

## 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**)  $\Rightarrow$  livello di potenziale **alto**
    - valore **logico False** (0 o **deasserted**)  $\Rightarrow$  livello di potenziale **basso**
  - operazioni logiche Booleane per combinare i valori

Arch. Elab. - S. Orlando 2

## 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	1	0

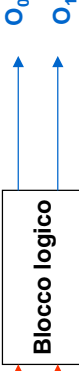
Arch. Elab. - S. Orlando 4

## Algebra booleana e circuiti logici

Salvatore Orlando

Arch. Elab. - S. Orlando 1

## Blocco logico

- **Blocco logico**
    - **circuito** elettronico con linee (*fil*) 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 **output** che dipende non solo dall'**input** ma anche dallo **stato** della memoria
  - All'inizio ci concentreremo sui **circuiti combinatori**

Arch. Elab. - S. Orlando 3

## 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

$$X = \sim A$$

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




$$X = A \cdot B$$

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

$$X = A + B$$

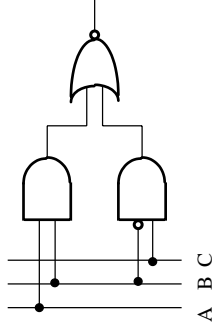
Arch. Elab. - S. Orlando 5

## Dalle equazioni logiche ai circuiti combinatori

- **Porte logiche**
  - **AND:**

  - **OR:**

  - **NOT:**


- Esempio di equazione e corrispondente circuito:

$$\sim((AB) + (\sim B))$$



Arch. Elab. - S. Orlando 7

## Proprietà dell'algebra di Boole

### PROPRIETÀ

- Identità:  $A \cdot 1 = A$
- Nullo:  $A \cdot 0 = 0$
- Idempotente:  $A + A = A$
- Inverso:  $A + (\sim A) = 1$
- Commutativa:  $A + B = B + A$
- Associativa:  $A + (B + C) = (A + B) + C$
- Distributiva:  $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- DeMorgan:  $\sim(A + B) = (\sim A) \cdot (\sim B)$

- 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

$$D = (\sim A \cdot \sim B \cdot \sim C) + (\sim A \cdot B \cdot \sim C) + (A \cdot B \cdot \sim C) =$$

$$= (\sim A \cdot \sim B \cdot \sim C) + B \cdot \sim C \cdot (\sim A + A) =$$

$$= (\sim A \cdot \sim B \cdot \sim C) + (B \cdot \sim C)$$

$$E = (\sim A \cdot \sim B \cdot C) + (A \cdot B \cdot \sim C)$$

Arch. Elab. - S. Orlando 6

## 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



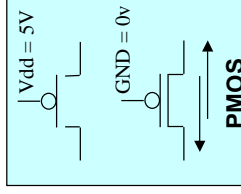
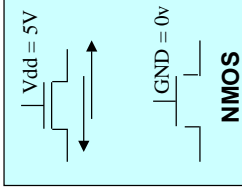
- **NAND** (inverso dell'operazione AND) :  $\sim(A \cdot B) = A \text{ NAND } B$
- **NOR** (inverso operazione OR) :  $\sim(A + B) = A \text{ NOR } B$
- Si può dimostrare che le operazioni NAND o NOR (e le corrispondenti porte) sono sufficienti per implementare qualsiasi funzione logica
- **NAND**

- $\sim A = \sim A + 0 = \sim(A \cdot 1) = A \text{ NAND } 1$
- $A + B = \sim \sim(A + B) = \sim(\sim A \cdot \sim B) = \sim(\sim(A \cdot 1) \cdot \sim(B \cdot 1)) = (A \text{ NAND } 1) \text{ NAND } (B \text{ NAND } 1)$
- $A \cdot B = (A \cdot B) + 0 = \sim \sim((A \cdot B) + 0) = \sim(\sim(A \cdot B) \cdot 1) = ((A \text{ NAND } B) \text{ NAND } 1)$
- **NOR**
- $\sim A = \sim A \cdot 1 = \sim(A + 0) = A \text{ NOR } 0$
- $A + B = (A + B) \cdot 1 = \sim \sim((A + B) \cdot 1) = \sim(\sim(A + B) + 0) = ((A \text{ NOR } B) \text{ NOR } 0)$
- $A \cdot B = \sim \sim(A \cdot B) = \sim(\sim(A + \sim B)) = \sim(\sim(A + 0) + \sim(B + 0)) = (A \text{ NOR } 0) \text{ NOR } (B \text{ NOR } 0)$

Arch. Elab. - S. Orlando 8

## Porte logiche implementabili tramite transistor

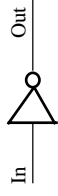
- Tecnologia CMOS (Complementary Metal Oxide Semiconductor) per realizzare transistor sul silicio
  - I **transistor** sono degli **interruttori velocissimi** che lasciano (o meno) passare la corrente, e sono comandati da un **segnale elettrico**
- NMOS (N-Type Metal Oxide Semiconductor) transistor
- 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”



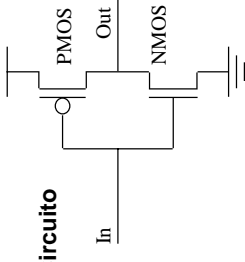
Arch. Elab. - S. Orlando 9

## Componenti base: Inverter CMOS

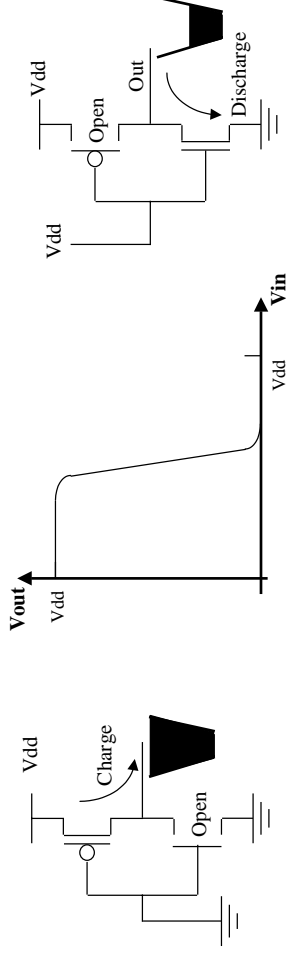
Simbolo



Circuito



- Operazione d'inversione



Arch. Elab. - S. Orlando 10

## Componenti base: Porte Logiche NOR e NAND

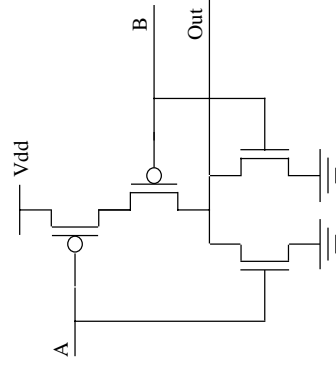
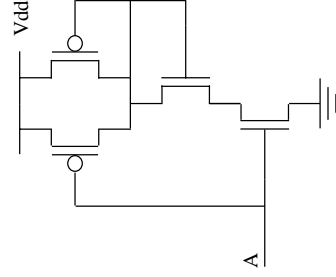
Porta NAND

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



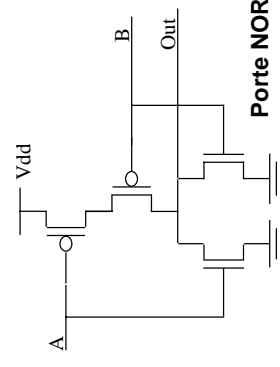
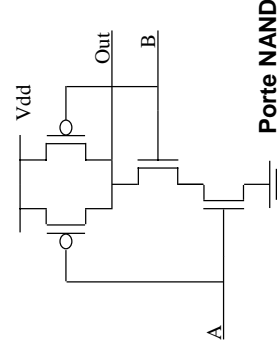
Porta NOR

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



Arch. Elab. - S. Orlando 11

## Confronto tra Porte



- 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

Arch. Elab. - S. Orlando 12

## 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

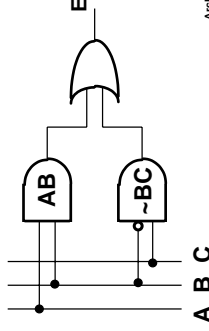
- Per ogni entry uguale ad 1 dell'output (E)
  - genera un **prodotto (mintermine)** degli input (A, B e C), dove gli input uguali a 0 appaiono negati.
- NOTA:** ciascun **prodotto** vale 1 solo per quella data combinazione dei fattori (dei valori delle variabili in input).
- Per ottenere l'equazione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

Arch. Elab. - S. Orlando 13

## 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:  **$E = (AB) + (\sim BC)$**



Arch. Elab. - S. Orlando 15

## Forme canoniche

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

- Per ogni entry uguale ad 0 dell'output (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:

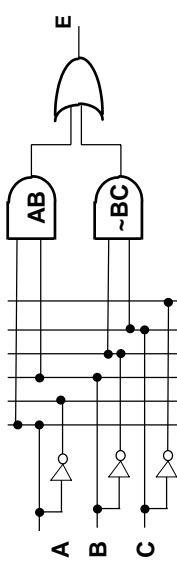
A	B	C	E
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$$E = (A+B+C) \cdot (A+\sim B+C) \cdot (A+\sim B+\sim C) \cdot (\sim A+B+C) \\ (\sim A+B+\sim C) \cdot (\sim A+\sim B+\sim C)$$

Arch. Elab. - S. Orlando 14

## 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:  **$E = (AB) + (\sim BC)$**



Arch. Elab. - S. Orlando 16

## 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 aritetà (fan-in) ridotta
- Esempio di minimizzazione usando le proprietà dell'algebra di Boole
  - Funzione F che assume certi valori **indipendentemente dal valore di A**:  
$$\begin{aligned} F &= \sim AB + AB = \text{(distributiva)} \\ &= B (\sim A + A) = \text{(inverso)} \\ &= B \cdot 1 = B \text{ (nullo)} \end{aligned}$$
  - A è un input **DON'T CARE** (che non importa ai fini della definizione dell'equazione)

Arch. Elab. - S. Orlando 17

## 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<sup>1</sup> (**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
  - 2<sup>2</sup> (**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
  - 2<sup>3</sup> (**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
  - 2<sup>4</sup> (**16-ple di**) righe con output 1 dove ....

Arch. Elab. - S. Orlando 19

## 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<sup>1</sup>, 2<sup>2</sup>, ... o 2<sup>n</sup> righe per cercare se esistono 1, 2, ... o **n** variabili **DON'T CARE**

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

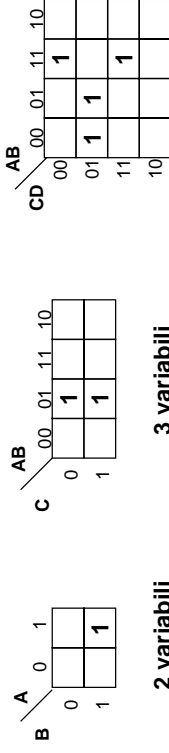
⇒

- $f = \sim A \sim B C \sim D + \sim A \sim B C D + \sim A B C \sim D + \sim A B C D$
- $\sim A C$  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:  
 $f = \sim A C$
- Infatti:  
 $f = \sim A C (\sim B \sim D + \sim B D + B \sim D + B D) = \sim A C (1) = \sim A C$

Arch. Elab. - S. Orlando 18

## 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



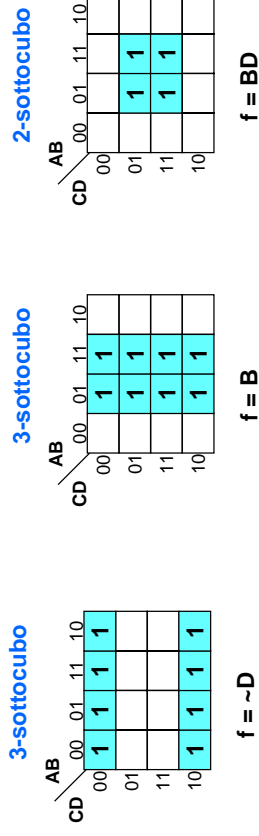
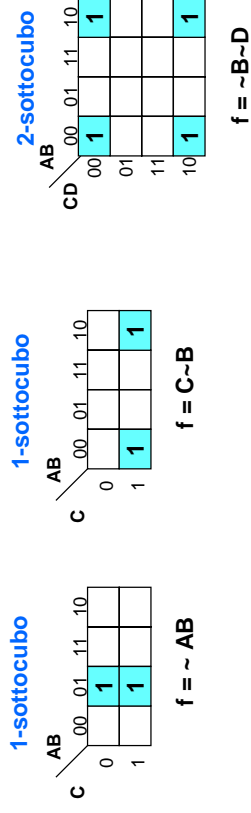
Arch. Elab. - S. Orlando 20

## 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<sup>p</sup> valori uguali ad 1**, e sono anche noti con il termine di **p-sottocubi**

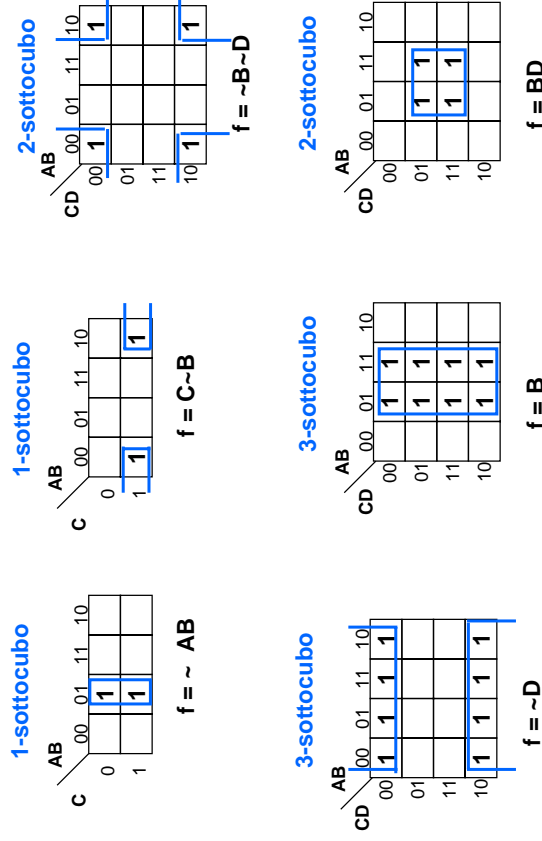
Arch. Elab. - S. Orlando 21

## Esempi di p-sottocubi



Arch. Elab. - S. Orlando 22

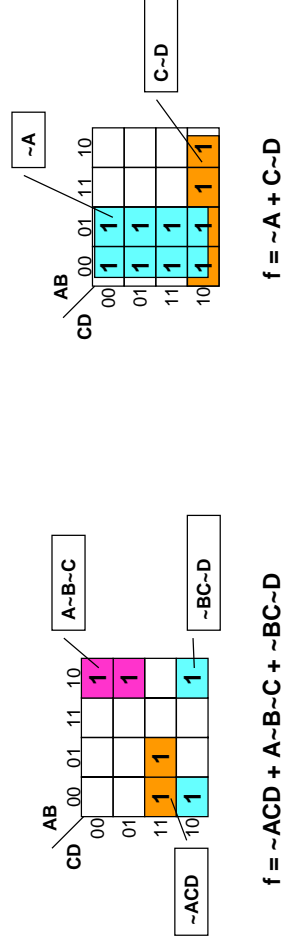
## Grafica differente per rappresentare p-sottocubi



Arch. Elab. - S. Orlando 23

## Minimizzazione con mappe di Karnaugh

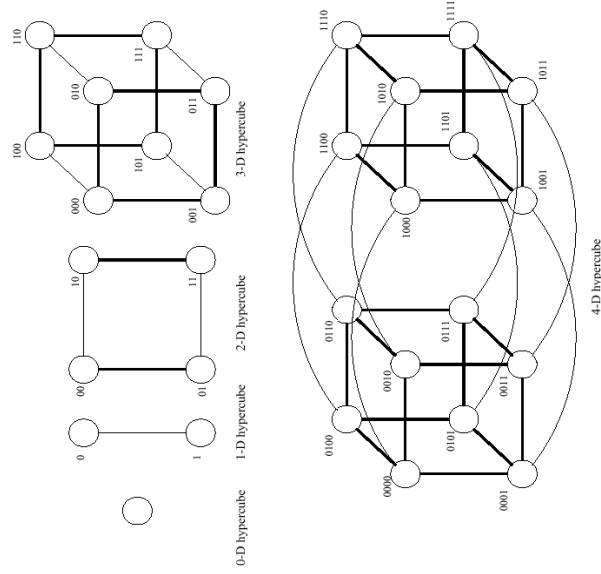
- Intuitivamente
  - per minimizzazione il più possibile, basta scegliere i **più grandi rettangoli** (p-sottocubi) che ricoprono gli 1 della mappa
  - **ATTENZIONE:** gli stessi 1 possono essere ricoperti da **più rettangoli** (da più p-sottocubi)



Arch. Elab. - S. Orlando 24



## Ipercubi e Mappe di Karnaugh

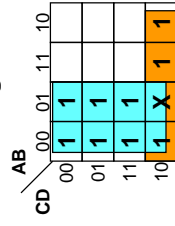


- Abbiamo definito alcuni gruppi di  $2^p$  celle delle mappe come **p-sottocubi**
- La **mappa di Karnaugh** è in effetti la **rappresentazione tabellare** di un grafo con topologia ad **ipercubo**
  - ogni **nodo** dell'ipercubo a **n dimensioni** è **etichettato** con un **numero binario a n cifre**
  - 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

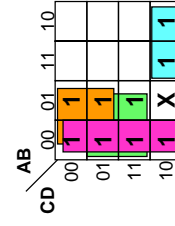
Arch. Elab. - S. Orlando 25

## Funzioni incomplete

- Alcuni output di una funzione, ovvero gli output corrispondenti a particolari configurazioni degli input, 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 **NON** possono essere incompleti
  - essi devono produrre un risultato in corrispondenza di **TUTTE** le combinazioni dei valori in input
  - TRUCCO**: 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:  
 $f = \sim A + C \sim D$

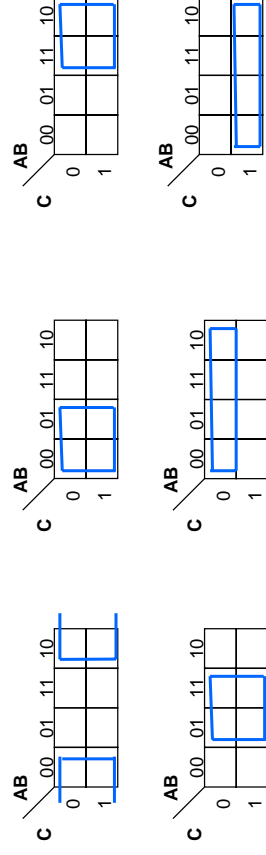
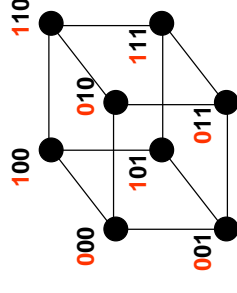


Considerando  $X=0$ , ben 4 p-sottocubi:  
 $f = \sim A \sim B + \sim A \sim C + \sim AD + AC \sim D$

Arch. Elab. - S. Orlando 27

## 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**



Arch. Elab. - S. Orlando 26