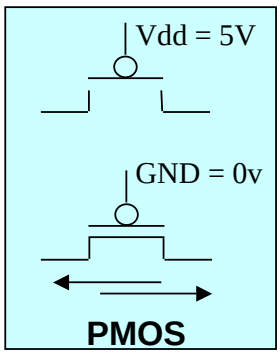
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

- **NAND (inverso dell'operazione AND) : ~(A · B) = A NAND B**
- **NOR (inverso operazione OR) : ~(A + B) = A NOR B**
  - **Si può dimostrare che le operazioni NAND o NOR (e le corrispondenti porte) sono sufficienti per implementare qualsiasi funzione logica**
  - **NAND**
    - **~A = ~A + 0 = ~(A · 1) = A NAND 1**
    - **A+B = ~ ~(A+B) = ~ (~A · ~B) = ~(~(A · 1) · ~(B · 1)) = (A NAND 1) NAND (B NAND 1)**
    - **A · B = (A · B)+0 = ~ ~((A · B)+0) = ~(~(A · B) · 1) = ((A NAND B) NAND 1)**
  - **NOR**
    - **~A = ~A · 1 = ~(A + 0) = A NOR 0**
    - **A+B = (A+B) · 1 = ~ ~((A+B) · 1) = ~(~(A+B) + 0) = ((A NOR B) NOR 0)**
    - **A · B = ~ ~(A · B) = ~(~A + ~B) = ~ (~(A+0) + ~(B+0)) = (A NOR 0) NOR (B NOR 0)**

Porte logiche implementabili tramite transistor

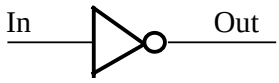
- **Tecnologia CMOS (Complementary Metal Oxide Semiconductor) per realizzare transistor sul silicio**
  - **I *transistor* sono degli *interruttori velocissimi* che lasciano (o meno) passare la corrente, e sono comandati da un segnale elettrico**
  - **NMOS (N-Type Metal Oxide Semiconductor) transistor**
  - **PMOS (P-Type Metal Oxide Semiconductor) transistor**
- **NMOS Transistor**
  - **Se applichi un ALTO voltaggio (Vdd), il transistor diventa un "conduttore"**
  - **Se applichi un BASSO voltaggio (GND), il transistor interrompe la conduzione (resistenza infinita)**
- **PMOS Transistor**
  - **Se applichi un ALTO voltaggio (Vdd), il transistor interrompe la conduzione (resistenza infinita)**
  - **Se applichi un BASSO voltaggio (GND), il transistor diventa un "conduttore"**



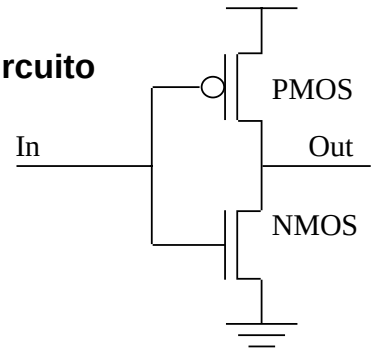


Componenti base: Inverter CMOS

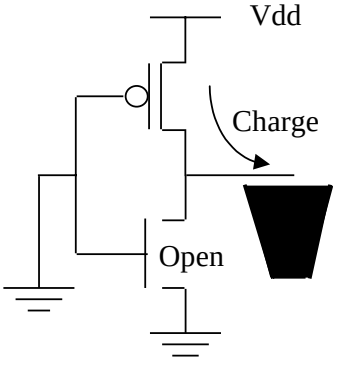
Simbolo

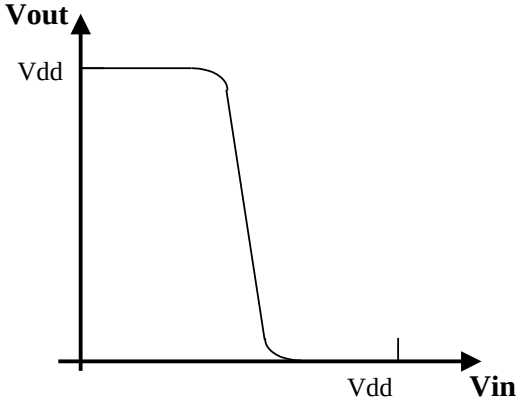


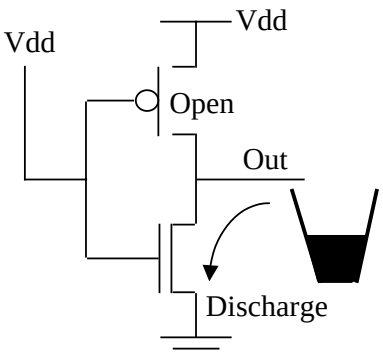
Circuito



- **Operazione d'inversione**

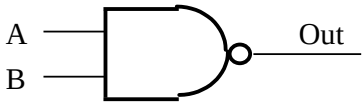




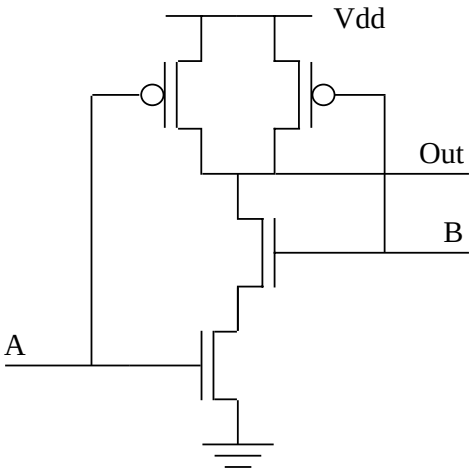


Componenti base: Porte Logiche NOR e NAND

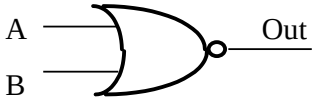
Porta NAND



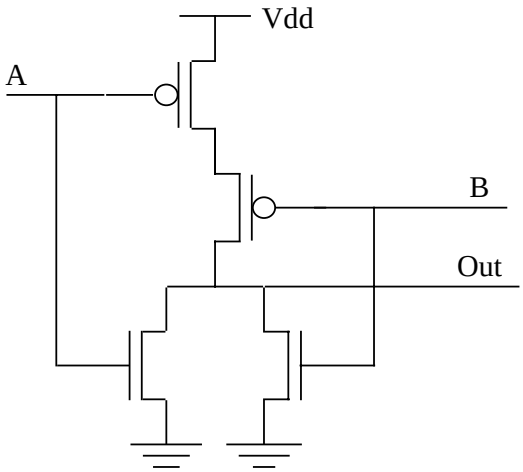
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



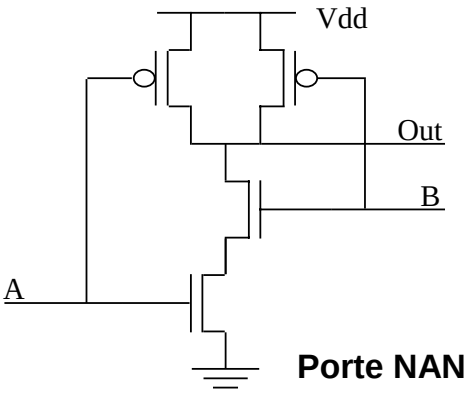
Porta NOR



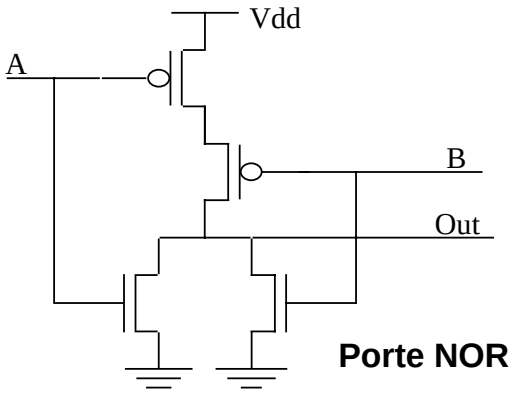
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0



Confronto tra Porte



Porte NAND



Porte NOR

- **Se i transistor PMOS sono più veloci:**
  - **È meglio avere transistor PMOS in serie**
  - **Porte NOR preferite**
- **Se i transistor NMOS sono più veloci:**
  - **È meglio avere transistor NMOS in serie**
  - **Porte NAND preferite**

Forme canoniche

- Ogni **funzione logica** può essere rappresentata come **equazione** o come **tabella di verità**
- Ogni equazione logica può essere scritta in **forma canonica** tramite l'uso degli operatori AND, OR e NOT
  - equazione in forma canonica derivabile dalla corrispondente tabella

- **Forma canonica** *SP (somma di prodotti)*

A	B	C	E
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

- Per ogni entry uguale ad 1 dell'ouput (E)
  - genera un *prodotto* (**mintermine**) degli input (A, B e C), dove gli input uguali a 0 appaiono negati.
- NOTA: ciascun *prodotto* vale 1 solo per quella data combinazione dei fattori (dei valori delle variabili in input).
- Per ottenere l'equazione in forma SP, *somma* i prodotti così ottenuti: **E = (~A~BC) + (AB~C)**

Forme canoniche

- **Forma canonica** *PS (prodotto di somme)*

A	B	C	E
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

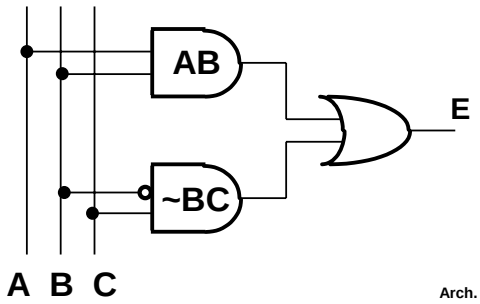
- Per ogni entry uguale ad 0 dell'ouput (E)
  - genera una *somma* (**maxtermine**) degli input (A, B e C), dove gli input uguali a 1 appaiono negati.
- NOTA: ciascuna *somma* vale 0 solo per quella data combinazione degli addendi (dei valori delle variabili in input).
- Per ottenere l'equazione in forma PS, effettua il *prodotto* delle somme così ottenute:

E = (A+B+C) · (A+~B+C) · (A+~B+~C) · (~A+B+C)

(~A+B+~C) · (~A+~B+~C)

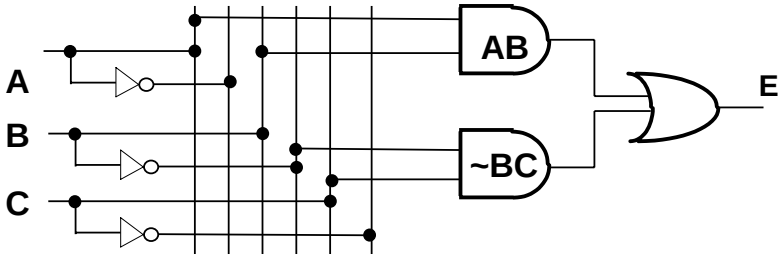
Dalle forme canoniche ai circuiti (2-level logic)

- **Prendiamo una equazione logica espressa come *somma di prodotti* (SP) che realizza una funzione logica di *n input* e *1 output***
  - **1^ livello di porte AND per i *prodotti***
    - **una porta AND per ogni prodotto**
    - **arietà (fan-in) delle porte dipende dal numero di fattori dei prodotti (max *arietà* = *no. variabili in input*)**
    - **fattori dei prodotti (*variabili in input*) entrano nelle porte *direttamente* o *invertite***
  - **2^ livello costituito da una porta OR per la *somma***
    - **arietà della porta dipende dal numero di prodotti**
  - **i segnali in input attraversano**
    - **2 livelli di porte logiche (AND e OR) + eventuali negazioni**
- **Esempio di forma SP: E = (AB) + (~BC)**



Rappresentazione alternativa di un circuito a 2 livelli

- **Prendiamo una equazione logica espressa come *somma di prodotti* (SP) che realizza una funzione logica di *n input* e *1 output***
  - **una porta AND per ogni prodotto**
  - **un invertitore per ogni variabile**
  - **input delle porte AND collegate con le linee corrispondenti alle varie variabili (o alla loro negazione)**
  - **l'output delle porte AND collegate in input alla porta OR**
- **Esempio di forma SP: E = (AB) + (~BC)**



Minimizzazione circuiti

- ***Scopo minimizzazione***
  - **data una equazione in forma normale (es. SP), si riduce il numero di prodotti, oppure il numero di variabili coinvolte in ogni prodotto**
  - **minimizzando si riduce quindi il costo del circuito combinatorio corrispondente => meno porte, con arietà (fan-in) ridotta**

- **Esempio di minimizzazione usando le proprietà dell'algebra di Boole**
  - **Funzione F che assume certi valori indipendentemente dal valore di A:**

$$F = \sim AB + AB = \quad \text{(distributiva)}$$
$$= B(\sim A + A) = \quad \text{(inverso)}$$
$$= B\,1 = B \quad \text{(nullo)}$$

  - **A è un input DON'T CARE (che non importa ai fini della definizione dell'equazione)**

Esempio di minimizzazione

- **Data una tabella di verità, per minimizzare la funzione logica è necessario esplicitare le variabili in input come DON'T CARE (che possiamo *non considerare*)**
  - **dovendo realizzare una funzione in forma SP, ci concentreremo sulle combinazioni di righe della tabella che generano gli 1**
  - **si tratta di individuare gli insiemi composti, rispettivamente, da 2^1, 2^2, … o 2^n righe per cercare se esistono *1*, *2*, … o *n* variabili DON'T CARE**

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$\Rightarrow$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	X	1	X	1
0	1	0	0	0
0	1	0	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

- **f = ~A~BC~D + ~A~B CD + ~ABC~D + ~ABCD**
- **~AC compare in tutti i prodotti, combinato con tutti i possibili valori di B e D**
- **B e D sono variabili DON'T CARE, e si può minimizzare eliminandole:**
  f = ~AC
- **Infatti:**
  f = ~AC (~B~D + ~BD + B~D + BD) =
  = ~AC (1) = ~AC

Tecniche di mimimizzazione

- **Intuitivamente, per semplificare una tabella di verità di N variabili e minimizzare la corrispondente forma normale SP, ovvero per scoprire le variabili DON'T CARE, basta individuare:**
  - **2^1 (coppie di) righe con output 1 dove**
    - **i valori assunti da N-1 variabili appaiono fissi**
    - **tutti i possibili valori di 1 variabile (X) appaiono combinati con con gli altri N-1 valori fissi**

      $\Rightarrow$ la variabile X è DON'T CARE
  - **2^2 (4-ple di) righe con output 1 dove**
    - **i valori assunti da N-2 variabili appaiono fissi**
    - **tutti i possibili valori di 2 variabili (X,Y) appaiono combinati con con gli altri N-2 valori fissi**

      $\Rightarrow$ le variabili X e Y sono DON'T CARE
  - **2^3 (8-ple di) righe con output 1 dove**
    - **i valori assunti da N-3 variabili appaiono fissi**
    - **tutti i possibili valori di 3 variabili (X,Y,Z) appaiono combinati con con gli altri N-3 valori fissi**

      $\Rightarrow$ le variabili X, Y e Z sono DON'T CARE
  - **2^4 (16-ple di) righe con output 1 dove ….**

Mappe di Karnaugh

- **Difficile minimizzare *a mano* guardando la tabella di verità. Esistono comunque algoritmi efficienti, automatizzabili, ma difficili da usare a mano.**
- **Per minimizzare a mano *funzioni* di poche variabili, si possono rappresentare le tabelle di verità con le *mappe di Karnaugh***
  - **ogni quadrato della mappa individua una combinazione di variabili in *input***
  - **il valore contenuto nel quadrato corrispondente al corrispondente valore in *output***
  - **per convenzione nella mappa si inseriscono solo gli output uguali ad 1**
  - **da notare la combinazioni delle variabili in input che *etichettano* i due assi delle mappe:**
    - **codice di Gray: differenza di un singolo bit tra combinazioni consecutive**

B \ A	0	1
0		
1		1

2 variabili

C \ AB	00	01	11	10
0		1		
1		1		

3 variabili

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

4 variabili

Mappe di Karnaugh

- ***Scopo mappe:***
  - **individuare facilmente insieme di righe (2^1, 2^2, 2^3 righe, ecc.) della tabella di verità con variabili (1, 2, 3 variabili, ecc.) DON'T CARE**
  - **gli 1 corrispondenti a queste righe risultano infatti *adiacenti* nella mappa corrispondente**
    - **nel considerare l'*adiacenza* delle celle nella mappa, si tenga conto che i bordi orizzontali/verticali della mappa è come se si *toccassero***
    - **le combinazioni di 2^1, 2^2, 2^3 righe della tabella di verità originale con 1,2,3 variabili DON'T CARE diventano "*rettangoli*" di valori uguali ad 1 nella mappa di Karnaugh**
    - **questi rettangoli sono composti da 2^p valori uguali ad 1, e sono anche noti con il termine di *p-sottocubi***

Esempi di p-sottocubi

1-sottocubo

C \ AB	00	01	11	10
0		1		
1		1		

f = ~ AB

1-sottocubo

C \ AB	00	01	11	10
0				
1	1			1

f = C~B

2-sottocubo

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

f = ~B~D

3-sottocubo

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

f = ~D

3-sottocubo

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

f = B

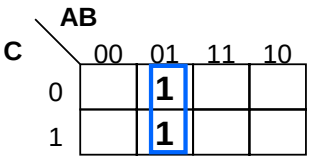
2-sottocubo

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

f = BD

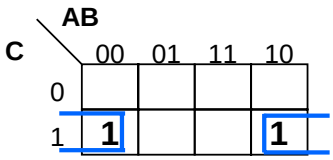
Grafica differente per rappresentare p-sottocubi

1-sottocubo



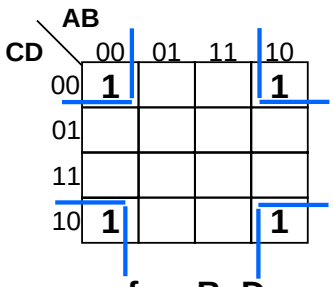
f = ~ AB

1-sottocubo



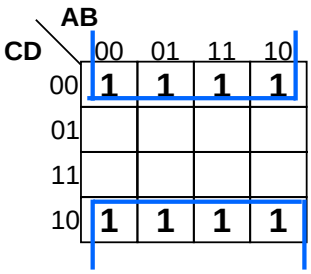
f = C~B

2-sottocubo



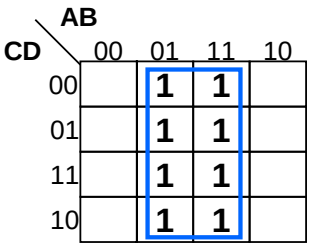
f = ~B~D

3-sottocubo



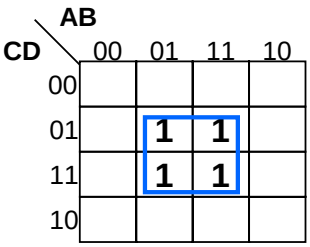
f = ~D

3-sottocubo



f = B

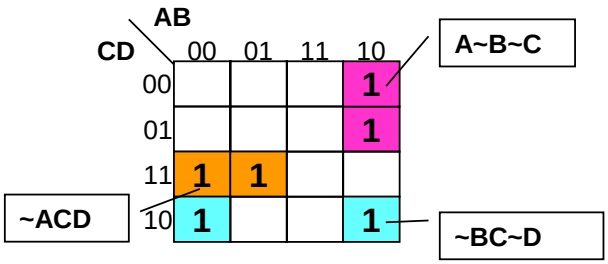
2-sottocubo



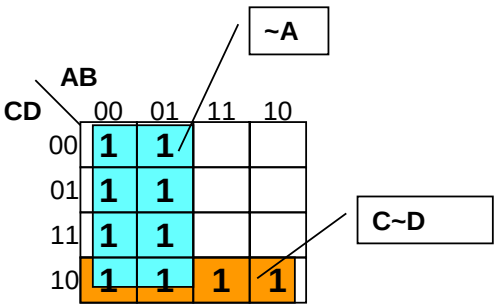
f = BD

Minimizzazione con mappe di Karnaugh

- **Intuitivamente**
  - **per minimizzazione il più possibile, basta scegliere i *più grandi rettangoli* (p-sottocubi) che ricoprono gli 1 della mappa**
  - **ATTENZIONE: gli stessi 1 possono essere ricoperti da *più rettangoli* (da più p-sottocubi)**

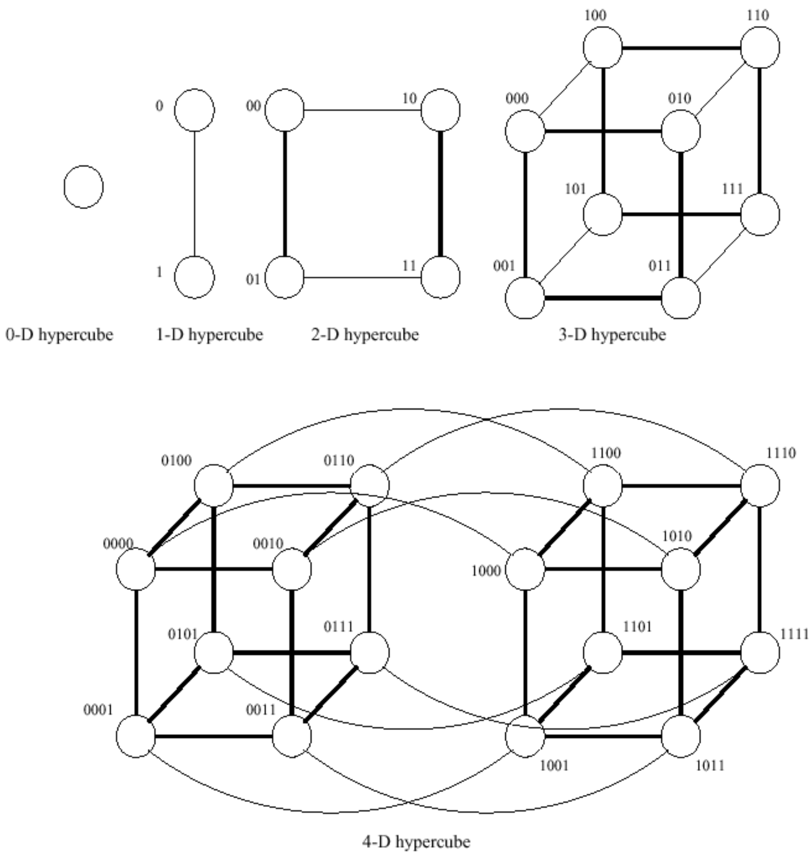


f = ~ACD + A~B~C + ~BC~D



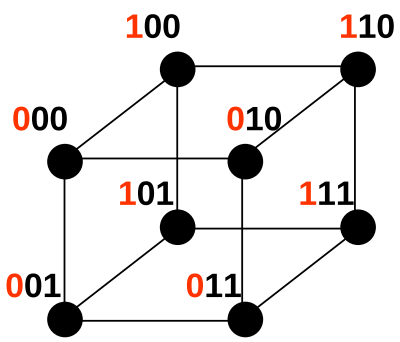
f = ~A + C~D

Ipercubi e Mappe di Karnaugh

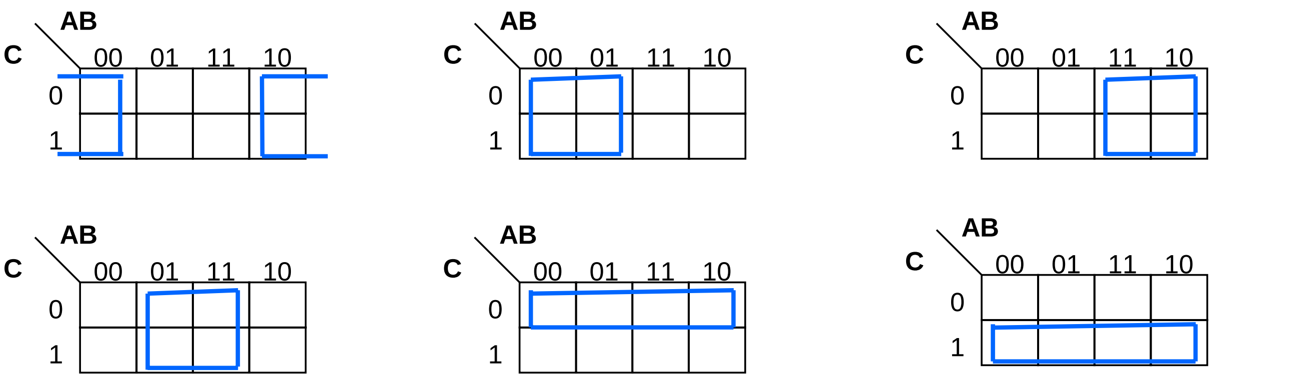


- Abbiamo definito alcuni gruppi di 2^p celle delle mappe come *p-sottocubi*
- La mappa di Karnaugh è in effetti la *rappresentazione tabellare* di un grafo con topologia ad ipercubo
  - ogni nodo dell'ipercubo a *n* dimensioni è *etichettato* con un numero binario a *n* cifre
  - ipercubo a *n dimensioni* ottenuto mettendo assieme 2 ipercubi di *n-1 dimensioni*
    - aggiungendo un bit nella rappresentazione delle etichette
  - i sottocubi si riferiscono a specifici sottoinsiemi di nodi connessi

Ipercubi e Mappe di Karnaugh

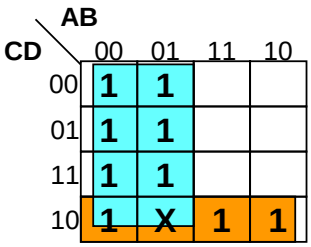


- Negli ipercubi le etichette dei nodi *connessi* differiscono di 1 solo bit (distanza di Hamming = 1)
- Consideriamo i 2-sottocubi dell'ipercubo a 3 dimensioni illustrato a sinistra
  - ogni 2-sottocubo contiene 2^2=4 nodi
  - ogni 2-sottocubo corrisponde ai 4 nodi che stanno su una delle 6 facce dell'ipercubo
  - abbiamo al più 6 2-sottocubi



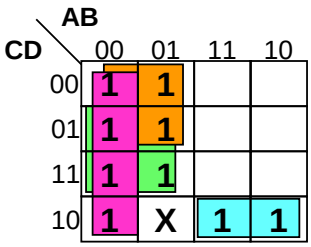
Funzioni incomplete

- **Alcuni output di una funzione, ovvero gli <u>output corrispondenti a particolari configurazione degli input</u>, possono *non interessare* (output DON'T CARE)**
  - **es. negli output della tabella di verità (o nella mappa di Karnaugh associata) possiamo avere degli X (dove X sta per DON'T CARE)**
- **Problema:**
  - **l'equazione logica e il corrispondente circuito <u>NON</u> possono essere incompleti**
  - **essi devono produrre un risultato in corrispondenza di TUTTE le combinazione dei valori in input**
  - **<u>TRUCCO</u>: al posto delle X (valori non specificati) si sceglie 1 o 0 in modo da ottenere la migliore minimizzazione**



Considerando X=1, solo 2 p-sottocubi:

f = ~A + C~D



Considerando X=0, ben 4 p-sottocubi:

f = ~A~B + ~A~C + ~AD + AC~D