Architettura degli Elaboratori I

Corso di Laurea Triennale in Informatica

Università degli Studi di Milano

Dipartimento di Informatica "Giovanni Degli Antoni"

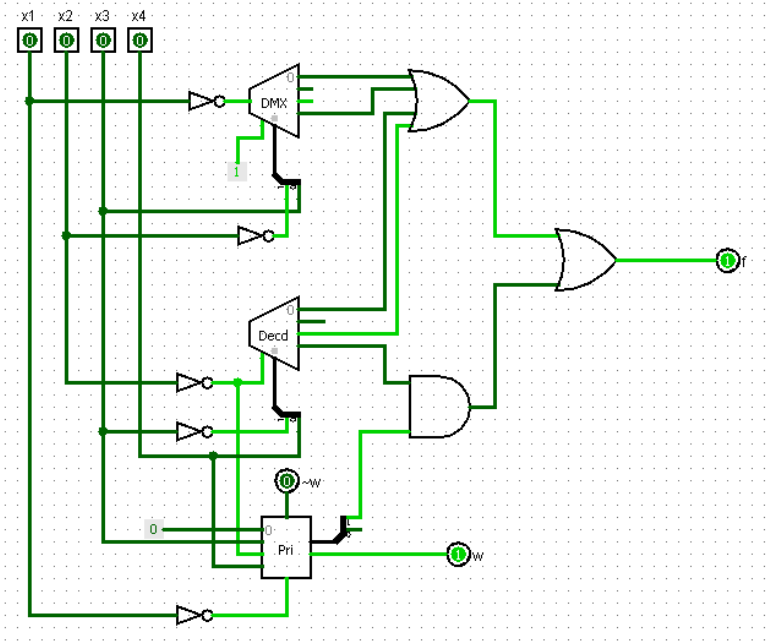
Edizione 2 (Cognomi H-Z), A.A. 2024-2025, Nicola.Basilico@unimi.it

Funzioni e porte logiche elementari

Introduzione

- Abbiamo visto come si rappresenta l'informazione numerica attraverso un sistema di numerazione binario
- Ora ci poniamo una seconda domanda: come si progetta un circuito in grado di rappresentare ed elaborare tale informazione?

- Un circuito digitale può essere pensato come composto da tanti elementi, ciascuno in grado di compiere una **elaborazione logica** elementare
- Connettendo tra loro tali elementi possiamo ottenere reti, o circuiti, in grado di svolgere elaborazioni più complesse
- Ma che cosa sono le elaborazioni logiche? Risposta: sono operazioni matematiche definite da un'algebra particolare detta **algebra di Boole**



Algebra di Boole

- Un’algebra può essere pensata come un insieme di **simboli**, **valori** e **regole** per svolgere operazioni su di essi
- L’algebra a cui siamo «normalmente» abituati usa valori e simboli numerici e operazioni come somma, sottrazione, moltiplicazione. Formulando sequenze di operazioni su simboli (variabili) si possono definire delle funzioni
- L’algebra di Boole comprende **simboli e valori binari** su cui possiamo svolgere **operazioni logiche**
  - variabili $a, x_1, x_2, u, \ldots$ ciascuna può avere valore TRUE (1) o FALSE (0)
  - i valori 1 e 0 e le variabili possono essere usate come operandi di tre operatori logici elementari: NOT, AND e OR
- Combinando valori e simboli binari con questi operatori possiamo definire **espressioni Booleane** che rappresentano **funzioni logiche**
  - Insieme dei valori binari $B = \{0,1\}$
  - Variabile binaria $a \in B$ (B è il dominio di ogni variabile in input)
  - Definizione di funzione logica su n variabili binarie $f: B^n \rightarrow B$ (si può anche indicare come $f(a_1, a_2, \ldots, a_n) \in B$)
- Come si usano e come funzionano gli operatori logici elementari?

Operatore logico $\mathbf{NOT}$

- $\mathbf{NOT}$: esprime la negazione logica di una espressione booleana
- Si indica con $\bar{a}$, dove a può essere una variabile o anche un’espressione booleana di più variabili
- Esistono notazioni alternative, ad esempio «$\mathrm{NOT}(a)$» e «$\neg a$»
- Interpretazione: se a ha valore 1 la sua negazione ha valore 0 e *vice versa*

a	$\bar{a}$
0	1
1	0

Operatore logico AND

- $\mathbf{AND}$: esprime la congiunzione logica tra due espressioni a e b
- Si indica con ab, ma esistono notazioni alternative, ad esempio «a and b», «$a \wedge b$»
- Se entrambe le espressioni, a e b, hanno valore 1 la loro congiunzione vale 1 altrimenti ha valore 0

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

- Anche detto «prodotto logico», conviene pensarlo come un $\min$

Operatore logico $\mathbf{OR}$

- $\mathbf{OR}$: esprime la disgiunzione logica tra due espressioni a e b
- Si indica con $a + b$, ma esistono notazioni alternative, ad es. «a or b» e «$a \vee b$»
- Se almeno una delle due espressioni, a o b, ha valore 1, la loro disgiunzione vale 1 altrimenti ha valore 0

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

- Anche detto «somma logica», conviene pensarlo come un $\max$

Precedenza tra operatori

- Quando scriviamo espressioni Booleane usando questi operatori logici dobbiamo ricordarci delle **regole di precedenza** con cui si svolgono le operazioni:
  - NOT ha precedenza su AND e su OR
  - AND ha precedenza su OR
- Esempio di funzione logica su tre variabili:

$$f(a, b, c) = a + \bar{b}c$$

- Esplicitando le precedenze con delle parentesi si ha:

$$f(a, b, c) = (a + ((\bar{b})c))$$

Il principio di dualità

- Data un'espressione di uguaglianza booleana, la sua **duale** si ottiene scambiando
  1. gli AND con gli OR e *vice versa*
  2. gli 1 con gli 0 e *vice versa*
- **Principio di dualità**: se un'uguaglianza booleana è valida allora lo è anche la sua duale
- **Esempi**:
  - Espressione: $a + \bar{a} = 1$, la sua duale: $a\bar{a} = 0$
  - Espressione: $(a\,\bar{b}) + 1 = 1$, la sua duale: $\left(a + \bar{b}\right)0 = 0$

Proprietà degli operatori logici

Proprietà	AND	OR
Identità	$1a = a$	$0 + a = a$
Elemento nullo	$0a = 0$	$1 + a = 1$
Idempotenza	$aa = a$	$a + a = a$
Inverso	$a\bar{a} = 0$	$a + \bar{a} = 1$
Commutativa	$ab = ba$	$a + b = b + a$
Associativa	$(ab)c = a(bc)$	$(a + b) + c = a + (b + c)$

Proprietà	di AND rispetto ad OR	di OR rispetto ad AND
Distributiva	$a(b + c) = ab + ac$	$a + bc = (a + b)(a + c)$
Assorbimento I	$a(a + b) = a$	$a + ab = a$
Assorbimento II	$a(\bar{a} + b) = ab$	$a + \bar{a}b = a + b$
De Morgan	$\overline{ab} = \bar{a} + \bar{b}$	$\overline{a + b} = \bar{a}\,\bar{b}$

- Identità, Inverso, Commutativa e Distributiva sono **postulati**: si assumono essere vere
- Le altre si dimostrano a partire dai postulati applicando le regole dell'algebra

Dimostrazione di alcune proprietà

Ipotesi: Dualità, Postulati
Tesi: Idempotenza, $aa = a$

Passaggio	Ottenuto grazie a…
a	
$= a + 0$	Identità $(0 + a = a)$
$= a + a\bar{a}$	Inverso $(a\bar{a} = 0)$
$= (a + a)(a + \bar{a})$	Distributiva
$= (a + a)1$	Inverso $(a + \bar{a} = 1)$
$= a + a$	Identità $(1a = a)$
$= aa$	Dualità

Dimostrata proprietà di idempotenza!

Ipotesi: Dualità, Postulati, Idempotenza
Tesi: Elemento nullo, $1 + a = 1$

Passaggio	Ottenuto grazie a…
$1 + a$	
$= a + \bar{a} + a$	Inverso $(a + \bar{a} = 1)$
$= a + \bar{a}$	Idempotenza $(a + a = a)$
$= 1$	Inverso $(a + \bar{a} = 1)$

Dimostrata proprietà dell'elemento nullo!

Ipotesi: Dualità, Postulati, Idempotenza, El. nullo
Tesi: Assorbimento I, $a(a + b) = a$

Passaggio	Ottenuto grazie a…
$a(a + b)$	
$= aa + ab$	Distributiva
$= a + ab$	Idempotenza $(aa = a)$
$= a1 + ab$	Identità $(1a = a)$
$= a(1 + b)$	Distributiva
$= a$	El. nullo $(1 + a = 1)$

Dimostrata proprietà dell'assorbimento I!

Applicazione delle proprietà

- **Esercizio**: semplificare la seguente espressione logica: $\overline{\bar{a}+c}+\bar{c}+b\left(\overline{c+c\,\bar{b}}\right)$

Passaggio	Ottenuto grazie a…
$\overline{\bar{a}+c}+\bar{c}+b\left(\overline{c+c\,\bar{b}}\right)$	
$=\bar{\bar{a}}\,\bar{c}+\bar{c}+b(\overline{c+c\,\bar{b}})$	De Morgan
$=\bar{c}+b(\overline{c+c\,\bar{b}})$	Assorbimento I ($a+ab=a$)
$=\bar{c}+b\overline{c(1+\bar{b})}$	Distributiva
$=\bar{c}+b\,\bar{c}$	El. nullo e identità
$=\bar{c}$	Assorbimento I

Applicazione delle proprietà

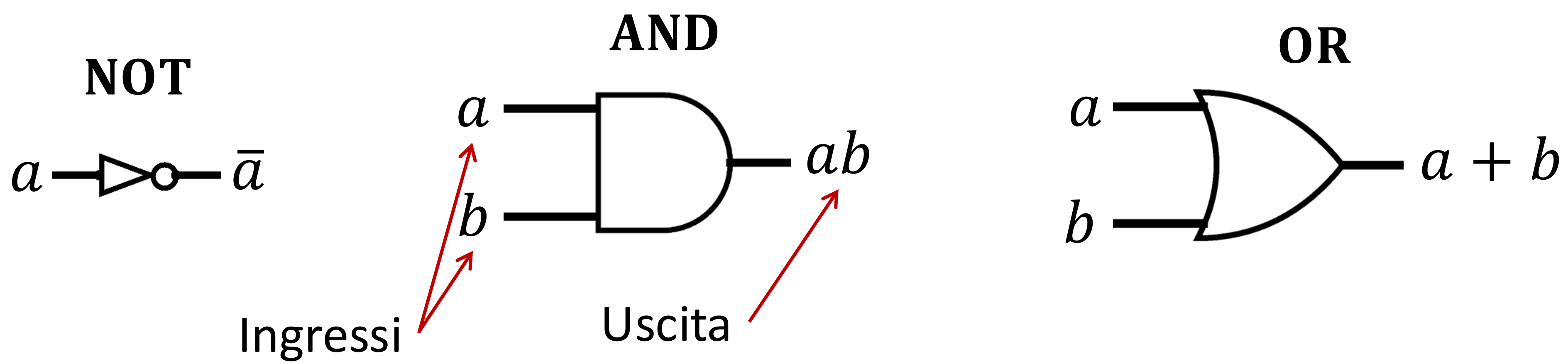
- **Esercizio**: semplificare la seguente espressione logica: $\left(\overline{\overline{\bar{a}b}} + \bar{c}\right) \overline{\bar{a}b} + \overline{\bar{a}b}c$

Passaggio	Ottenuto grazie a…
$\left(\overline{\overline{\bar{a}b}} + \bar{c}\right) \overline{\bar{a}b} + \overline{\bar{a}b}c$	
$= \overline{\bar{a}b}\bar{c} + \overline{\bar{a}b}c$	Assorbimento II $a(\bar{a} + b) = ab$
$= \overline{\bar{a}b}(\bar{c} + c)$	Distributiva
$= \overline{\bar{a}b}$	El. inverso
$= \bar{\bar{a}} + \bar{b}$	De Morgan
$= a + \bar{b}$	

Porte logiche

Porte logiche

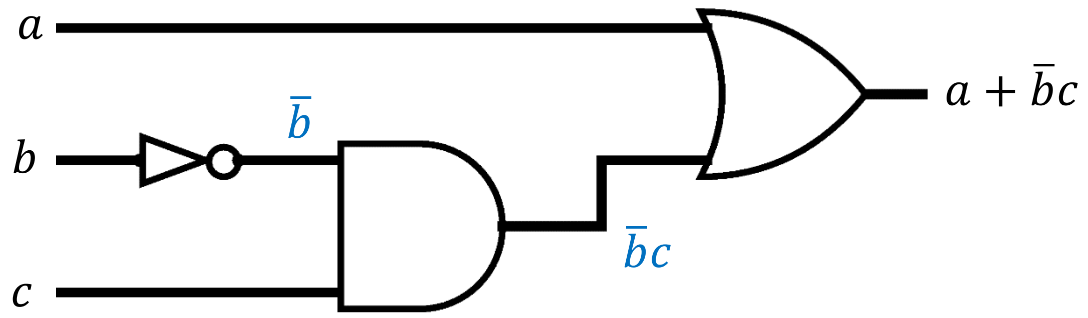
- Oltre alla loro espressione Booleana, i tre operatori logici possiedono delle «controparti hardware»: un circuito digitale elementare che svolge sui i segnali di tensione la stessa funzione che l'operatore logico svolge nell'Algebra di Boole sui valori logici
- Questi elementi si chiamano **porte logiche** e si indicano graficamente con questi simboli



Circuiti combinatori

- Così come possiamo usare i tre operatori logici per creare funzioni, possiamo analogamente combinare le porte logiche connettendole tra di loro, collegando uscite con ingressi

Esempio: $f(a,b,c) = a + \bar{b}c$ ⇨

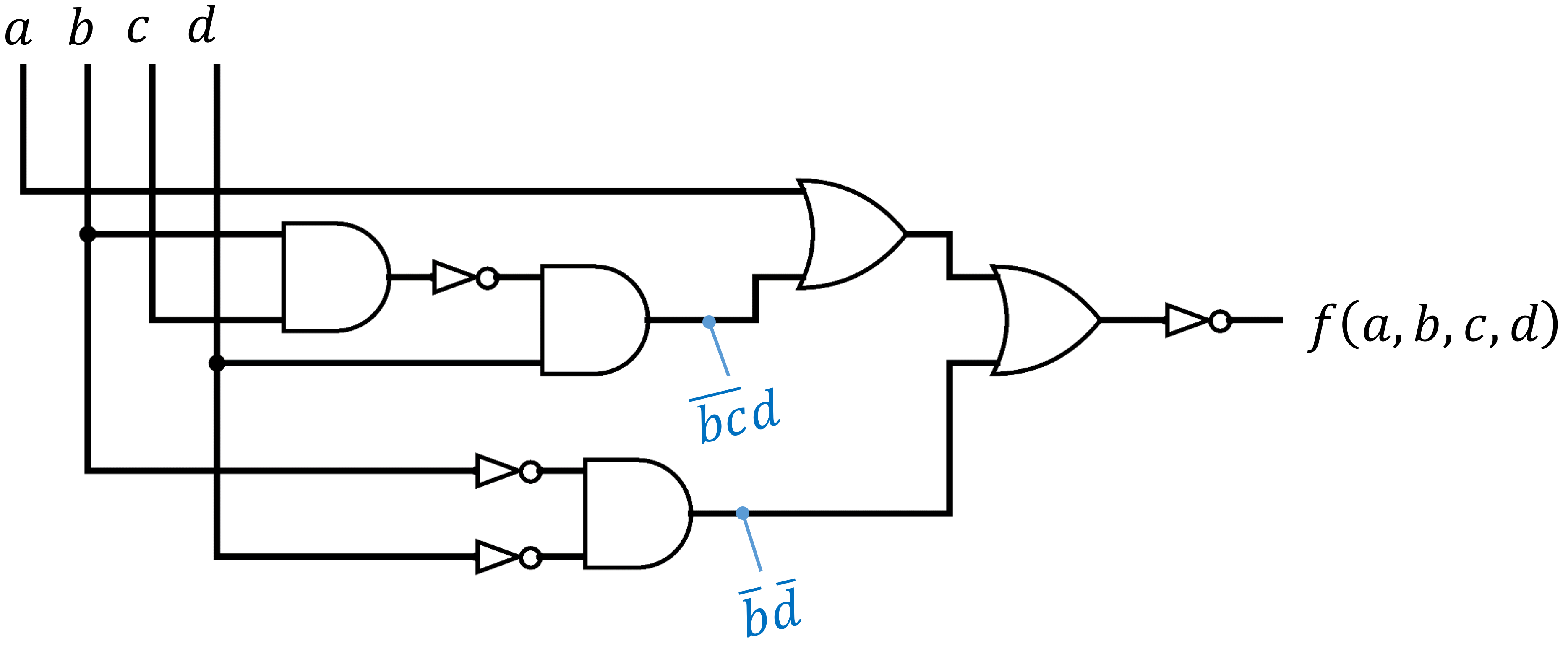


- Abbiamo progettato il nostro primo circuito digitale!
- Nello specifico, è un **circuito combinatorio**: «combina» gli input (che rappresentano le variabili binarie a,b, e c) per ottenere un output (che rappresenta la funzione logica $f(a,b,c)$)
- I circuiti combinatori hanno due proprietà fondamentali (che sono in realtà legate tra loro):
  1. L'elaborazione procede in un senso solo: da sinistra a destra
  2. L'uscita dipende solo dagli input: **a parità di input l'uscita è sempre la stessa** (è un circuito senza memoria, come vedesse sempre gli input per la prima volta!)

Circuiti combinatori

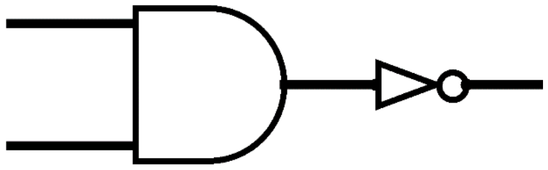
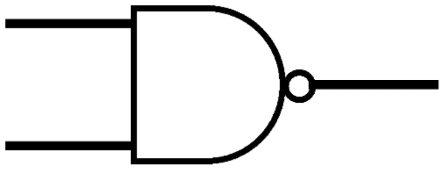
- **Esercizio**: progettare il circuito che implementa la seguente funzione logica:

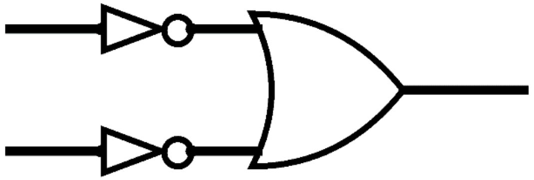
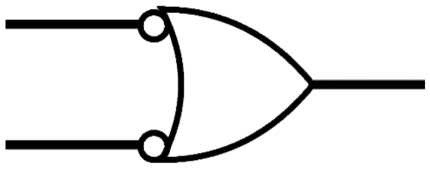
$$f(a,b,c,d) = \overline{a + \overline{bc}d + \bar{b}\bar{d}}$$



Circuiti combinatori

- Notazione grafica compatta: quando il NOT (l’inverter) si trova su un ingresso o su una uscita di una porta logica può essere denotato con un **pallino** su quell’ingresso o uscita
- Ad esempio:

Questo:  si rappresenta, in modo compatto, così: 

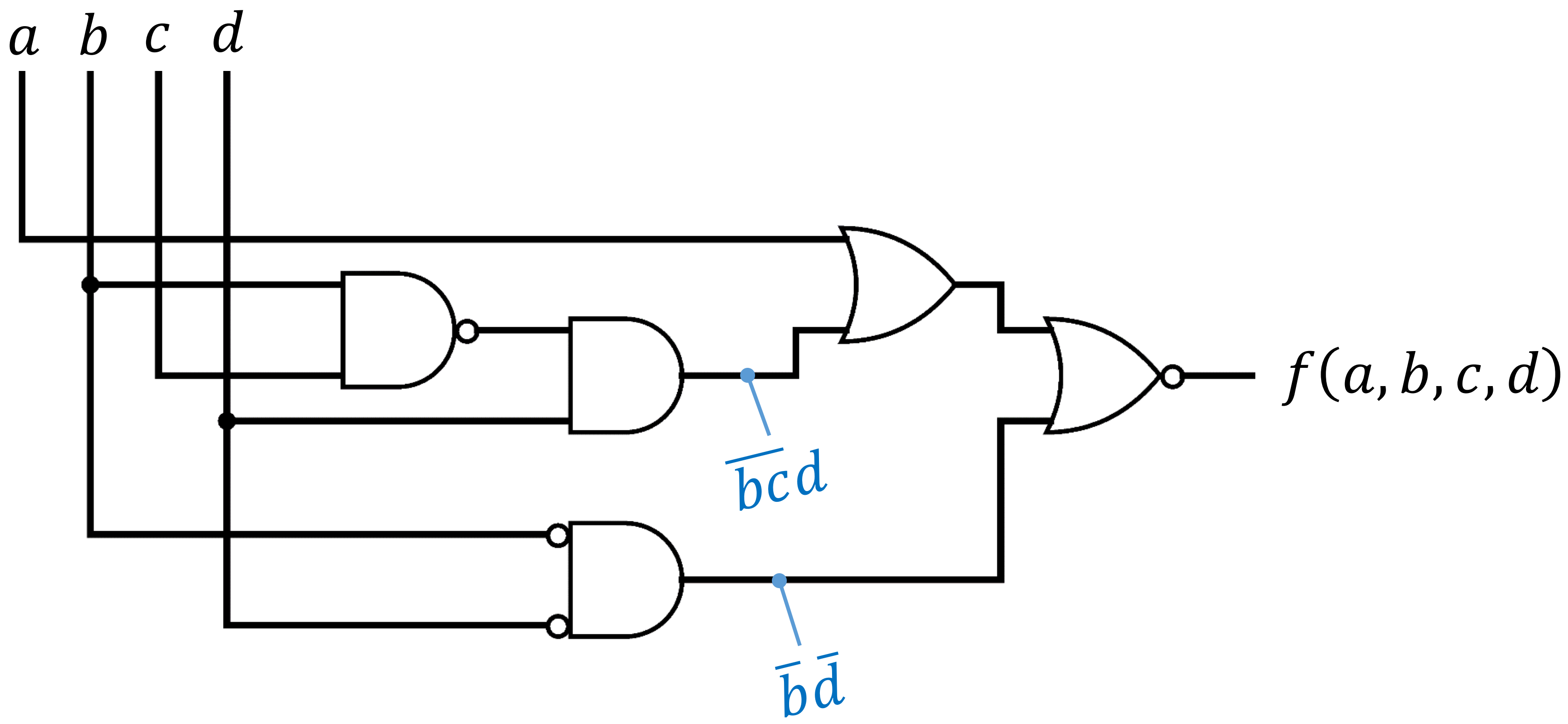
Questo:  si rappresenta, in modo compatto, così: 

- Rivediamo l’esempio dell’esercizio precedente …

Circuiti combinatori

- **Esercizio**: progettare il circuito che implementa la seguente funzione logica:

$$f(a,b,c,d) = \overline{a + \overline{bc}d + \bar{b}\bar{d}}$$



Metodi per rappresentare una funzione logica

Rappresentare una funzione logica

- In generale data una funzione logica f definita nell'algebra Booleana, abbiamo tre modi di rappresentarla (li abbiamo già incontrati tutti e tre nelle slide precedenti)
    1. **Espressione Booleana**
    2. **Circuito combinatorio**
    3. **La tabella di verità**: una tabella che ha una riga per ogni possibile combinazione di valori di input e che specifica, su ogni riga, il valore della funzione nella configurazione corrispondente

Rappresentare una funzione logica

- Consideriamo la funzione $f(a,b,c) = a + \bar{b}c$

Espressione Booleana

$$a + \bar{b}c$$

Circuito combinatorio

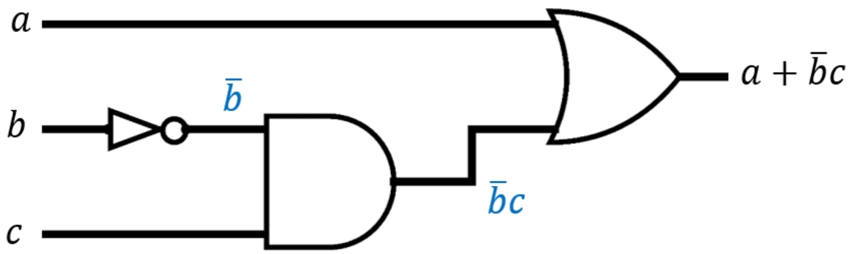


Tabella di verità

a	b	c	$a + \bar{b}c$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Rappresentare una funzione logica

Data una funzione $f: B^n \to B$

- **La tabella di verità** è una descrizione esaustiva, ha 2^n righe e $n + 1$ colonne, una funzione ne ammette una sola
- **L'espressione booleana** è una descrizione formale che possiamo leggere facilmente e manipolare con le regole dell'Algebra di Boole, una funzione ne ammette infinite
- **Il circuito combinatorio** è l'implementazione hardware della funzione: una macchina che rappresenta input e output con dei segnali di tensione e che associa a un dato input un output che rappresenta il valore della funzione in quell'input, una funzione ne ammette infiniti

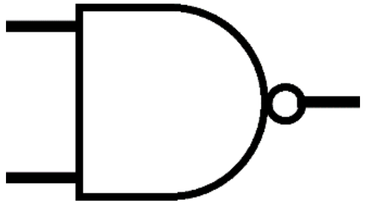
Operatori composti

Operatori composti

- NOT, AND e OR sono gli operatori logici elementari, il mezzo primario con cui costruire funzioni logiche
- Esistono anche degli **operatori composti**: sono più complessi ma hanno delle proprietà interessanti, per questo motivo risulta conveniente definirli e rappresentarli con una porta logica associata

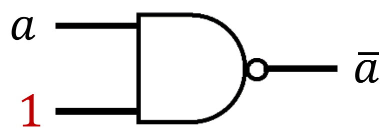
Operatori composti: NAND

- NAND («Not AND»): è un AND negato, quindi con un NOT all'uscita
- È l'«opposto» di AND, vale 0 solo quando entrambi gli input sono pari a 1
- **Completezza funzionale**: NOT, AND e OR possono essere implementati con la sola porta NAND, per questo la si chiama «**porta universale**»

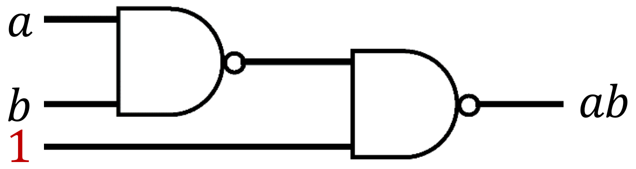


a	b	$\overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0

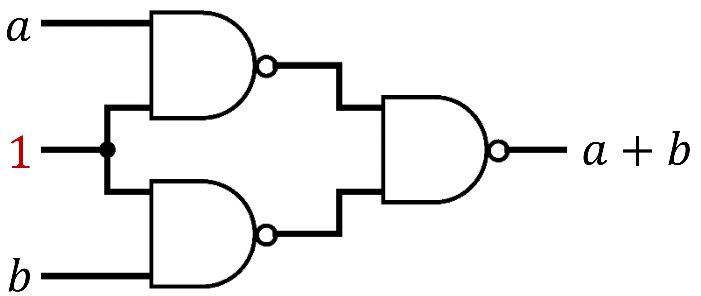
$$\bar{a} = \overline{1a}$$



$$ab = \overline{\overline{ab}} = \overline{\overline{ab}\;1}$$

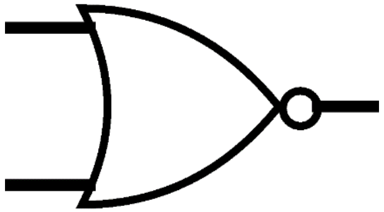


$$a + b = \overline{\bar{a}\,\bar{b}} = \overline{\overline{1a}\;\overline{1b}}$$



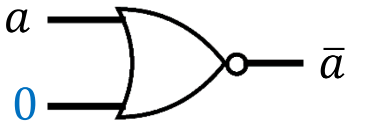
Operatori composti: **NOR**

- NOT («Not OR»): è un OR negato, quindi con un NOT all'uscita
- È l'«opposto» di OR vale 1 solo quando entrambi gli input sono pari a 0
- **Completezza funzionale**: NOT, AND e OR possono essere implementati con la sola porta NOR , per questo la si chiama «**porta universale**»

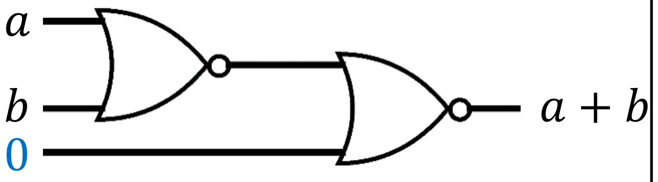


a	b	$\overline{a+b}$
0	0	1
0	1	0
1	0	0
1	1	0

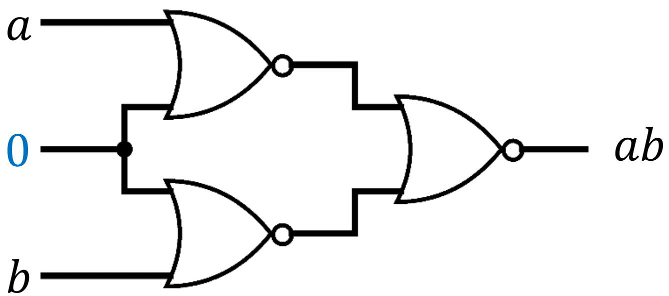
$$\bar{a} = \overline{0 + a}$$



$$a + b = \overline{\overline{a+b}} = \overline{0 + \overline{a+b}}$$



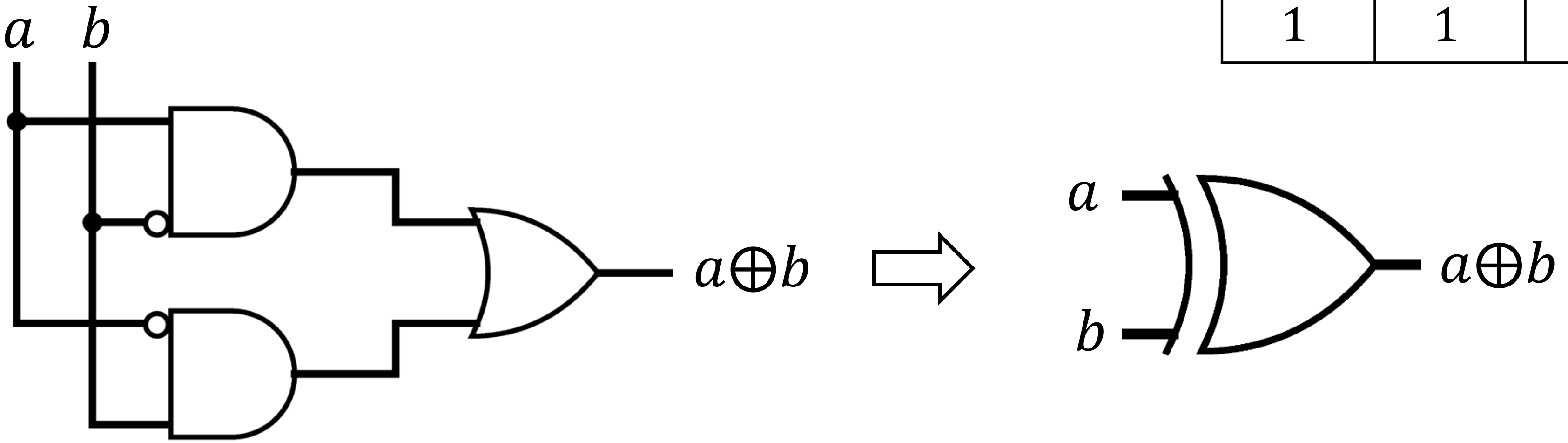
$$ab = \overline{\bar{a} + \bar{b}} = \overline{\overline{0+a} + \overline{0+b}}$$



Operatori composti: XOR

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

- XOR («eXclusive OR»): è un OR esclusivo, operatore Booleano $\oplus$
- vale 1 solo quando **uno e uno solo** degli input è pari a 1
- Si esprime con la seguente espressione booleana: $a\bar{b} + \bar{a}b$

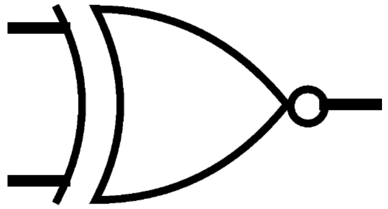


- XOR ha tre interpretazioni:
  - **Funzione di diversità**: vale 1 quando i bit sono diversi
  - Circuito per il **complemento a 1** di un bit: a è il dato in input, b un segnale di controllo. Se $b = 0$ il dato passa verso l'uscita inalterato, altrimenti viene fatto il complemento a 1
  - *(la terza la vedremo più avanti)*

Operatori composti: XNOR

- XOR («Not XOR»): è uno XOR negato
- vale 1 solo quando gli input **sono uguali**: **funzione di uguaglianza**

a	b	$\overline{a \oplus b}$
0	0	1
0	1	0
1	0	0
1	1	1

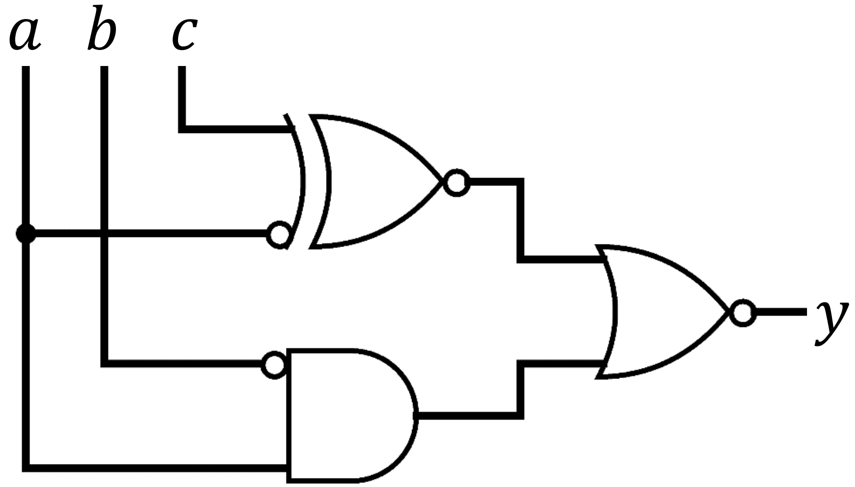


Analisi e sintesi di un circuito: forme canoniche

Analisi di un circuito combinatorio

- **Analisi di un circuito**: a partire dal circuito o dalla espressione della funzione logica, costruisco la tabella di verità, determinando il valore dell'uscita (o dell'espressione) a fronte di ogni possibile configurazione di input

- **Esercizio**:



Espressione?

$$y = \overline{\overline{\overline{a}\oplus c} + a\overline{b}}$$

Tabella di verità?

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Sintesi di un circuito combinatorio

- **Sintesi di un circuito**: a partire dalla tabella di verità o dall'espressione Booleana **costruiamo il circuito che la implementa**
- E' la parte più interessante! La tabella di verità o l'espressione possono essere pensate come una specifica, il circuito è l'implementazione: la macchina che calcola quella funzione

Prima forma canonica

- Idea: descriviamo la funzione partendo dalla sua tabella e indicando **tutti** i punti in cui ha valore 1
- È una descrizione completa? Sì! **Ipotesi del mondo chiuso**: ciò che non ha valore 1 (che non ho indicato) ha valore 0

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

y vale 1 se e soltanto se:

$a = 0, b = 0$ e $c = 0$

oppure

$a = 0, b = 1$ e $c = 0$

oppure

$a = 1, b = 0$ e $c = 1$

oppure

$a = 1, b = 1$ e $c = 1$

y vale 1 se e soltanto se (dualità):

$\bar{a} = 1, \bar{b} = 1$ e $\bar{c} = 1$ $\Rightarrow$ $\bar{a}\bar{b}\bar{c}$

oppure

$\bar{a} = 1, b = 1$ e $\bar{c} = 1$ $\Rightarrow$ $\bar{a}b\bar{c}$

oppure

$a = 1, \bar{b} = 1$ e $c = 1$ $\Rightarrow$ $a\bar{b}c$

oppure

$a = 1, b = 1$ e $c = 1$ $\Rightarrow$ abc

$$\Rightarrow \quad y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c + abc$$

- È un modo talmente naturale di descrivere una funzione che prende il nome di **prima forma canonica**
- È anche un procedimento meccanico che possiamo applicare a qualsiasi funzione

Prima forma canonica

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$\Rightarrow$ $y = \underline{\bar{a}\bar{b}\bar{c}} + \underline{\bar{a}b\bar{c}} + \underline{a\bar{b}c} + \underline{abc}$

Mintermini:

- Definiti come AND tra tutte le variabili (input) della funzione, ogni variabile compare una volta sola nella sua forma naturale o negata
- Ne abbiamo uno per ogni configurazione di input in cui la funzione vale 1
- Se nella configurazione di input corrispondente la variabile vale 0, nel mintermine comparirà negata, altrimenti in forma naturale
- Chiamando m_i l'i-esimo mintermine, una funzione y può essere sempre espressa come l'OR tra tutti i suoi n mintermini: $y = \sum_{i=1}^{n} m_i$

- Prima forma canonica: **somma di prodotti** (mintermini), Sum Of Products (**SOP**)

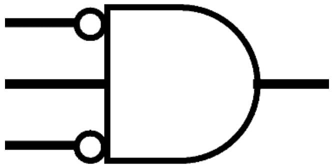
Prima forma canonica

- Scrittura dell'espressione booleana
  1. Identificare i mintermini della funzione
  2. Combinarli con un OR

$$y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c + abc$$

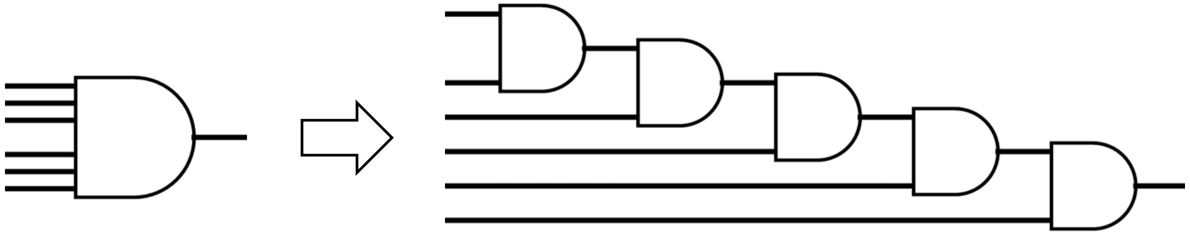
- Sintesi del circuito
  - Ogni mintermine corrisponde ad un AND a più ingressi (tanti quante sono le variabili)

$\bar{a}b\bar{c}$ ⇨



Buona pratica: collegare sempre gli input in ordine alfabetico seguendo lo stesso verso (dall'alto al basso, da sinistra a destra): guardando la porta logica capiamo subito l'espressione del mintermine associato!

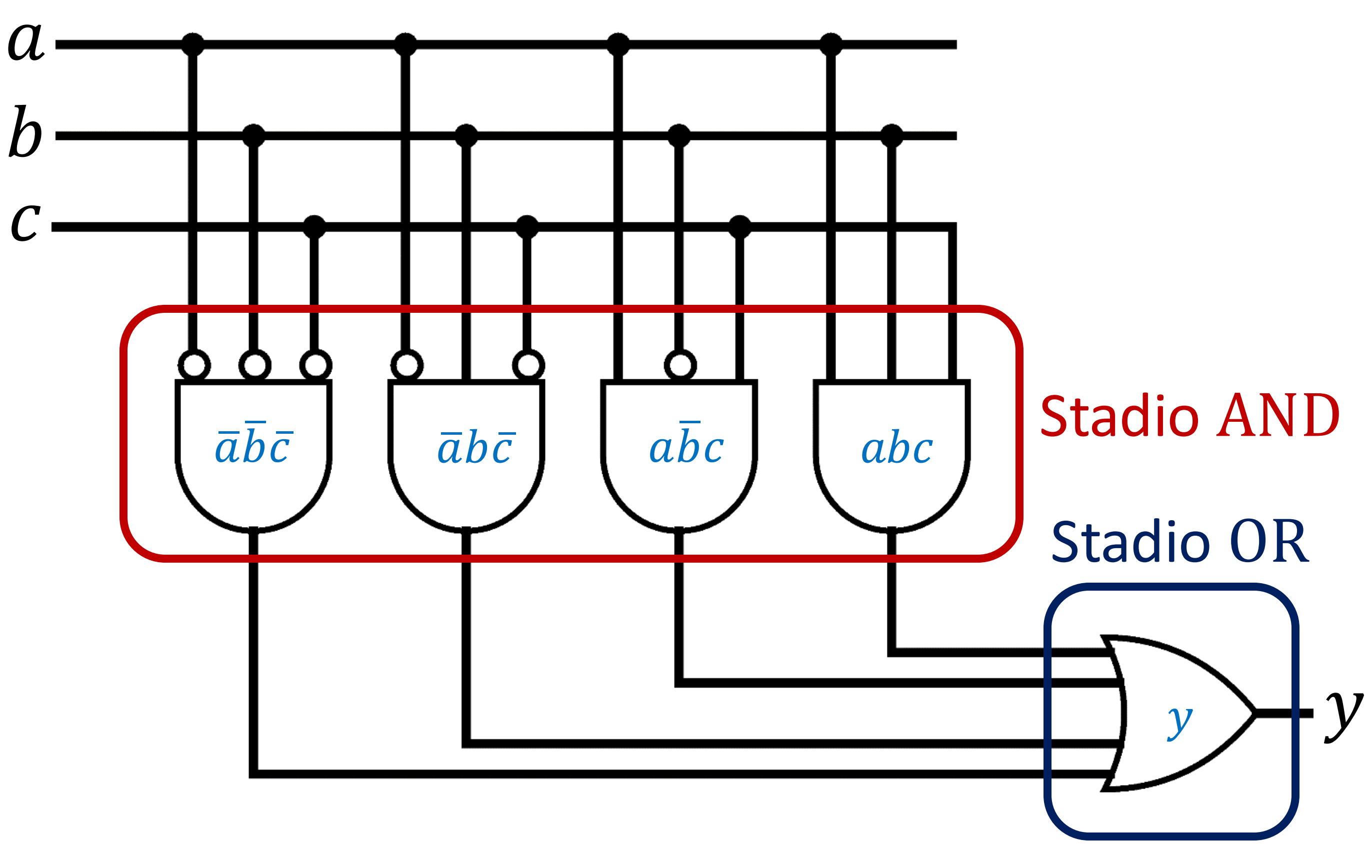
Nota: Le porte a n ingressi si ottengono collegando in serie $n - 1$ porte a 2 ingressi



Prima forma canonica

- Sintesi del circuito: schema a due stadi
    1. Stadio AND: una porta AND per ogni mintermine
    2. Stadio OR: un OR tra tutte le uscite dello stadio AND
- Anche la sintesi del circuito, come la scrittura dell'espressione Booleana, è un procedimento meccanico: lo stesso per ogni funzione logica
- E la forma più compatta? **Probabilmente no!** Semplificando l'espressione potremmo sintetizzare un circuito più semplice

$$y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c + abc$$

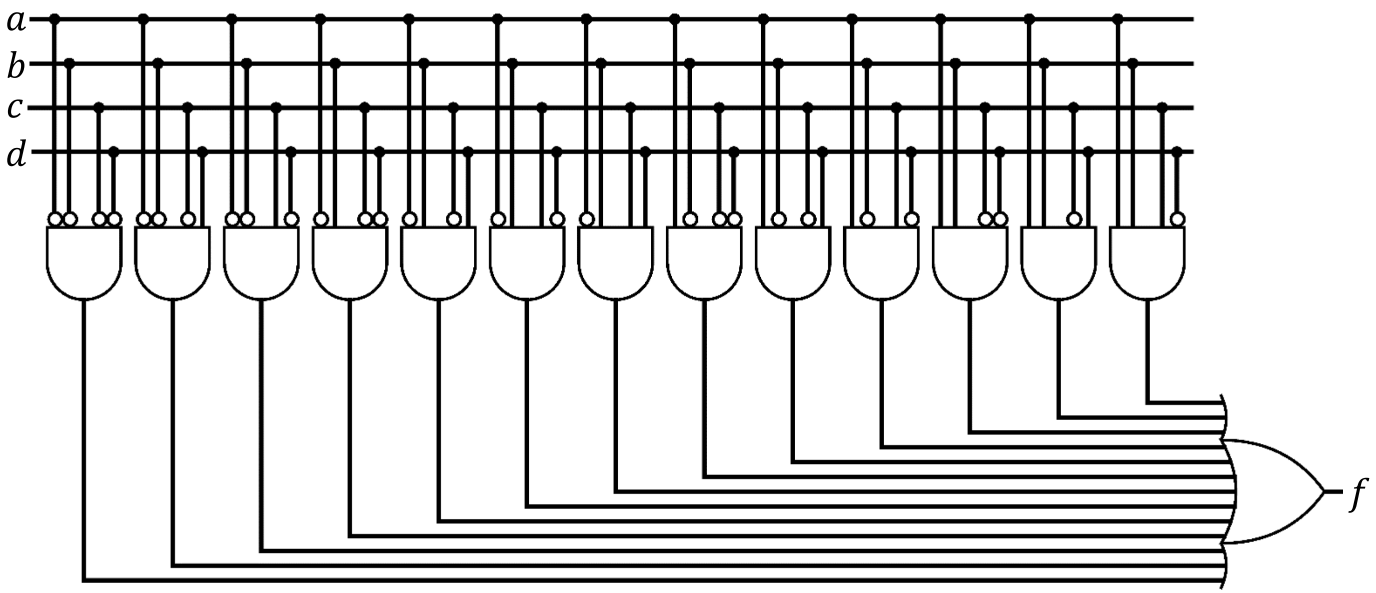


Prima forma canonica

- **Esercizio**: sintetizzare il circuito che implementa la prima forma canonica di $f(a,b,c,d) = \bar{a}b + \overline{cd}$
  1. Determino la tabella di verità
  2. Indentifico i mintermini
  3. Sintetizzo espressione Booleana e circuito combinatorio

a	b	c	d	f	
0	0	0	0	1	$\bar{a}\bar{b}\bar{c}\bar{d}$
0	0	0	1	1	$\bar{a}\bar{b}\bar{c}d$
0	0	1	0	1	$\bar{a}\bar{b}c\bar{d}$
0	0	1	1	0	
0	1	0	0	1	$\bar{a}b\bar{c}\bar{d}$
0	1	0	1	1	$\bar{a}b\bar{c}d$
0	1	1	0	1	$\bar{a}bc\bar{d}$
0	1	1	1	1	$\bar{a}bcd$
1	0	0	0	1	$a\bar{b}\bar{c}\bar{d}$
1	0	0	1	1	$a\bar{b}\bar{c}d$
1	0	1	0	1	$a\bar{b}c\bar{d}$
1	0	1	1	0	
1	1	0	0	1	$ab\bar{c}\bar{d}$
1	1	0	1	1	$ab\bar{c}d$
1	1	1	0	1	$abc\bar{d}$
1	1	1	1	0	

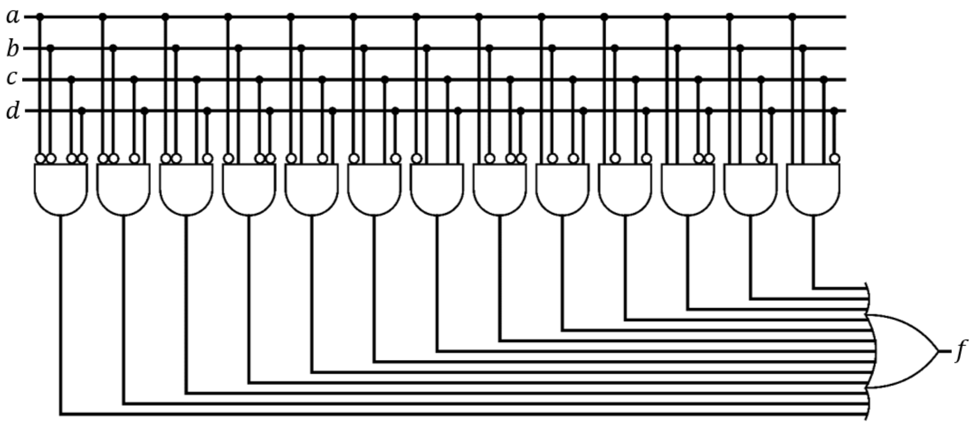
$$f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd$$
$$+ a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + ab\bar{c}\bar{d} + ab\bar{c}d + abc\bar{d}$$



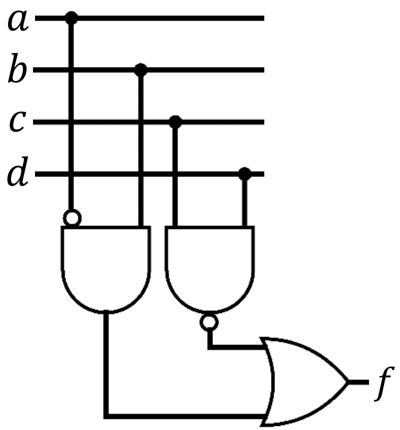
Prima forma canonica

- Se anziché usare la forma canonica avessi usato la forma semplificata?

$$f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd \\ + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + ab\bar{c}\bar{d} + ab\bar{c}d + abc\bar{d}$$



$$f(a,b,c) = \bar{a}b + \overline{cd}$$



- La SOP è più grande di come sembra! Ricordiamoci che una porta con n input si implementa come $n-1$ porte da 2 input collegate in serie
- La SOP che abbiamo qui richiederebbe $13 \times 3 = 39$ porte per lo stadio AND e 12 porte per lo stadio OR, in totale 41 porte!

Seconda forma canonica

- Grazie alla **ipotesi del mondo chiuso** posso fornire una descrizione completa della funzione logica indicando tutti i punti in cui ha valore 0, ciò che non indico avrà implicitamente valore 1

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

y vale 0 se e soltanto se:

$a = 0, b = 0$ e $c = 1$

oppure

$a = 0, b = 1$ e $c = 1$

oppure

$a = 1, b = 0$ e $c = 0$

oppure

$a = 1, b = 1$ e $c = 0$

y vale 0 se e soltanto se:

$\bar{a} = 1, \bar{b} = 1$ e $c = 1$ $\Rightarrow$ $\bar{a}\bar{b}c$

oppure

$\bar{a} = 1, b = 1$ e $c = 1$ $\Rightarrow$ $\bar{a}bc$

oppure

$a = 1, \bar{b} = 1$ e $\bar{c} = 1$ $\Rightarrow$ $a\bar{b}\bar{c}$

oppure

$a = 1, b = 1$ e $\bar{c} = 1$ $\Rightarrow$ $ab\bar{c}$

- $y = \overline{\bar{a}\bar{b}c + \bar{a}bc + a\bar{b}\bar{c} + ab\bar{c}}$ (devo negare tutto per riportarmi verso la descrizione di verità)
- Applicando De Morgan ottengo $y = (a + b + \bar{c})(a + \bar{b} + \bar{c})(\bar{a} + b + c)(\bar{a} + \bar{b} + c)$
- È la **seconda forma canonica**, ottenuta con un ragionamento duale rispetto alla prima
- Anche in questo caso è un procedimento meccanico che possiamo applicare a qualsiasi funzione

Seconda forma canonica

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$\Rightarrow \quad y = (\underline{a + b + \bar{c}})(\underline{a + \bar{b} + \bar{c}})(\underline{\bar{a} + b + c})(\underline{\bar{a} + \bar{b} + c})$$

Maxtermini:

- Definiti come OR tra tutte le variabili (input) della funzione, ogni variabile compare una volta sola nella sua forma naturale o negata
- Ne abbiamo uno per ogni configurazione di input in cui la funzione vale 0
- Se nella configurazione di input corrispondente la variabile vale 1, nel maxtermine comparirà negata, altrimenti in forma naturale
- Chiamando M_i l'i-esimo maxtermine, una funzione y può essere sempre espressa come l'AND tra tutti i suoi n maxtermini: $y = \prod_{i=1}^{n} M_i$

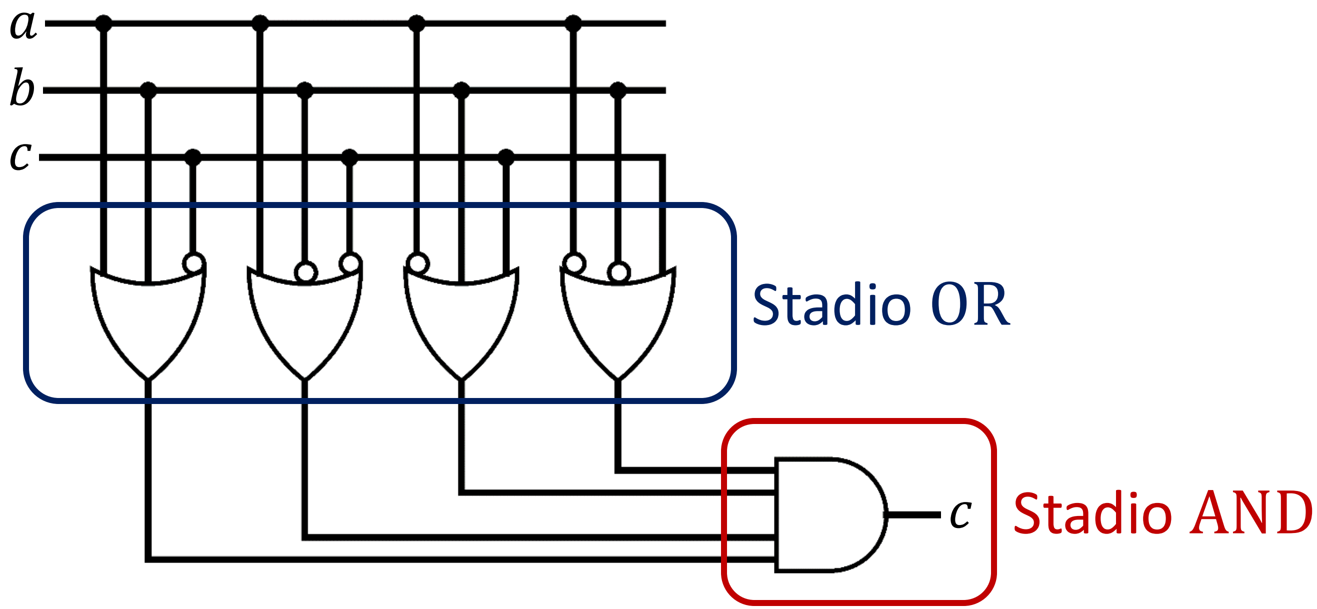
- Seconda forma canonica: **prodotto di somme** (maxtermini), Product of Sums (**POS**)

Seconda forma canonica

- Scrittura dell'espressione booleana
  1. Identificare i maxtermini della funzione
  2. Combinarli con un AND

$$y = (a + b + \bar{c})(a + \bar{b} + \bar{c})(\bar{a} + b + c)(\bar{a} + \bar{b} + c)$$

- Sintesi del circuito
  - Ogni maxtermine corrisponde ad un OR a più ingressi (tanti quante sono le variabili)
  - Valgono le stesse considerazioni che abbiamo per la SOP

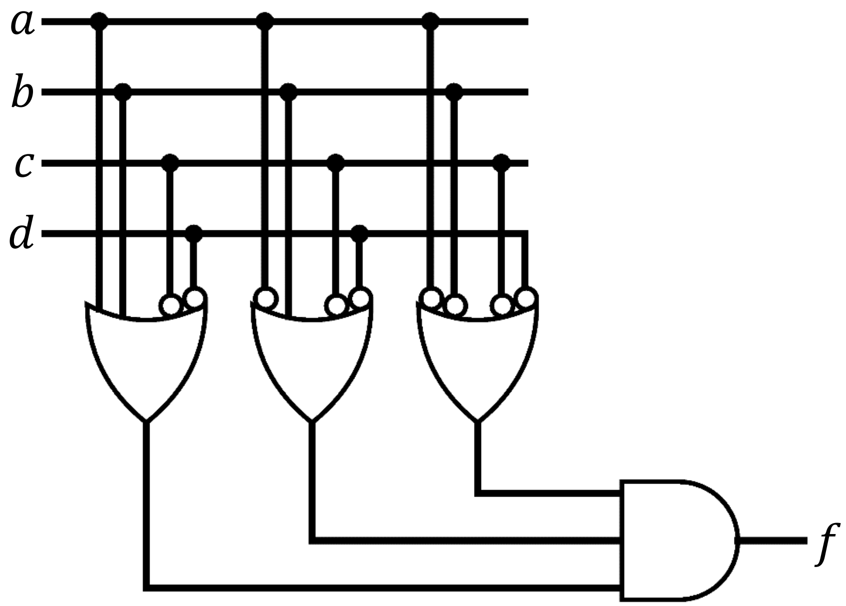


Seconda forma canonica

- **Esercizio**: sintetizzare il circuito che implementa la seconda forma canonica di $f(a,b,c) = \bar{a}b + \overline{cd}$
  1. Determino la tabella di verità
  2. Indentifico i maxtermini
  3. Sintetizzo espressione Booleana e circuito combinatorio

a	b	c	d	f	
0	0	0	0	1	
0	0	0	1	1	
0	0	1	0	1	
0	0	1	1	0	$\Rightarrow a + b + \bar{c} + \bar{d}$
0	1	0	0	1	
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	1	
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	0	$\Rightarrow \bar{a} + b + \bar{c} + \bar{d}$
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	0	$\Rightarrow \bar{a} + \bar{b} + \bar{c} + \bar{d}$

$$f = (a + b + \bar{c} + \bar{d})(\bar{a} + b + \bar{c} + \bar{d})(\bar{a} + \bar{b} + \bar{c} + \bar{d})$$



Implicanti

- Questa espressione $f(a, b, c, d) = \bar{a}b + \overline{cd}$ non rappresenta una forma canonica
- Il termine $\bar{a}b$ è un prodotto ma non un mintermine perché non compaiono tutte le variabili
- Se una funzione è una somma di prodotti, quei prodotti che non includono tutte le variabili si chiamano **implicanti**
- Gli implicanti «sintetizzano» somme di mintermini perché è come se dicessero che le variabili non specificate possono assumere qualsiasi valore
- **Esempio**: l'implicante $\bar{a}b$ sintetizza la somma $\bar{a}bcd + \bar{a}bc\bar{d} + \bar{a}b\bar{c}d + \bar{a}b\bar{c}\bar{d}$ (per dimostrare l'equivalenza basta applicare la proprietà distributiva)

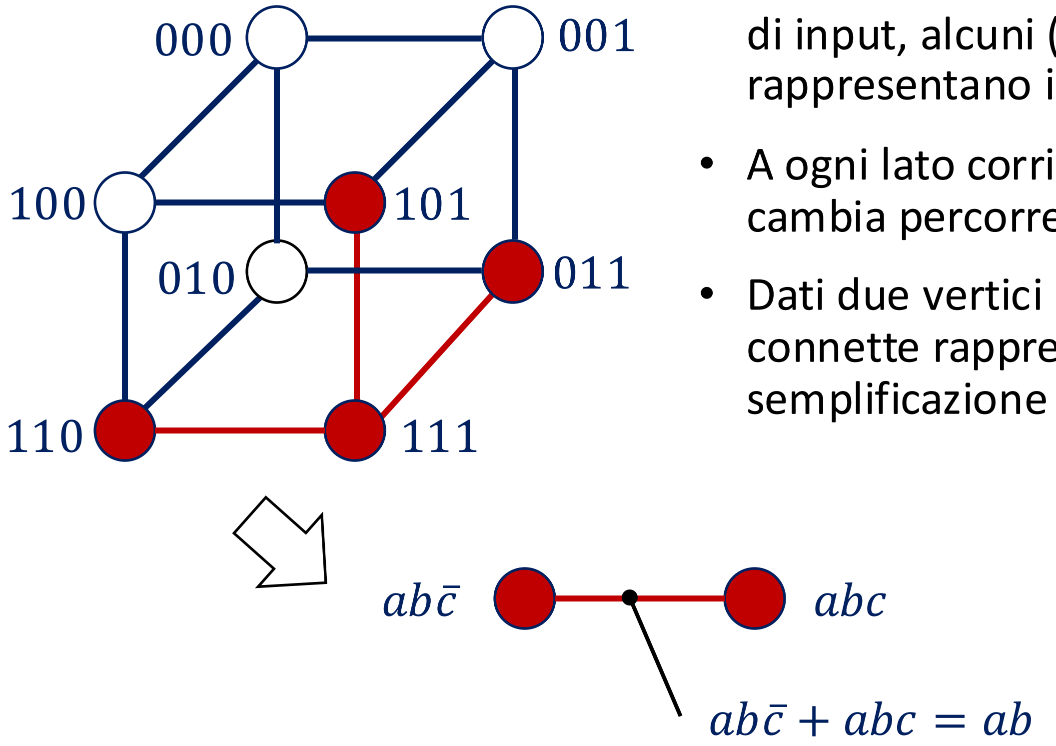
Semplificazione di circuiti

- Per ottenere un circuito più semplice possiamo lavorare sulla sua espressione Booleana
- Applicando le proprietà degli operatori logici possiamo semplificare l’espressione di partenza: a seconda di quali proprietà usiamo otteniamo semplificazioni diverse, di costo diverso e, in generale, non abbiamo garanzie di aver trovato l’espressione «più semplice»
- **Mappe di Karnaugh**: metodo meccanico (e grafico) per la semplificazione delle espressioni Booleane basato sull’identificazione degli implicanti
- Garantisce di trovare l’espressione minima

Mappe di Karnaugh

- In una funzione con n variabili, ogni configurazione di input è rappresentata da una stringa di n bit
- Come abbiamo visto nella codifica dei numeri binari naturali, esiste uno schema grafico per rappresentare stringhe binarie

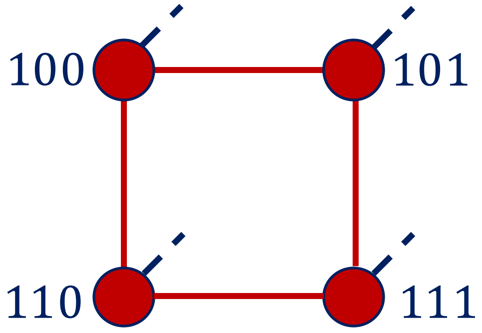
a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



- Ciascun vertice rappresenta una configurazione di input, alcuni (evidenziati in rosso) rappresentano i mintermini della funzione
- A ogni lato corrisponde una variabile, quella che cambia percorrendo quel lato
- Dati due vertici mintermini, il lato che li connette rappresenta un'opportunità di semplificazione da cui calcolare un implicante

Mappe di Karnaugh

- Lo stesso principio si applica anche a «gruppi» di vertici connessi



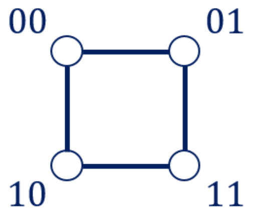
$$a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc$$
$$= a(\bar{b}\bar{c} + \bar{b}c + b\bar{c} + bc)$$
$$= a\left(\bar{b}(\bar{c} + c) + b(\bar{c} + c)\right)$$
$$= a(\bar{b} + b)$$
$$= \mathrm{a}$$

- **Idea**: identificare gruppi connessi di vertici sulla rappresentazione grafica può guidarci nel costruire una forma semplificata
- Limite: per farlo manualmente abbiamo bisogno di una rappresentazione grafica ma …
- Un software non necessita di visualizzare la rappresentazione e può lavorare nello spazio delle adiacenze in n dimensioni senza difficoltà
- Noi ci limitiamo allo svolgimento manuale che si riesce a fare facilmente fino a $n = 4$

Mappe di Karnaugh

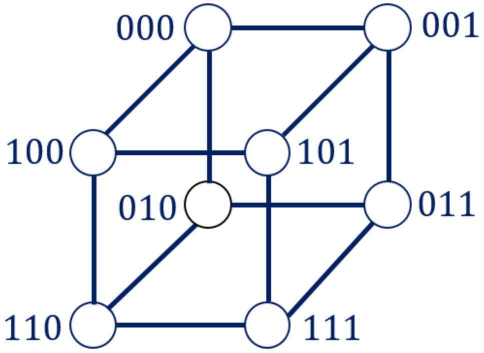
- Passiamo dalla rappresentazione grafica su n dimensioni a una **rappresentazione piana su 2 dimensioni** con cui è più facile lavorare (trovare gruppi di adiacenze)

2 variabili, $f(a,b)$



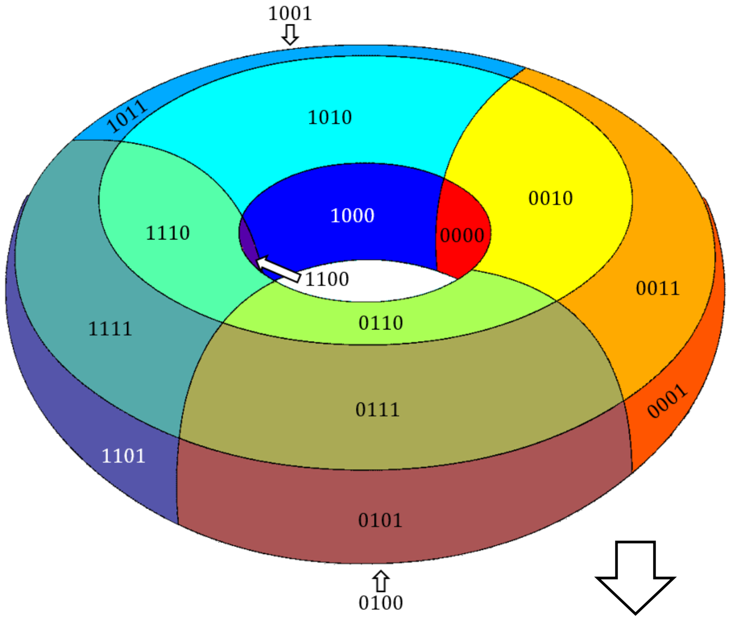
$b \backslash a$	0	1
0		
1		

3 variabili, $f(a,b,c)$



$c \backslash ab$	00	01	11	10
0				
1				

4 variabili, $f(a,b,c,d)$



$cd \backslash ab$	00	01	11	10
00				
01				
11				
10				

Mappe di Karnaugh

- In ogni direzione della tabella seguiamo la **codifica di Grey**
- In ogni casella, indichiamo il valore corrispondente della funzione: 0 oppure 1
- Formare rettangoli, **anche sovrapposti**, che racchiudano tutti gli 1 e la cui area:
  - Sia il più grande possibile
  - Sia una potenza di 2
- Per ogni rettangolo, scriviamo l'implicante associato includendo le variabili (negate o naturali) che non cambiano di valore (restando 0 o 1) quando percorro ogni direzione di lato del rettangolo
- Metto in OR tutti gli implicanti
- **Intuizione**: se una variabile cambia di valore lungo un lato del rettangolo, significa che la funzione continua a valere 1 anche se la variabile cambia, quindi la variabile non determina il valore della funzione e può essere scartata

$b \backslash a$	0	1
0	0	1
1	1	1

$$f(a,b) = b + a$$

$c \backslash ab$	00	01	11	10
0	0	0	1	1
1	0	0	1	1

$$f(a,b,c) = a$$

$cd \backslash ab$	00	01	11	10
00	0	1	0	0
01	0	1	0	1
11	1	1	1	1
10	0	1	0	0

$$f(a,b,c,d) = cd + \bar{a}b + a\bar{b}d$$

Mappe di Karnaugh

- Attenzione: la rappresentazione piana è **ciclica** (*wraparound world!*)

$c \backslash ab$	00	01	11	10
0	1	0	1	1
1	1	0	1	1

$$f(a,b) = \bar{b} + a$$

$cd \backslash ab$	00	01	11	10
00	1	1	0	0
01	0	1	0	0
11	0	1	1	0
10	1	1	0	1

$$f(a,b,c,d) = \bar{a}\bar{d} + \bar{a}b + bcd + \bar{b}c\bar{d}$$

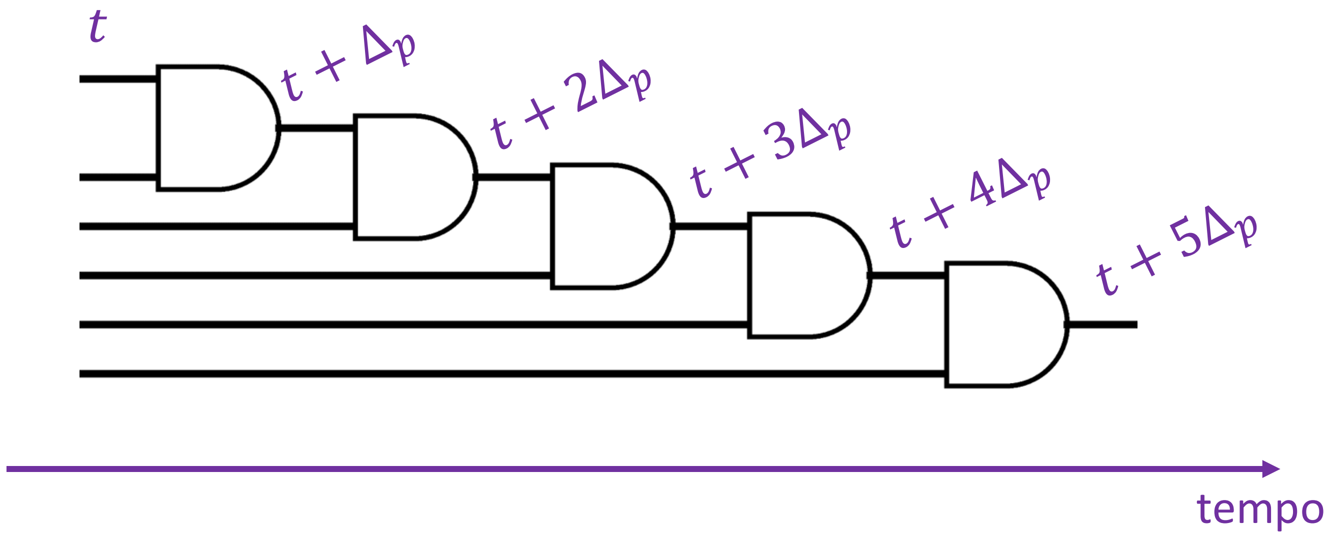
Valutare costi e prestazioni

Limiti dei circuiti logici

- Le due forme canoniche sono il metodo più semplice con cui sintetizzare un circuito combinatorio a partire dalla tabella di verità
- La semplicità ha un costo, finora lo abbiamo valutato considerando aspetti come la lunghezza dell'espressione, la grandezza del circuito e il numero di porte
- Per caratterizzare in modo quantitativo e rigoroso questo costo dobbiamo considerare il fatto che i circuiti sono componenti hardware con annessi limiti fisici:
  - **Propagation delay**: in ogni porta logica se al tempo t gli input cambiano l'uscita non commuta (passa da 0 a 1 o viceversa) in modo istantaneo, l'output sarà stabile dal tempo $t + \Delta_p$
  - **Fan-out limitato**: il numero di ingressi a cui posso collegare una uscita (pilotaggio) è limitato; in generale collegando un'uscita a un numero maggiore di ingressi il tempo di commutazione aumenta

Cammino critico

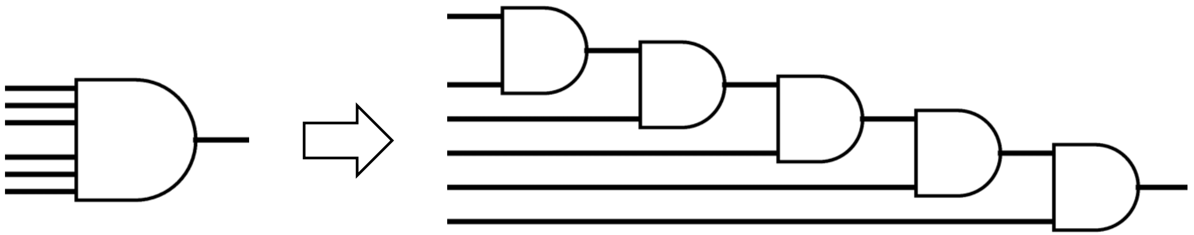
- Ogni porta logica che un segnale deve attraversare introduce un ritardo additivo sul tempo di commutazione dell'uscita



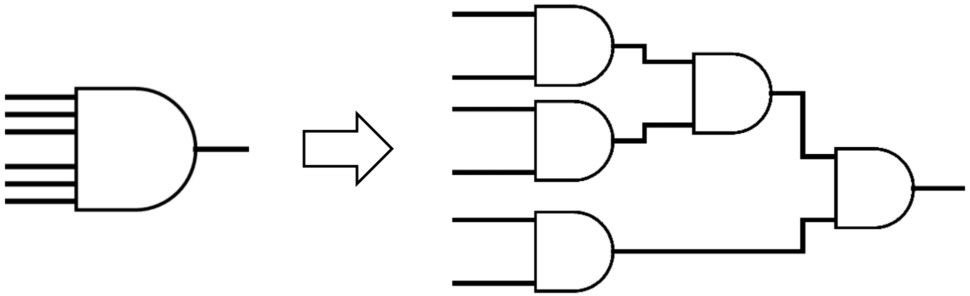
- Nel suo percorso dall'ingresso all'uscita il segnale paga un ritardo Δ_p ogni volta che viene attraversata una porta logica
- Dato il percorso da un ingresso ad un'uscita, il numero porte logiche attraversate si chiama **lunghezza del cammino**
- La lunghezza massima di tutti i percorsi presenti in un circuito si chiama **cammino critico**
- È una metrica con cui valutare le performance di un circuito, più grande è il cammino critico più lento sarà il circuito

Cammino critico

- Se consideriamo il cammino critico come metrica di performance possiamo proporre un'implementazione migliore delle porte logiche ad n ingressi



n ingressi cammino critico pari a $n-1$ (nell'esempio: 5)



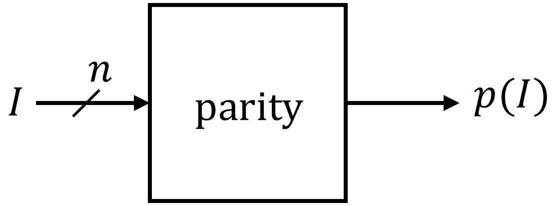
n ingressi cammino critico pari a $\lceil \log_2 n \rceil$ (nell'esempio: 3)

- Altri criteri che possiamo usare: area occupata, numero totale di porte, energia dissipata, facilità di interpretazione
- In generale avere una forma semplificata migliora queste metriche, semplificare è più difficile ma porta a dei vantaggi

Parità e maggioranza

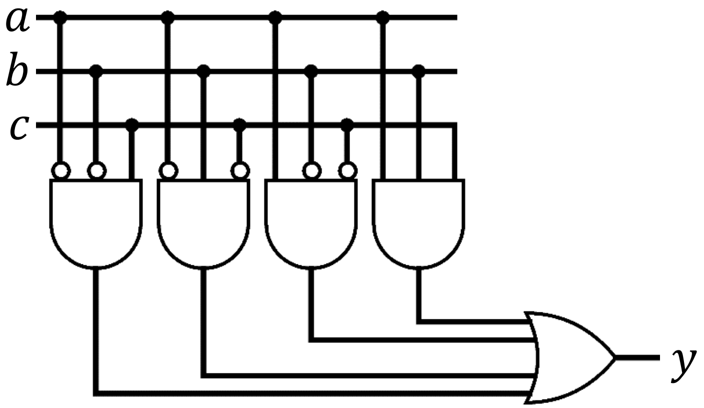
Funzione di parità

- **Esercizio**: sintetizzare il circuito della funzione di parità su 3 bit
- La funzione di parità vale 1 se il numero di input a 1 è **dispari,** può essere pensato come un modulo che riceve un input I su n bit e che pone in uscita il bit di parità $p(I)$



a	b	c	y	
0	0	0	0	
0	0	1	1	$\bar{a}\bar{b}c$
0	1	0	1	$\bar{a}b\bar{c}$
0	1	1	0	
1	0	0	1	$a\bar{b}\bar{c}$
1	0	1	0	
1	1	0	0	
1	1	1	1	abc

$$y = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$



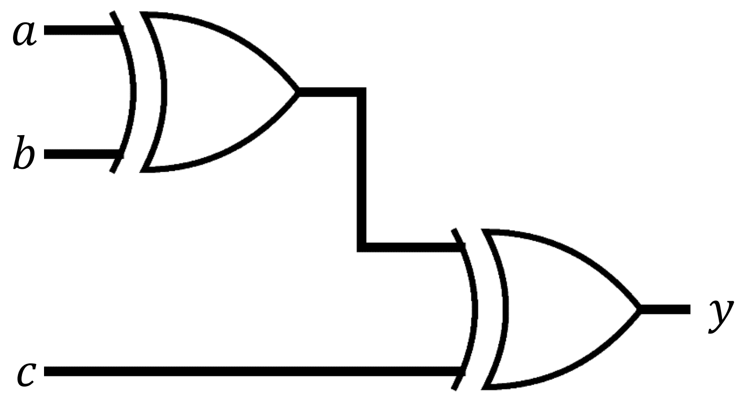
Funzione di parità

- Supponiamo di «spezzare» I in due parti, un suo prefisso I_1 e un postfisso I_2, se $I = 00110101$, possiamo avere ad esempio $I_1 = 001$ e $I_2 = 10101$
- Quando $p(I)$ è pari a 1? Possiamo scrivere la risposta in funzione dei bit di parità di I_1 e I_2

$p(I_1)$	$p(I_2)$	$p(I)$	
0	0	0	*pari+pari=pari*
0	1	1	*pari+dispari=dispari*
1	0	1	*dispari+pari=dispari*
1	1	0	*dispari+dispari=pari*

$\boldsymbol{p(I) = p(I_1) \oplus p(I_2)}$ Definizione ricorsiva basata su XOR

Su 3 bit:

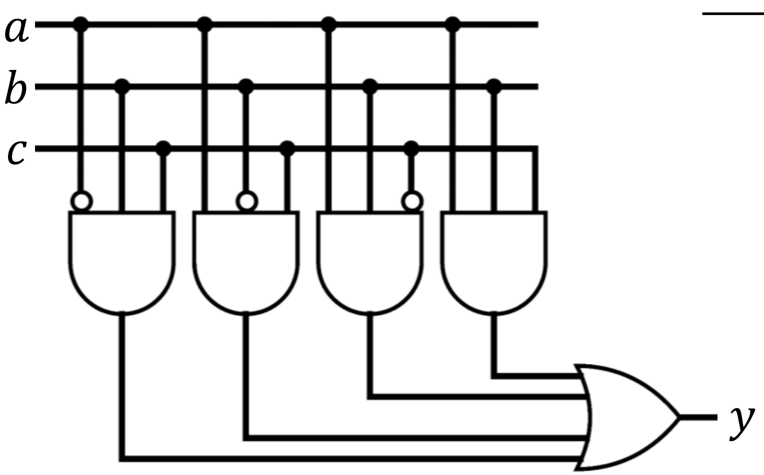


Funzione di maggioranza

- **Esercizio**: sintetizzare il circuito della funzione di maggioranza su 3 bit
- La funzione di maggioranza vale 1 se il numero di input a 1 è **maggiore** del numero di input pari a 0

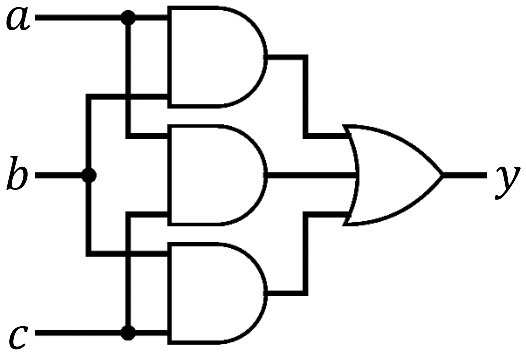
a	b	c	y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{a}bc$
1	0	0	0	
1	0	1	1	$a\bar{b}c$
1	1	0	1	$ab\bar{c}$
1	1	1	1	abc

$$y = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$



Semplificazione

Passaggio	Ottenuto grazie a…
$\bar{a}bc + a\bar{b}c + ab\bar{c} + abc$	
$= \bar{a}bc + a\bar{b}c + ab\bar{c} + abc + abc + abc$	Idempotenza
$= ab(c + \bar{c}) + ac(b + \bar{b}) + bc(a + \bar{a})$	Distributiva
$= ab + ac + bc$	Inverso e el. nullo



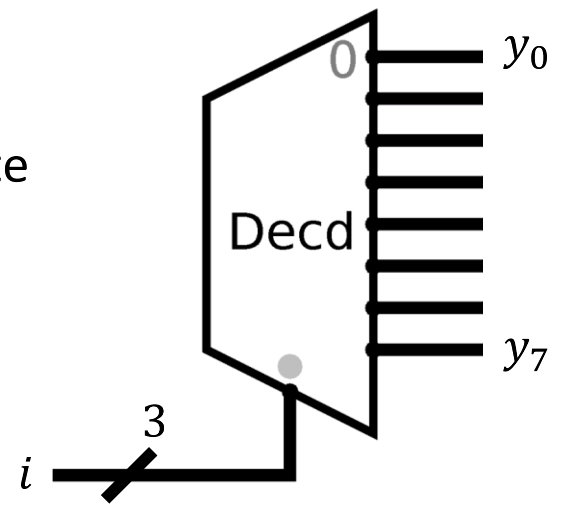
Blocchi funzionali

- Nella sintesi di circuiti digitali, esistono dei sotto-circuiti notevoli che ritornano spesso perché svolgono elaborazioni molto comuni
- **Idea**: **modularizzare** questi circuiti in blocchi funzionali che possiamo utilizzare come veri e propri elementi di una libreria
- In questo modo li possiamo usare «dimenticandoci» della loro implementazione interna e possiamo facilitare la sintesi di circuiti molto complessi
- Quali sono gli elementi di questa «libreria»?

Decoder

- **Decoder**: circuito che ha n ingressi e 2^n uscite
- Gli n bit in ingresso possono essere pensati come un valore binario naturale nell'intervallo $[0,2^n - 1]$
- Il numero di valori possibili è pari al numero di uscite: ogni valore identifica una linea di uscita
- Se in ingresso ho il valore i, allora l'i-esima uscita è «asserita» (vale 1), tutte le altre valgono 0 (le linee si contano a partire da 0)
- **Esempio**:
  - se $i = 101$ (5 in base 10) allora $y_5 = 1$ e tutte le altre 0
  - se $i = 111$ (7 in base 10) allora $y_7 = 1$ e tutte le altre 0
- **Interpretazioni**
  - Il numero i è il codice che identifica l'i-esima linea, Il circuito riceve un codice e «accende» la linea corrispondente
  - Converte un valore nel suo **codice one-hot** (un gruppo di bit con un singolo 1)

esempio di decoder a 3 bit

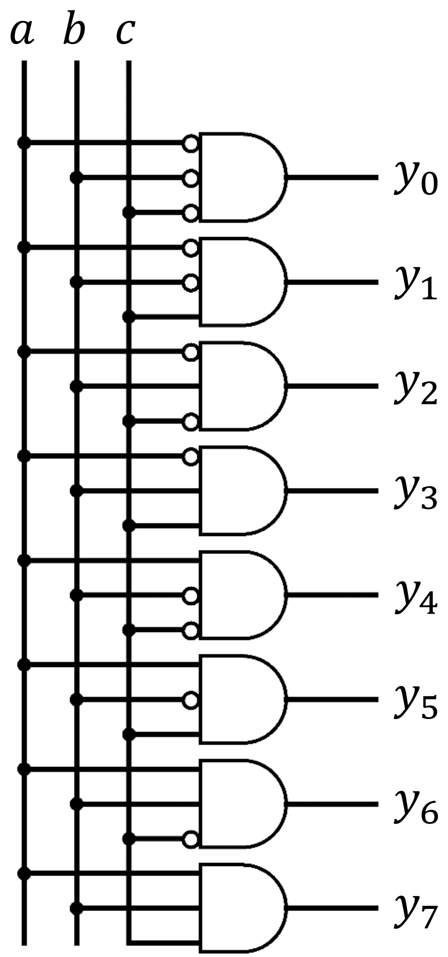


Decoder

- Come è fatto al suo interno?

Tabella di verità

a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



- **Ogni uscita** è una funzione logica con **un solo mintermine**
- La stessa struttura scala con l'aumentare dei bit in ingresso

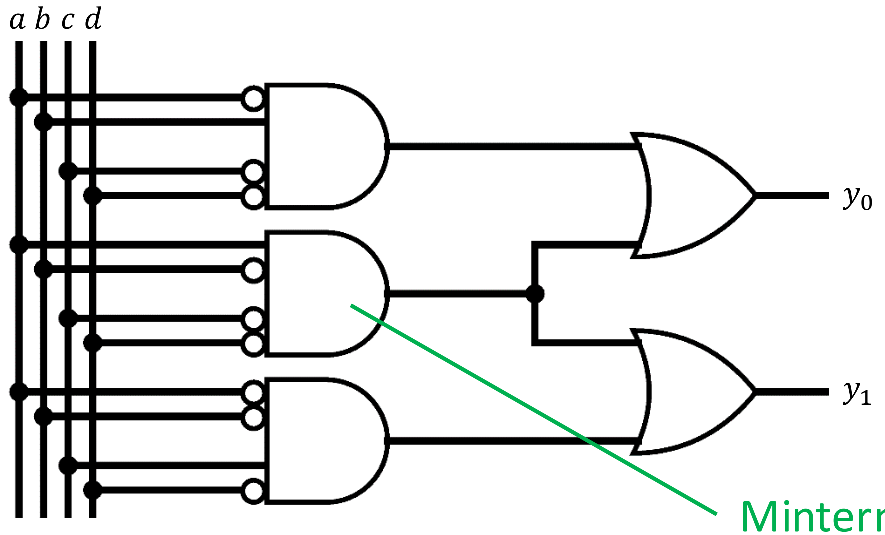
Encoder

a	b	c	d	y_0	y_1
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	x	x
0	1	0	0	1	0
0	1	0	1	x	x
0	1	1	0	x	x
0	1	1	1	x	x
1	0	0	0	1	1
1	0	0	1	x	x
1	0	1	0	x	x
1	0	1	1	x	x
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x

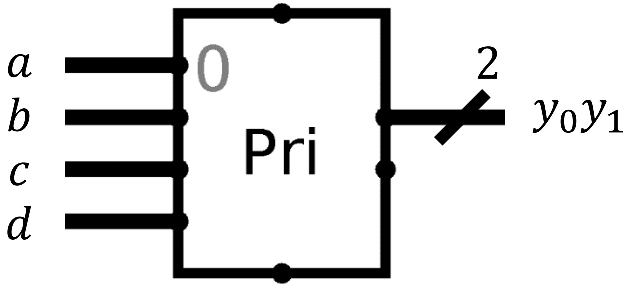
- **Encoder**: è l'opposto del decoder, ha 2^n ingressi e n uscite
- Tra i 2^n input uno solo deve essere 1, in uscita leggo un codice binario corrispondente
- Nel caso di $n = 2$ si hanno questi codici

Input	Codice binario
0001	00
0010	01
0100	10
1000	11

- Le x nella tabella di verità rappresentano dei «**don't care**» cioè valori rispetto a cui sono indifferente: sono configurazioni di input possibili, ma che non sono soggette alla specifica
- Si scelgono in modo da semplificare il più possibile l'implementazione (conviene scegliere $x = 0$!)



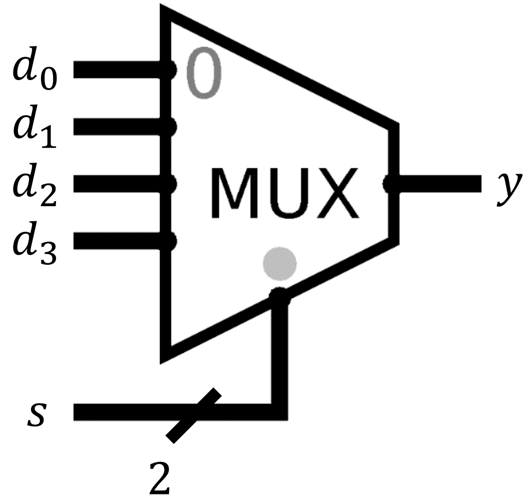
Mintermine condiviso tra le due uscite!



Multiplexer

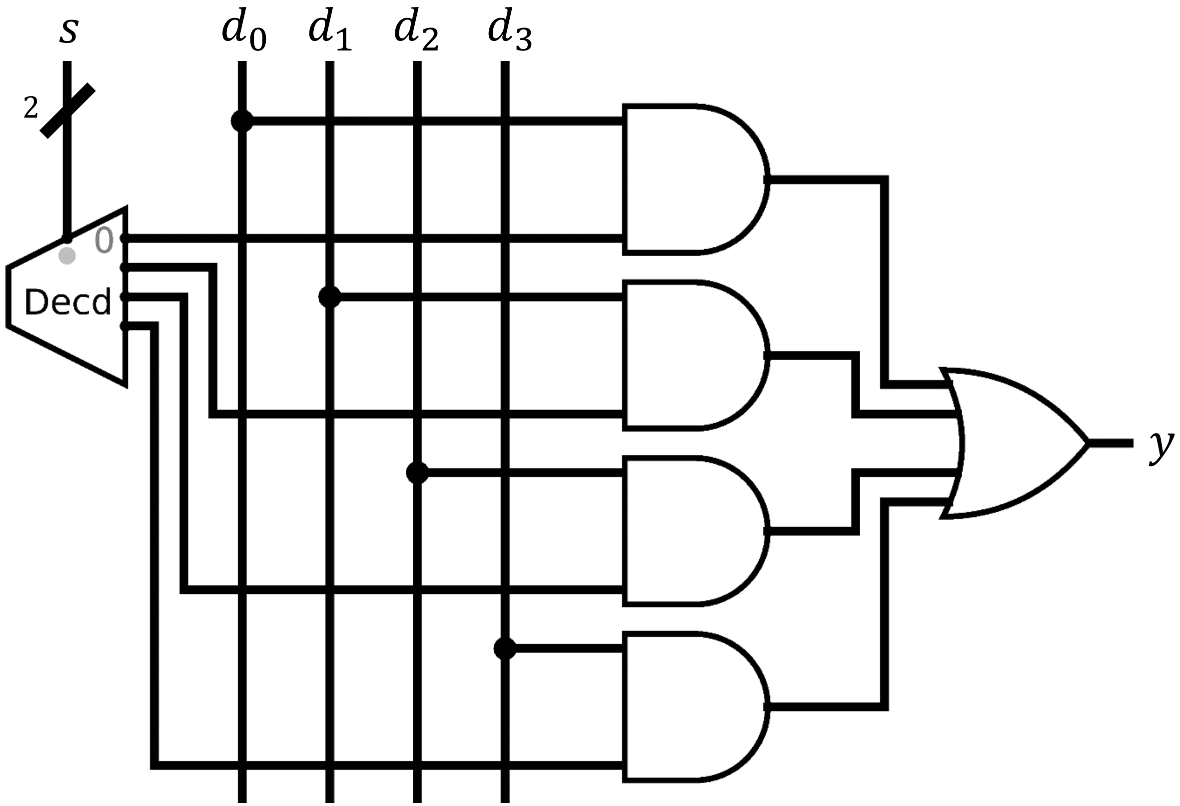
- **Multiplexer** (MUX): un circuito che $2^n + n$ input e 1 bit in uscita
- I 2^n ingressi rappresentano le linee dati: ciascuna linea presenta al MUX un singolo bit
- I restanti n input sono un segnale di controllo (selezione): un valore binario naturale che identifica quale delle 2^n linee dati passa verso l'uscita
- **Esempi**: se $s = 01$ (1 in base 10) allora $y = d_1$, se $s = 11$ (3 in base 10) allora $y = d_3$

esempio di mux con selezione a 2 bit



Multiplexer

- Come è fatto al suo interno?
- Possiamo sfruttare il decoder



- Attraverso il segnale di controllo s poniamo ad 1 la linea in ingresso ad uno degli AND
- Quando un input di AND viene posto a 1, l'uscita è uguale all'altro input (come se questo passasse verso l'uscita)
- Poiché il decoder asserisce una sola linea, un solo AND avrà in uscita il dato, gli altri avranno in uscita 0

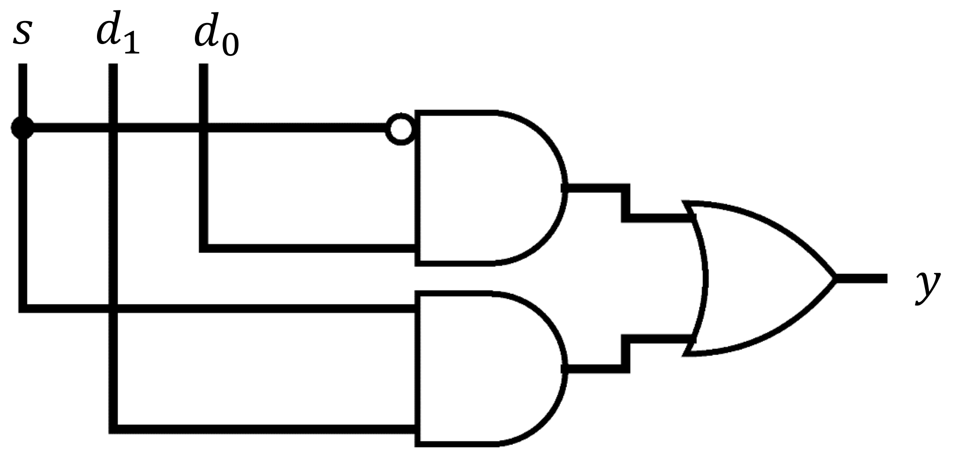
Multiplexer

- **Esercizio**: sintesi e semplificazione della prima forma canonica di un MUX con selezione a 1 bit

s	d_0	d_1	y	
0	0	0	0	
0	0	1	0	
0	1	0	1	$\bar{s}d_0\overline{d_1}$
0	1	1	1	$\bar{s}d_0d_1$
1	0	0	0	
1	0	1	1	$s\overline{d_0}d_1$
1	1	0	0	
1	1	1	1	sd_0d_1

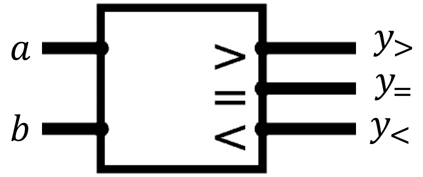
$$y = \bar{s}d_0\overline{d_1} + \bar{s}d_0d_1 + s\overline{d_0}d_1 + sd_0d_1 = \bar{s}d_0\left(\overline{d_1} + d_1\right) + sd_1\left(\overline{d_0} + d_0\right)$$

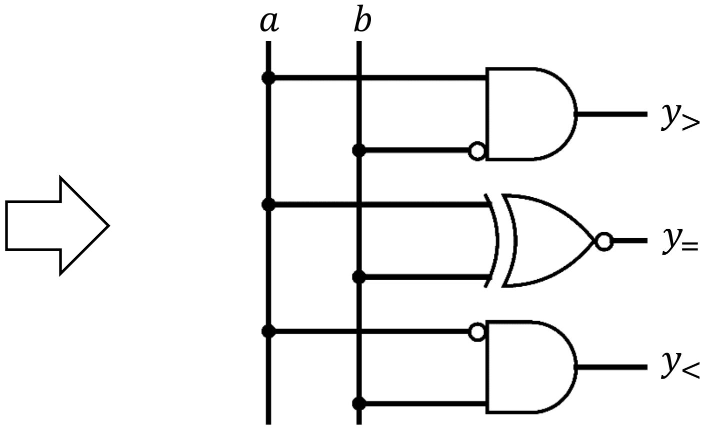
$$= \bar{s}d_0 + sd_1$$



Comparatore

- **Comparatore**: riceve in ingresso due numeri binari su n bit a e b e calcola tre uscite corrispondenti a tre test di confronto sul valore rappresentato:
  - Test di maggioranza stretta $a > b$, uscita $y_>$
  - Test di uguaglianza $a == b$, , uscita $y_=$
  - Test di minoranza stretta $a < b$, uscita $y_<$
- Dati i due valori in input, una sola delle tre uscite sarà pari a 1, le altre due saranno pari a 0
- Comparatore a $n = 1$ bit:

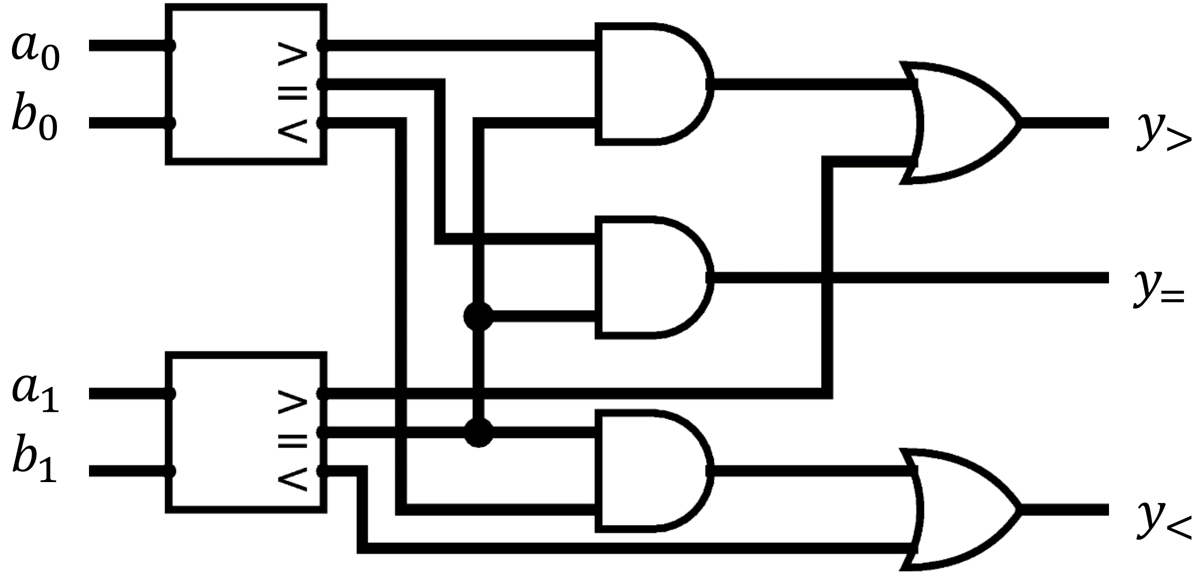




Per il test di uguaglianza usiamo lo XNOR

Comparatore

- Comparatore a $n = 2$ bit: si ottiene combinando le uscite di due comparatori a 1 bit



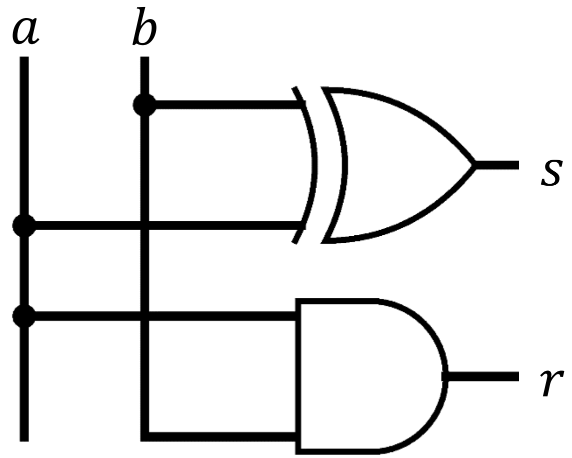
- Questo schema può essere replicato per ottenere un comparatore da n bit, usando due comparatori da $\frac{n}{2}$ bit: un comparatore confronta le metà meno significative dei due valori mentre l'altro confronta le metà più significative

Circuiti aritmetici

Half adder

- Consideriamo il caso più semplice della somma binaria, quello di due numeri su 1 bit (ricordiamo che le regole della somma tra naturali e interi in complemento a 2 sono le stesse)
- Scriviamo la regola della somma binaria come se fosse una tabella di verità di una funzione che ha due uscite: il risultato della somma s e il riporto r
- La somma è uno XOR, il riporto è un AND

a	b	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- Cammino critico: 1

Full Adder

- L'half adder non basta per descrivere tutti i casi della somma su un singolo bit: ci potrebbe essere un riporto da considerare in aggiunta ai due bit da sommare
- **Full adder**: oltre ai due bit a e b prevede un input r_{in} per un eventuale riporto in ingresso

a	b	r_{in}	s	r_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$s = \bar{a}b\overline{r_{in}} + a\bar{b}\overline{r_{in}} + \bar{a}\bar{b}r_{in} + abr_{in}$$

$$= \overline{r_{in}}\left(\bar{a}b + a\bar{b}\right) + r_{in}\left(\bar{a}\bar{b} + ab\right)$$

$$= \overline{r_{in}}(a \oplus b) + r_{in}\left(\overline{a \oplus b}\right)$$

$$= r_{in} \oplus (a \oplus b) = r_{in} \oplus a \oplus b$$

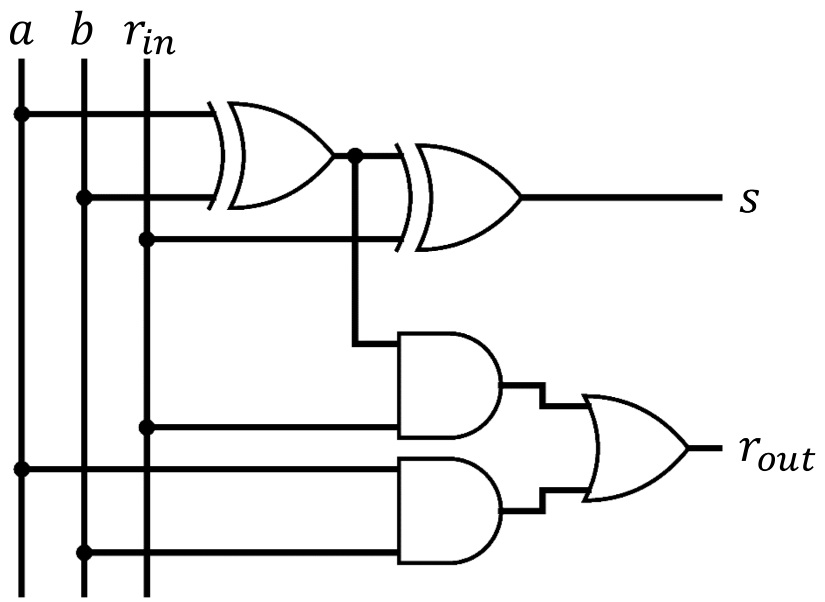
$$r_{out} = ab\overline{r_{in}} + \bar{a}br_{in} + a\bar{b}r_{in} + abr_{in}$$

$$= ab(\overline{r_{in}} + r_{in}) + r_{in}\left(\bar{a}b + a\bar{b}\right)$$

$$= ab + r_{in}(a \oplus b) = \boldsymbol{ab + r_{in}(a + b)}$$

Implementiamo questa

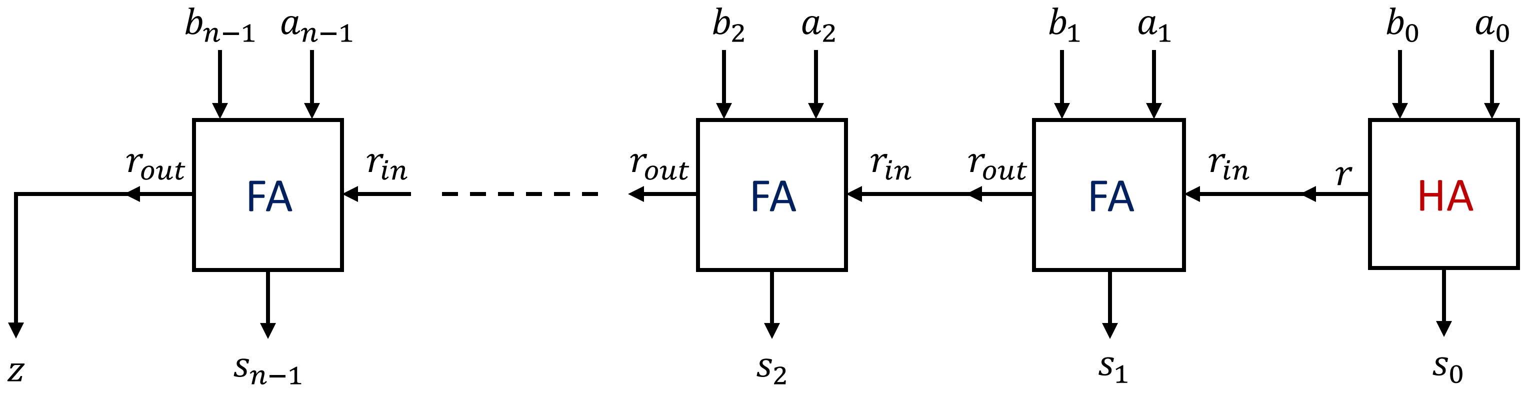
Ma questa si dimostra che è equivalente! (grazie al termine ab)



- Cammino critico: 3

Sommatore a propagazione di riporto su n bit

- Per poter sommare numeri $a = a_{n-1}a_{n-2} \dots a_1a_0$ e $b = b_{n-1}b_{n-2} \dots b_1b_0$ su n bit posso collegare in serie i sommatori su 1 bit
- L'Half Adder somma i due bit meno significativi: calcola il bit meno significativo della somma e «propaga» il riporto verso il Full Adder adiacente che somma i due bit successivi

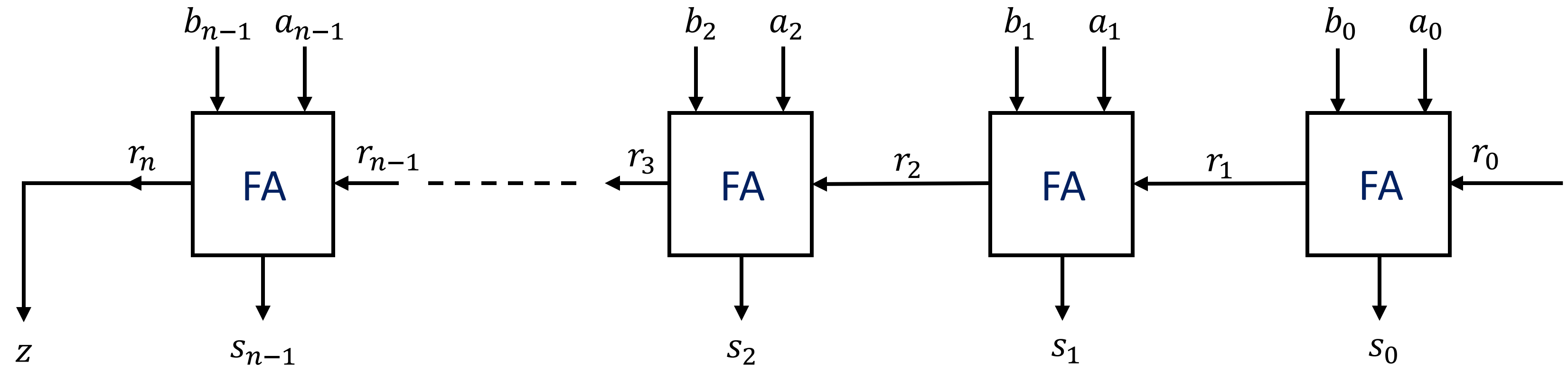


$$\begin{array}{r} a_{n-1}a_{n-2} \dots a_1a_0 + \\ b_{n-1}b_{n-2} \dots b_1b_0 = \\ \hline s_{n-1}s_{n-2} \dots s_1s_0 \end{array}$$

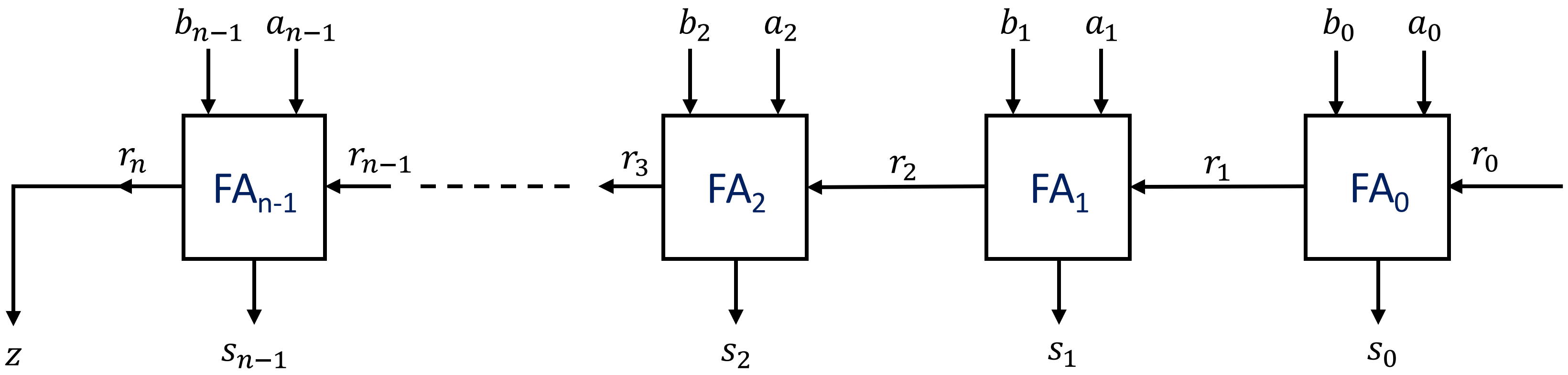
- Ultimo riporto z: se sommiamo due numeri naturali indica l'overflow, se sommiamo due numeri in C2 va ignorato
- Calcolo del cammino critico: la propagazione del riporto implica che l'$(i+1)$-esimo blocco debba aspettare che l' i-esimo blocco calcoli r_{out} (che diventa r_{in}), gli adder non lavorano in parallelo
- Cammino critico (da a_0 a s_{n-1}, attraversa tutto il circuito!): $1 + 3 + 3 + \cdots + 3 = 1 + 3(n-1) = 3n - 2 \approx 3n$

Anticipazione di riporto (carry lookahead)

- Il sommatore a propagazione di riporto ha un cammino critico elevato, dover attraversare tutto il circuito rallenta l'elaborazione
- La causa principale è proprio la propagazione dei riporti
- **Idea**: anziché calcolare i riporti percorrendo la catena dei Full Adder provo a scrivere un'espressione diretta per ogni riporto in modo da anticipare il suo calcolo con un sotto-circuito *ad hoc*
- Considero questa implementazione equivalente (solo Full Adder, notazione compatta) che mi consente di semplificare la derivazione



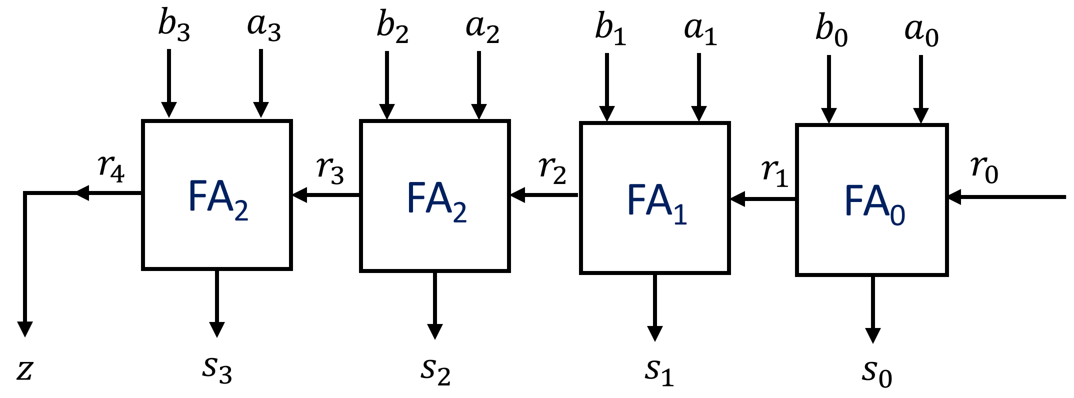
Anticipazione di riporto

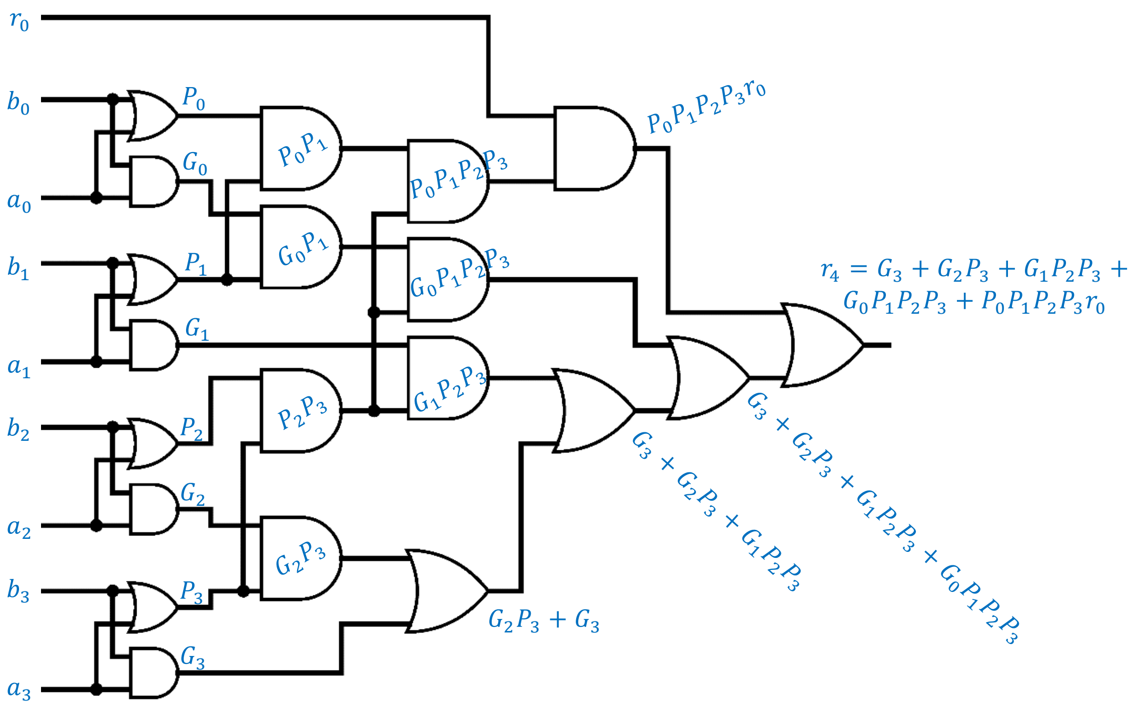


- Ricordiamo che nell'i-esimo Full Adder (FA$_i$) $r_{i+1} = a_i b_i + r_i(a_i + b_i)$, usiamo l'espressione semplificata (senza lo XOR)
- Rinomino i termini dentro l'espressione:
  - $a_i b_i$ è il termine di generazione: G_i; in ogni FA è «subito pronto», non deve aspettare!
  - $(a_i + b_i)$ è il termine di propagazione: P_i; deve aspettare la propagazione di r_i
- Per ogni riporto derivo la sua espressione, inizio con r_1 e vado avanti fino a r_n
- $r_1 = a_0 b_0 + r_0(a_0 + b_0) = G_0 + r_0 P_0$
- $r_2 = a_1 b_1 + r_1(a_1 + b_1) = G_1 + r_1 P_1 = G_1 + (G_0 + r_0 P_0)P_1 = G_1 + P_1 G_0 + P_1 P_0 r_0$
- $r_3 = a_2 b_2 + r_2(a_2 + b_2) = G_2 + r_2 P_2 = G_2 + (G_1 + r_1 P_1)P_2 = G_2 + (G_1 + (G_0 + r_0 P_0)P_1)P_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 r_0$
- …
- In generale : $r_n = G_{n-1} + P_{n-1} G_{n-2} + P_{n-1} P_{n-2} G_{n-3} + \cdots + P_{n-1} P_{n-2} \ldots P_0 r_0$

Anticipazione di riporto

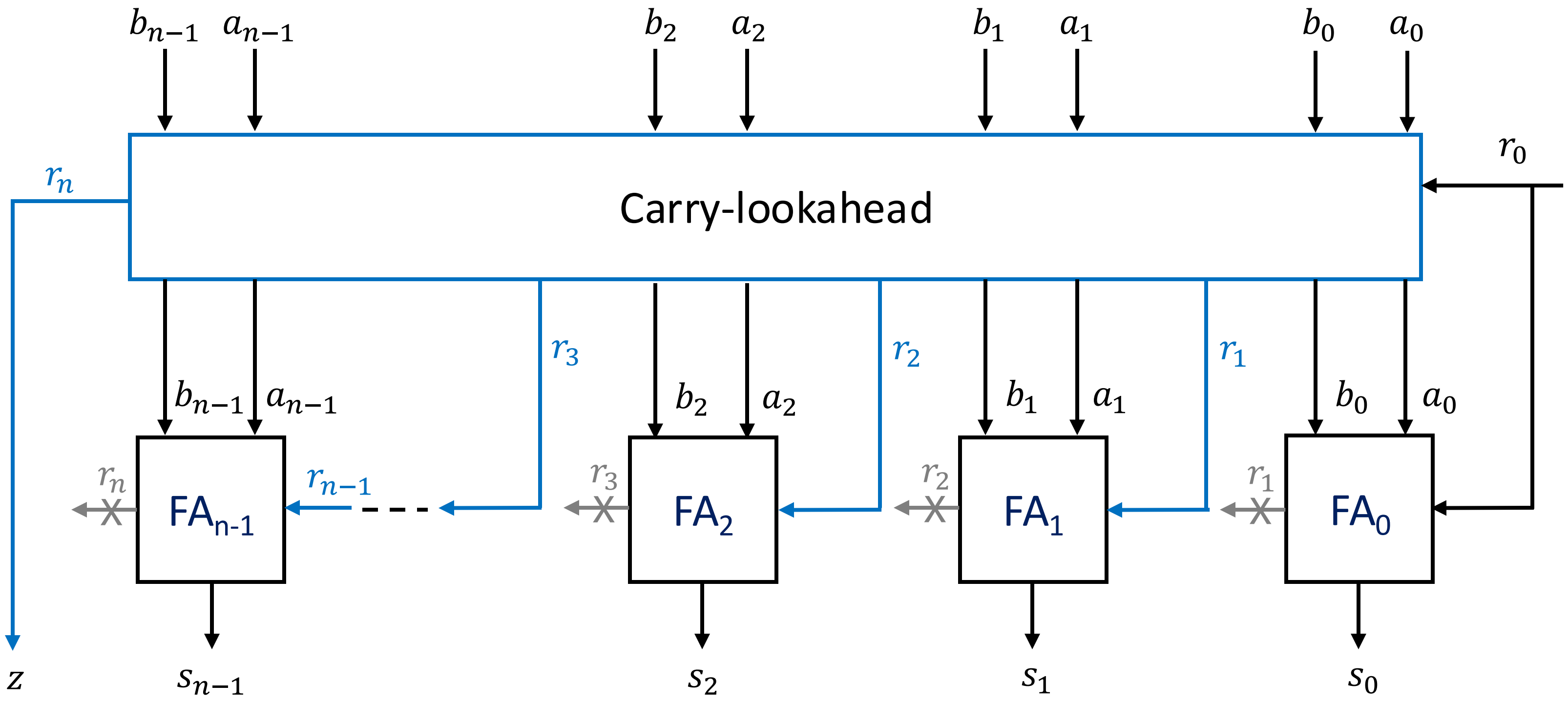
- Consideriamo un sommatore a propagazione di riporto con $n = 4$, se lo implementiamo con lo schema precedente (solo FA) otteniamo un cammino critico pari a $3n = 12$, che corrisponde al numero di porte da attraversare per propagare i riporti fino a r_4
- Se calcoliamo ogni riporto con un circuito dedicato che implementa la sua espressione diretta otteniamo che :
  $r_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0r_0$
- Il cammino critico di questo circuito è 6: se calcoliamo il riporto con questa rete anticipiamo il suo arrivo all'uscita (dimezziamo il numero di porte attraversate) e abbassiamo il cammino critico di tutto il circuito!



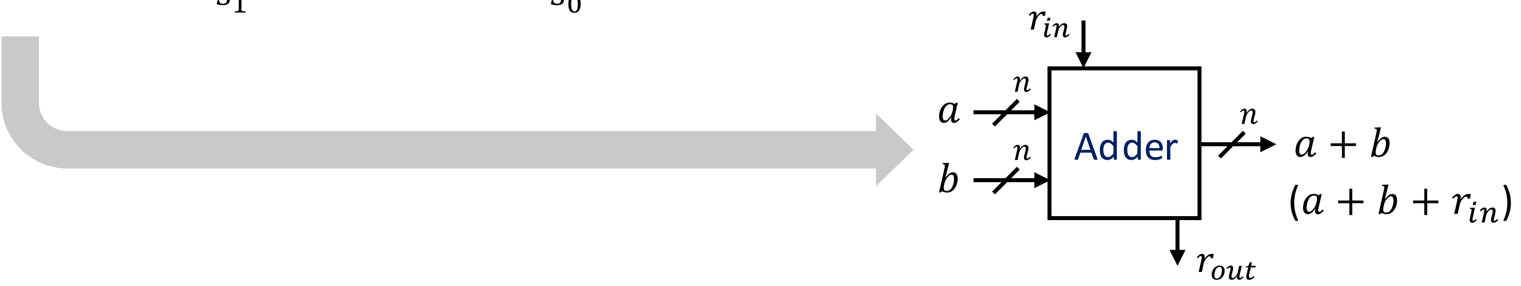


Anticipazione di riporto

- Come cambia la struttura del sommatore?

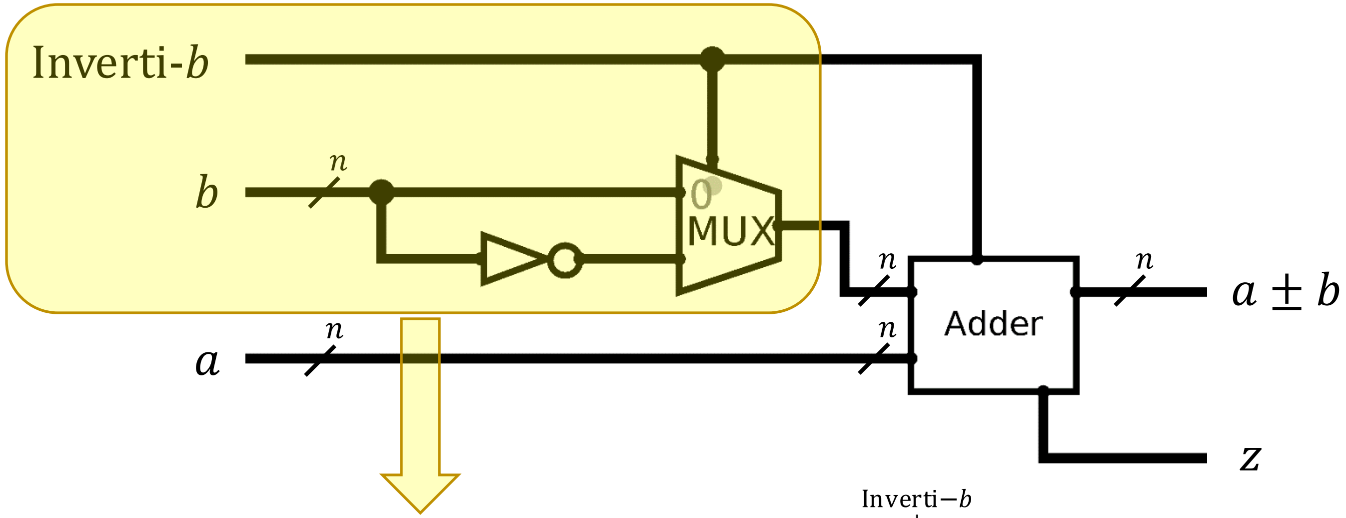


- Spezziamo la catena della propagazione dei riporti!
- Ora ciascun FA può lavorare in parallelo senza dover aspettare il riporto del FA precedente, il riporto è calcolato «in anticipo» dall’**unità di look-ahead**
- Il cammino critico è sempre dato dall’ultimo riporto, ma è più corto rispetto all’implementazione con propagazione dei riporti



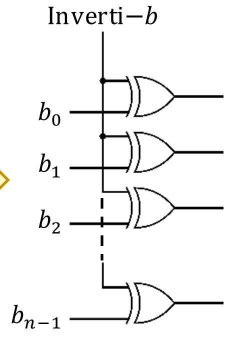
Sottrazione

- Il modulo per l'addizione che abbiamo costruito può essere esteso per gestire anche le sottrazioni
- **Metodo**: la sottrazione $a - b$ si interpreta, e quindi si esegue, come una somma binaria tra a e il complemento a 2 di b (la somma si esegue sempre con le stesse regole dei naturali, eccezion fatta per l'uso dell'ultimo riporto)
- Fare il complemento a 2 di b significa complementare a 1 e sommare 1



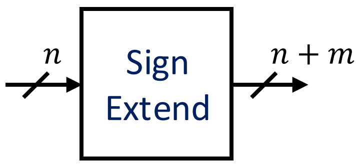
- Inverti-b è un bit di controllo che se viene posto a 1 produce due effetti:
  1. L'Adder riceve il complemento a 1 di b (NOT bit a bit) anziché b
  2. L'Adder riceve un riporto in ingresso pari a 1
- In questo caso quindi l'Adder esegue:
$$a + \bar{b} + 1 = a + (\bar{b} + 1) = a - b$$
- Se Inverti$-b$ è posto a 1, z va scartato, altrimenti indica un overflow

- Ci ricorda qualcosa?
- La seconda interpretazione dello XOR!

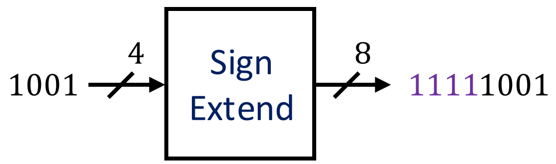


Estensione di segno e shift

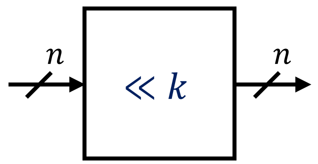
- Due operazioni comuni che ci possono tornare utili
- **Estensione di segno**: ho un segnale su n bit che voglio dare in input ad un circuito che riceve segnali su $n + m$ bit
- Estendere il segno vuol dire replicare a sinistra l'MSD fino a raggiungere il numero totale di bit desiderati; nel caso in cui gli n bit iniziali rappresentino un naturale o un intero in C2, questa operazione non altera il valore rappresentato



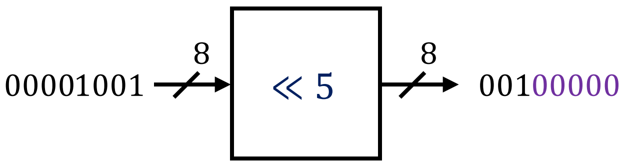
Esempio:



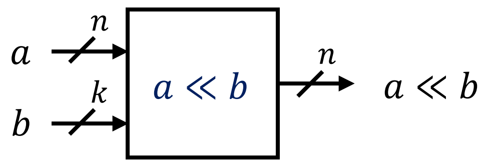
- **Shift**: trascinare gli n bit verso sinistra o destra di k posizioni: sinistra $\ll k$, destra $\gg k$;
- Facendo lo shift, k cifre scompaiono e compaiono k nuovi 0 a destra ($\ll k$) o a sinistra ($\gg k$); se i bit rappresentano un numero naturale e non vengono cancellati 1, $\ll k$ equivale ad una moltiplicazione per 2^k



Esempio:



Esercizio: usando lo shifter a singolo input si realizzi uno shifter a 2 input (si consideri $k = 2$)



Moltiplicatore

- La moltiplicazione binaria si organizza in due fasi
  1. Calcolo dei prodotti parziali: AND a coppie di bit in posizione shiftata
  2. Somma bit a bit (considerando anche i riporti) dei prodotti parziali, useremo dei Full Adder

$$
\begin{array}{lr l}
 & 10111 \times & a \\
 & 101 = & b \\
\hline
 & 111 & \\
b_0 \times a \rightarrow & 10111 & \\
b_1 \times a \rightarrow & 00000 & \\
b_2 \times a \rightarrow & 10111 & \\
\hline
 & 1110011 & a \times b
\end{array}
$$

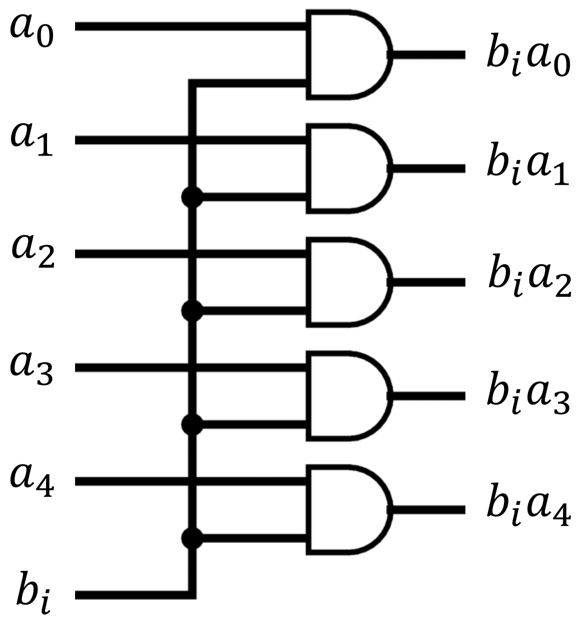
Prodotti parziali

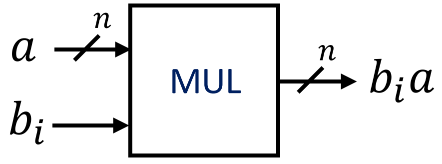
$$
\begin{array}{ccccccc}
 & & b_0a_4 & b_0a_3 & b_0a_2 & b_0a_1 & b_0a_0 \\
 & b_1a_4 & b_1a_3 & b_1a_2 & b_1a_1 & b_1a_0 & \\
b_2a_4 & b_2a_3 & b_2a_2 & b_2a_1 & b_2a_0 & &
\end{array}
\Bigg\downarrow \text{Somma}
$$

- In generale il prodotto di 2 numeri su n bit, può dare un risultato su $2n$ bit, di norma si suddivide il risultato in due numeri separati: con HI si indicano gli n bit della parte alta (indicano anche se c'è un overflow), con LO gli n della parte bassa

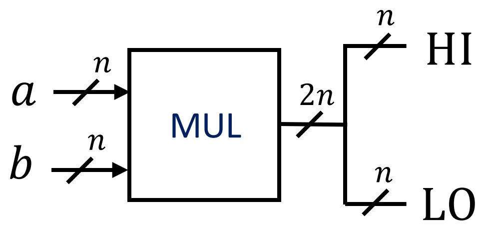
Moltiplicatore $1 \times n$

- Consideriamo un moltiplicatore che prende in input un singolo bit e un operando su n bit, ci serve per calcolare ciascuna riga di prodotti parziali (consideriamo sempre l'esempio con $n = 5$)





Usando questo componente posiamo costruire un moltiplicatore $n \times n$



Moltiplicatore $n \times n$