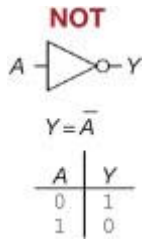
2. ALGEBRA DI BOOLE E RETI COMBINATORIE

un calcolatore può essere visto come un complesso sistema digitale che manipola e memorizza informazioni rappresentate in codice binario.
I componenti digitali che costituiscono i mattoni fondamentali di un calcolatore sono le porte logiche
Le **porte logiche** sono semplici circuiti digitali che prendono uno o più input e producono un output, usate per realizzare delle semplici operazioni
La relazione tra ingressi e uscita può essere descritta con una tabella di verità oppure con un espressione booleana.
Una **tabella di verità** ha come campi gli ingressi in input e l'uscita e ha una riga per ogni possibile combinazione dei valori di ingresso
Un **espressione booleana** è un espressione matematica che utilizza variabili binarie e operatori dell'algebra di boole

Porta NOT



La porta NOT restituisce in output il complemento dell'input

Una porta NOT ha un ingresso, A, e un'uscita, Y
L'uscita della porta NOT è l'esatto contrario del suo ingresso:
se A è FALSO allora Y è VERO
se A è VERO allora Y è FALSO
La porta NOT (negatore) viene anche chiamata porta invertente (**inverter**).

Buffer

L'altra porta logica a un solo ingresso viene chiamata buffer.
Questa porta logica riproduce semplicemente il valore di ingresso in uscita

Porta AND

$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

La porta AND restituisce in output la **congiunzione** degli input

genera all'uscita Y il valore VERO se e soltanto se sia A sia B hanno valore VERO, altrimenti l'uscita vale FALSO.
Y è uguale a 1 se e solo se A e B sono entrambi uguali a 1

Porta OR

$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

La porta OR restituisce in output la **disgiunzione** degli input

produce all'uscita Y il valore VERO se A, oppure B, oppure entrambi gli ingressi hanno valore VERO.
Y è uguale a 1 se e solo se A o B è uguale a 1 (oppure A e B sono uguali a 1)

Altre porte logiche a due ingressi

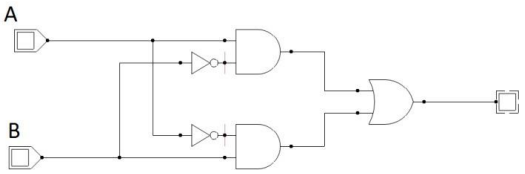
XOR

$Y = A \oplus B$

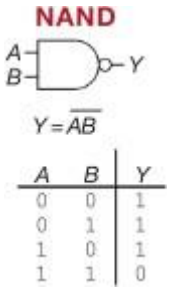
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

La porta XOR (OR esclusivo, detto "ex–OR") dà uscita VERO se A oppure B, ma non entrambi, hanno valore VERO.
Y = 1 se e solo se A oppure B è uguale a 1
Una porta XOR a N ingressi viene anche chiamata **"porta di parità"** perché produce in uscita VERO se un numero dispari di ingressi è VERO.
la porta XOR può essere ottenuta mediante porte AND, OR e NOT

XOR: Y = 1 sse A≠B
A≠B sse (A=1 e B=0) o (A=0 e B=1)
$Y=(A\cdot\overline{B})+(\overline{A}\cdot B)$

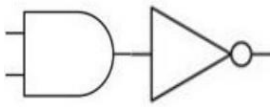


NAND

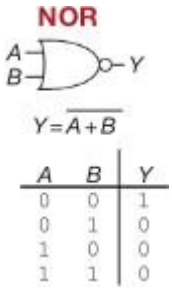


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

La porta NAND esegue le operazioni NOT e AND.
La sua uscita ha sempre valore VERO tranne quando entrambi gli ingressi sono VERO
Y è uguale a 1 se e solo se A o B non sono uguali a 1
La porta NAND può essere ottenuta complementando una porta AND, ovvero mettendo in serie una porta AND e una NOT

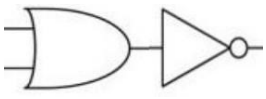


NOR

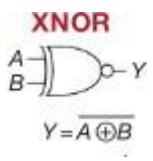


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

La porta NOR esegue le operazioni NOT e OR.
La sua uscita è VERO se né A né B sono VERO
Y è uguale a 1 se e solo se A e B non sono uguali a 1
la porta NOR si può ottenere mettendo in serie una porta OR e una NOT



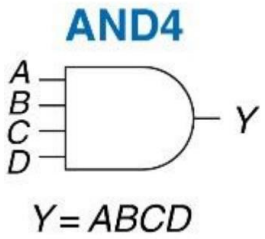
XNOR
porta XNOR esegue l'operazione negata rispetto a una porta XOR.



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

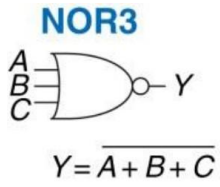
Porte logiche con più linee di input
Le porte logiche AND, OR, NAND,… possono avere anche più di 2 linee di ingresso in input
Una porta AND con un numero N di ingressi produce un valore di uscita VERO quando tutti i valori di ingresso sono VERO.
Invece una porta OR con un numero N di ingressi produce un valore di uscita VERO quando almeno uno dei suoi ingressi è VERO.

AND4
L'uscita vale VERO solo se tutti gli ingressi sono VERO



A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	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	1

NOR3
L'uscita vale VERO se e solo se nessuno degli ingressi è VERO



A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

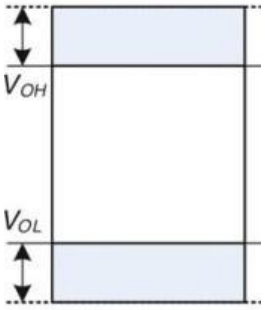
Porte logiche dal punto di vista fisico
Per realizzare le porte logiche in concreto i valori in ingresso binari devono corrispondere ad una certa grandezza fisica
La grandezza fisica che reifica i valori 0 e 1 è il potenziale elettrico che genera corrente all'interno di un circuito
Il valore logico 0 è rappresentato dal valore di potenziale di 0V **(GRD)**
Il valore logico 1 è rappresentato da un valore di potenziale **VDD** fornito dal generatore.
VDD<=1,5V ad oggi
Il **potenziale elettrico** è una grandezza fisica continua con cui rappresentiamo dei valori discreti (0 e 1) mediante le grandezze nominali GRD e VDD

Come ogni sistema fisico reale, un circuito digitale è soggetto a del **rumore** che può alterare i valori nominali su cui operiamo.
Per questo i componenti digitali hanno delle **soglie di tolleranza**

$GRD - V_{OL}$ e $V_{OH} - V_{DD}$

Per un corretto funzionamento di un circuito, il rumore non deve eccedere le soglie di tolleranza di tutti i suoi componenti e andare in zone "proibite" dove si hanno dei valori illegali.

Quindi abbiamo un intervallo di potenziale per cui se stiamo nei termini di tolleranza lo consideriamo GRD o VDD altrimenti se eccediamo questi termini di tolleranza VOL e VOH avremo un rumore così forte che non permette al circuito di operare in un regime digitale e quindi potrebbe malfunzionare o bruciarsi



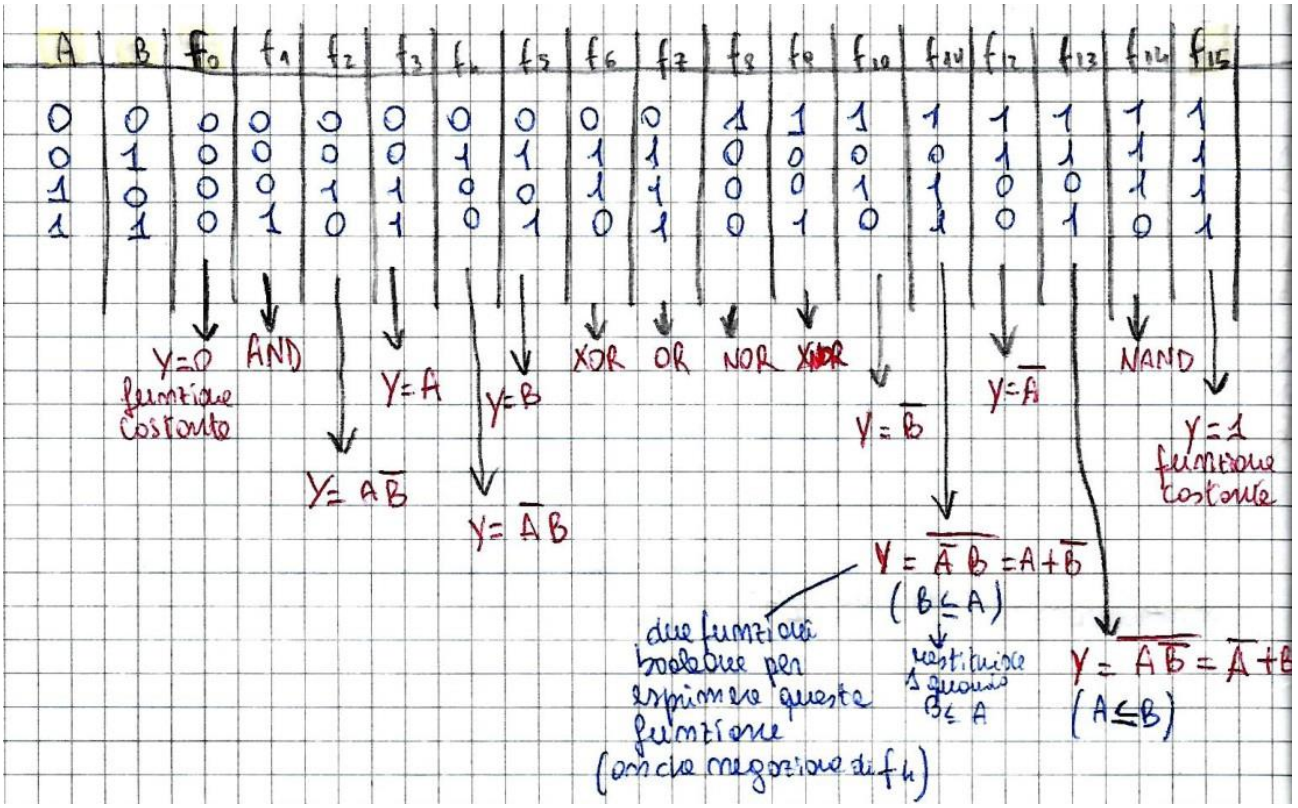
Funzioni booleane
Le porte logiche esaminate finora costituiscono delle specifiche funzioni booleane

$$f:\{0,1\}^N \rightarrow \{0,1\}$$

Quante funzioni booleane di N variabili esistono?
Data una parola di lunghezza N, abbiamo 2^n combinazioni possibili.
Per ogni combinazione di variabili una generica funzione f booleana può assegnare liberamente due valori 0 o 1 alla combinazione.
Quindi se la parola in ingresso è lunga N, avremo 2^n combinazioni possibili, alle quali si può associare 0 o 1.

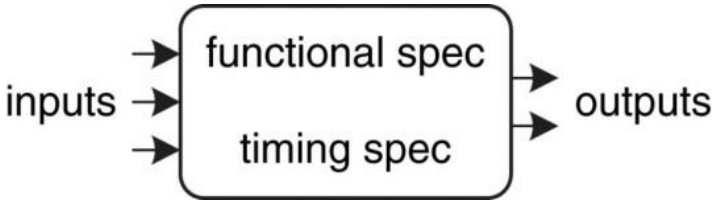
Quindi il numero di funzioni booleane di N variabili è pari al numero di parole binarie di lunghezza 2^N, ovvero:

$$2^{2^N}$$



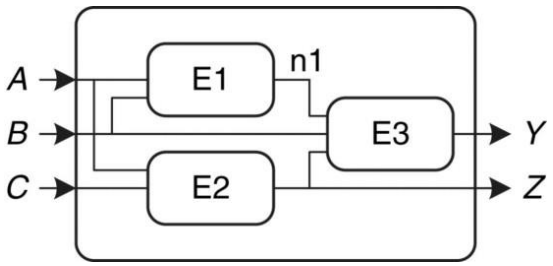
Circuiti digitali
Un **circuito digitale** è una rete che elabora segnali discreti (rappresentati da variabili booleane).

Un circuito digitale può essere visto come una **black-box** che contiene:
- **Uno o più input**
- **Uno o più output**
- **Una specifica funzionale** che rappresenta la relazione fra input e output
- **Una specifica temporale** che descrive il ritardo che intercorre affinché i segnali di input si propaghino nel circuito fino agli output.



Un circuito al suo interno è composta da **elementi** e **nodi**
elemento: è esso stesso un circuito digitale

nodo: è una connessione che trasporta il segnale, può essere di input, output o interno
- **nodo di input**: riceve il segnale dal mondo esterno
- **nodo di output:** riporta il segnale al mondo esterno
- **nodo interno:** connette due elementi



Reti combinatorie e sequenziali
Vi sono due grandi categorie di circuiti digitali: le **reti combinatorie** e le **reti sequenziali**.

rete combinatoria: i valori degli output dipendono esclusivamente dal valore corrente degli input. si dicono **memoryless**, ovvero non hanno memoria della "storia" precedente del circuito
rete sequenziale: gli output dipendono non solo dal valore corrente degli input, ma anche dai valori precedenti. Si dice quindi che il circuito ha memoria.

Regole di composizione di reti combinatorie
1) Ogni elemento circuitale è esso stesso una rete combinatoria
2) Ogni nodo della rete che non sia di input è connesso ad un unico output di un altro elemento della rete

3) la rete non contiene percorsi ciclici: ogni cammino interno alla rete visita un nodo al più una volta

Visualizzare una rete come una scatola nera con un'interfaccia e una funzione ben definite è un'applicazione del principio dell'astrazione e della modularità, così come la costruzione di una rete a partire da elementi circuitali più piccoli è un'applicazione del principio della gerarchia. Le regole della composizione combinatoria sono, infine, un'applicazione della regolarità.

La specifica funzionale di un circuito combinatorio è solitamente espressa come una tabella di verità o come un'espressione booleana.
In generale si può ricavare un'espressione booleana data una tabella di verità
oppure si può ricavare una tabella di verità data una espressione booleana

Ricavare una tabella di verità data un'espressione booleana
Le espressioni booleane si basano su variabili che possono assumere i due valori VERO o FALSO
Data un espressione booleana è possibile ricavare la tavola di verità calcolando a ritroso le tabelle di verità delle sotto espressioni dell'espressione booleana principale, partendo dalle singole variabili booleane.

Terminologia

litterale: una variabile booleana A o la sua negata A*

implicante: è un prodotto di litterali

mintermine: Dato un insieme K di variabili booleane, un mintermine di K è un **implicante**

che comprende tutte le variabili in K, siano esse positive o negate.
Quindi il mintermine è il prodotto di tutti gli ingressi di una funzione

maxtermine: Dato un insieme K di variabili booleane, un maxtermine di K è una **somma di litterali** in cui occorrono tutte le variabili in K.

Quindi il maxtermine è la somma di tutti gli ingressi di una funzione.

Forma SOP (Sum of product)
1=variabile in forma dritta 0=variabile
in forma negata

Ogni riga di una tabella delle verità è associata a un **mintermine** che è VERO per quella riga.

Si scrive un'espressione booleana a partire dalla tabella di verità tramite la somma di tutti i mintermini in corrispondenza dei quali l'uscita della tabella, Y, vale VERO= 1

Forma POS (Product of sum)

E' una forma duale per rappresentare una funzione booleana
1=variabile in forma negata
0=variabile in forma dritta

Ad ogni riga di una tabella di verità corrisponde un **maxtermine** che è FALSO solo per quella riga
Si scrive un'espressione booleana a partire dalla tabella di verità tramite il prodotto di tutti i maxtermini in corrispondenza dei quali l'uscita della tabella, Y, vale FALSO=0.

SOP – sum-of-products

O	C	E	minterm
0	0	0	$\overline{O}\ \overline{C}$
0	1	0	$\overline{O}\ C$
1	0	1	$O\ \overline{C}$
1	1	0	$O\ C$

$$E = O\overline{C}$$
$$= \Sigma(2)$$

POS – product-of-sums

O	C	E	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$E = (O + C)(O + \overline{C})(\overline{O} + \overline{C})$$
$$= \Pi(0, 1, 3)$$

Forma SOP o POS?

- Se la tabella di verità **ha pochi 1** allora la forma **SOP** è più succinta della forma POS
- Se la tabella di verità **ha pochi 0** allora la forma **POS** è più succinta della forma SOP
- Nel caso in cui il numero di 0 e 1 è pressappoco lo stesso le due forme si equivalgono

Algebra di boole

Un'espressione booleana ricavata a partire da una tabella delle verità tramite forma POS o SOP non sempre corrisponde all'insieme minimo di porte logiche necessario per realizzare la funzione considerata. **Una funzione booleana può essere semplificata per ricavare da essa un'espressione minimizzata più succinta.**

Proprio come si utilizza l'algebra per semplificare le espressioni matematiche, è possibile utilizzare l'algebra booleana per semplificare le espressioni booleane

L'algebra booleana si basa su un insieme di **assiomi** che come tali vengono per definizione considerati corretti. A partire da essi è possibile invece dimostrare tutti i teoremi dell'algebra booleana.

I postulati e i teoremi dell'algebra booleana obbediscono al **principio di dualità: se i simboli 0 e 1 e gli operatori • (AND) e + (OR) sono scambiati tra loro, l'affermazione rimane corretta.**

Assiomi dell'algebra di Boole

	Axiom		Dual	Name
A1	$B=0$ if $B\neq 1$	A1′	$B=1$ if $B\neq 0$	Binary field
A2	$\overline{0}=1$	A2′	$\overline{1}=0$	NOT
A3	$0 \bullet 0=0$	A3′	$1+1=1$	AND/OR
A4	$1 \bullet 1=1$	A4′	$0+0=0$	AND/OR
A5	$0 \bullet 1=1 \bullet 0=0$	A5′	$1+0=0+1=1$	AND/OR

A1 e A1' ci dicono che il valore di una variabile booleana può essere 0 oppure 1
A2 e A2' definiscono l'operatore NOT (di fatto questi assiomi ripropongono la tabella di verità dell'operatore)
A3, A4 e A5 definiscono l'operatore AND
A3', A4' e A5' definiscono l'operatore OR

Teoremi ad una variabile

	Theorem		Dual	Name
T1	$B \bullet 1=B$	T1′	$B+0=B$	Identity
T2	$B \bullet 0=0$	T2′	$B+1=1$	Null Element
T3	$B \bullet B=B$	T3′	$B+B=B$	Idempotency
T4		$\overline{\overline{B}}=B$		Involution
T5	$B \bullet \overline{B}=0$	T5′	$B+\overline{B}=1$	Complements

Teoremi più variabili: dualità

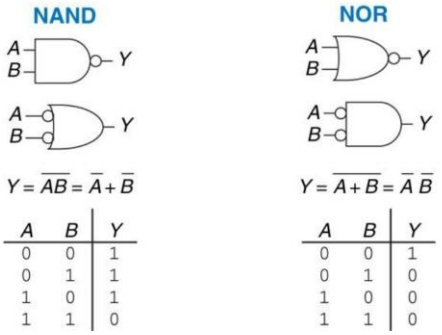
#	Theorem	Dual	Name
T6	B•C = C•B	B+C = C+B	Commutativity
T7	(B•C) • D = B • (C•D)	(B + C) + D = B + (C + D)	Associativity
T8	B • (C + D) = (B•C) + (B•D)	B + (C•D) = (B+C) (B+D)	Distributivity
T9	B • (B+C) = B	B + (B•C) = B	Covering
T10	(B•C) + (B•$\overline{C}$) = B	(B+C) • (B+$\overline{C}$) = B	Combining
T11	(B•C) + ($\overline{B}$•D) + (C•D) = (B•C) + ($\overline{B}$•D)	(B+C) • ($\overline{B}$+D) • (C+D) = (B+C) • ($\overline{B}$+D)	Consensus

Teoremi di De Morgan

#	Theorem	Dual	Name
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ = $\overline{B_0}+\overline{B_1}+\overline{B_2}\ldots$	$\overline{B_0+B_1+B_2\ldots}$ = $\overline{B_0}\bullet\overline{B_1}\bullet\overline{B_2}\ldots$	DeMorgan's Theorem

- La negata di un prodotto è uguale alla somma delle negate
- La negata di una somma è uguale al prodotto delle negate

- I teoremi di De Morgan consentono di convertire un porta AND in una OR e viceversa



Secondo il teorema di De Morgan, una porta NAND è equivalente a una porta OR con gli ingressi negati. Allo stesso modo, una porta NOR è uguale a una porta AND con gli ingressi negati.

Il principio di dualità è una conseguenza dei teoremi di De Morgan

Tecniche di dimostrazione per dimostrare l'equivalenza di due espressioni

Perfect induction: si calcolano le rispettive tabelle di verità delle due espressioni e se coincidono (negli output) allora le due espressioni sono equivalenti
Limiti: Al crescere della lunghezza delle espressioni diventa sempre più laboriosa

Proof teoretico: si usano assiomi e teoremi precedentemente provati per manipolare le espressioni fino ad ottenere espressioni uguali

Per casi rilevanti: si sfrutta la struttura delle espressioni e si verifica che le tabelle di verità coincidono solo nei casi rilevanti

Teoremi di De Morgan e forme SOP/POS
I teoremi di De Morgan possono essere usati per ridurre una generica espressione booleana in forma SOP/POS senza «passare» per la tabella di verità

- **Si applica «De Morgan» per negare la struttura della formula**
- **Applica esaustivamente la proprietà distributiva dell'AND sull'OR (se si vuole SOP)**
- **Applica esaustivamente la proprietà distributiva dell'OR sull'AND (se si vuole POS)**

I teoremi di De Morgan consentono anche di ricavare la forma POS a partire dalla SOP (e viceversa)

A	B	Y	$\bar{Y}$	minterm
0	0	0	**1**	$\bar{A}\,\bar{B}$
0	1	0	**1**	$\bar{A}\,B$
1	0	1	**0**	$A\,\bar{B}$
1	1	1	**0**	$A\,B$

$$\bar{Y} = \bar{A}\bar{B} + \bar{A}B$$

$$\bar{\bar{Y}} = \overline{\bar{A}\bar{B} + \bar{A}B} = \overline{\bar{A}\bar{B}} \cdot \overline{\bar{A}B} = (A + B) \cdot (A + \bar{B})$$

Completezza degli operatori
La forma SOP usa solo gli operatori NOT, AND, OR per rappresentare qualsiasi espressione booleana
Questo vuol dire che questo insieme di operatori costituisce un insieme completo, per il fatto che si possono ricavare tutti gli altri operatori a partire da questi tre.

Ma per le leggi di De Morgan:

- $A \cdot B = \overline{\bar{A} + \bar{B}}$ (AND può essere espresso in termini di NOT e OR)
- $A + B = \overline{\bar{A} \cdot \bar{B}}$ (OR può essere espresso in termini di NOT e AND)

Quindi {NOT, AND, OR} è un insieme completo ma non minimale essendo

{AND,NOT} e {OR, NOT} anche loro completi, ma minimali

Semplificare le espressioni booleane, semplificazione di forma SOP, SOP minima

Avere una forma più succinta di un espressione booleana significa anche avere un circuito combinatorio con meno porte logiche e quindi più economico e performante.

L'idea è costruire un circuito combinatorio quanto più ottimizzato possibile per la data funzione booleana da calcolare

Alcuni teoremi per semplificare espressioni

- Distributivity (T8, T8') $B(C+D) = BC + BD$; $B + CD = (B+C)(B+D)$
- Covering (T9') $A + AP = A$
- Combining (T10) $P\overline{A} + PA = P$
- Expansion $P = P\overline{A} + PA$; $A = A + AP$
- Duplication $A = A + A$
- "Simplification" theorem $P\overline{A} + A = P + A$; $PA + \overline{A} = P + \overline{A}$

I teoremi dell'algebra booleana sono utili per semplificare le espressioni booleane.
Il principio base per semplificare equazioni in forma SOP è combinare i termini utilizzando la relazione PA + PA* = P, dove P rappresenta un implicante qualsiasi. Nel minimizzare una SOP, può essere anche necessario «sdoppiare» un implicante allorché questo può essere ridotto in modi differenti.

La proprietà principale che si usa è il combining $(B+C) \bullet (B+\overline{C}) = B$

Quanto si può semplificare un'espressione?

Si definisce un'espressione SOP come **minima** se utilizza il **minor numero possibile di implicanti, quindi se tutti gli implicanti sono implicanti primi**.

Se si confrontano espressioni con lo stesso numeri di implicanti, l'espressione minima è quella che usa il **minor numero possibile di letterali**.

Un implicante è detto **implicante primo** se non può essere combinato con nessun altro elemento all'interno dell'espressione per formare un nuovo implicante con un numero minore di letterali.

Schemi circuitali, logica a due e più livelli

- Ad ogni espressione booleana corrisponde in circuito combinatorio, rappresentato da uno schema circuitale

Schemi circuitali SOP: le espressioni booleane in forma SOP hanno degli schemi circuitali molto regolari

- abbiamo una linea di input per ogni variabile che occorre positiva
- abbiamo linee con una porta NOT per ogni variabile che occorre negata
- ogni **mintermine** si costruisce con una porta **AND** che ha in ingresso le linee che corrispondono ai relativi litterali che lo compongono
- tutte le uscite delle porte AND che corrispondono ai mintermine vengono collegate in un unico **OR,** che rappresenta la **somma dei mintermini.**

Per questo la logica in forma SOP viene chiamata logica a due livelli, perché consiste di letterali connessi a un primo livello di porte AND che, a sua volta, sono connesse a un secondo livello di porte OR

Logiche a più livelli: Spesso i progettisti costruiscono reti con più di due livelli di porte logiche, perché può accadere che queste reti combinatorie a più livelli richiedano di utilizzare meno hardware (porte logiche) rispetto alle loro controparti a due livelli. Vedi esempio XOR3

La scelta della logica di realizzazione circuitale dipende da diversi fattori, fra cui la tecnologia di costruzione del circuito utilizzata.

Vantaggi forme SOP/POS logiche a due livelli

La logica a 2 livelli della forma SOP presenta dei vantaggi, ad esempio, nei tempi di propagazione che sono molto stabili e veloci al mutare di qualsiasi valore delle variabili booleane

Logiche non a due livelli possono avere tempi di propagazione più lunghi dovuto al delay delle diverse porte che si accumula di volta in volta, possiamo trovarci in situazioni in cui si verifica un critical path e si deve passare fra tante porte logiche per ottenere il risultato e quindi i tempi di propagazione si allungano.

Limiti SOP

Alcune funzioni booleane, poste in forma SOP, sono estremamente poco succinte e quindi richiedono un numero considerevole di porte per realizzare il circuito

Es. XOR a più variabili

- XOR3 in forma SOP

$$Y = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

la funzione può essere realizzata + semplicemente con una cascata di XOR a due ingressi, logica multilivello

Circuito a priorità

I circuiti a priorità vengono utilizzati per assegnare una risorsa condivisa secondo un ordine di priorità fra chi ne fa richiesta

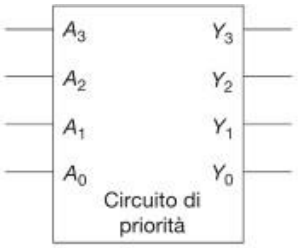
Ad esempio posso avere 4 possibili richiedenti con priorità diverse

A3 ha la priorità più alta, A2, A1, A0 che ha la priorità più bassa.

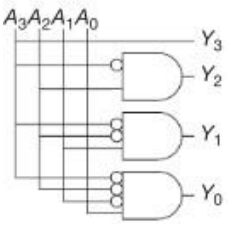
- **Gli input A0 ,…, A3 rappresentano le richieste della risorsa**
- **Gli output Y0 ,…, Y3 rappresentano a chi viene assegnata la risorsa, di volta in volta uno solo di essi sarà uguale a 1, ossia quello che ha la priorità più alta.**

Se per esempio A3 e A2 richiedono una risorsa, questa viene assegnata ad A3 perchè ha priorità più alta.

Ad A0 che ha la priorità più bassa ad esempio la risorsa viene assegnata solo se gli altri richiedenti non ne hanno bisogno, quindi se A3,A2,A1=0



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



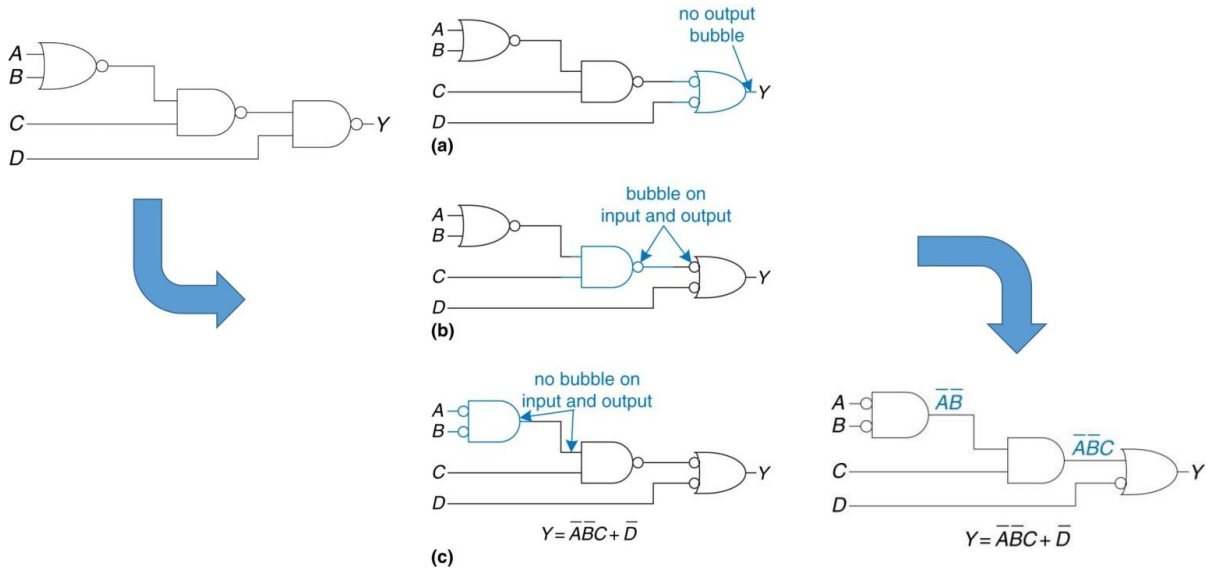
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

Valori don't cares

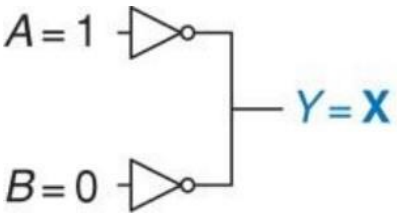
Quando l'ingresso A3 è portato a uno nel circuito, le uscite non tengono conto dei valori presenti agli altri ingressi. Tali ingressi, ignorati dalle uscite, vengono contrassegnati con il simbolo X, e sono chiamati valori **don't cares, non conta il loro valore al fine di calcolare l'uscita della funzione**.La tabella di verità si riduce sensibilmente quando si inseriscono i

valori don’t cares.. A partire da questa tabella delle verità è più semplice leggere l’espressione booleana in forma somma di prodotti ignorando gli ingressi con una X.
Valori «don’t care» (X) si possono avere anche in output allorché per una data configurazione degli input il valore dell’output è irrilevante (non ci interessa se sia 0 o 1)

Bubble pushing
Usare logiche multilivello e porte NAND/NOR a volte rende difficile capire quale funzione booleana un circuito realizza a causa delle molte negazioni annidate
Per avere un'espressione un po’ più leggibile si possono applicare le leggi di De Morgan
Il bubble pushing è una tecnica grafica che applica le leggi di De Morgan e la legge della doppia negazione per eliminare le negazioni annidate nei circuiti con tante porte NAND e NOR, per semplificarli e renderli più leggibili
Partendo dall’output Y si applicano a ritroso le leggi di De Morgan in modo che l’input e l’output di ogni nodo siano entrambi positivi o negati



Valori illegali



In questo circuito il valora del potenziale è portato ad essere contemporaneamente sia 0 che VDD. Questo, generalmente, accade se al nodo vengono applicati entrambi i valori 0 e 1 allo stesso tempo.
Questo tipo di configurazioni vengono dette illegali

La tensione elettrica effettiva di un nodo è un valore indefinito compreso tra 0 e VDD e per questo configurazioni del genere possono indurre ad una forte dissipazione che riscalda e danneggia il circuito **(corto circuito)**

Stato di alta impedenza

Un nodo, oltre ad essere in uno stato 0/1, può anche essere in un stato **Z floating o di alta impedenza nel quale non c'è passaggio di corrente**

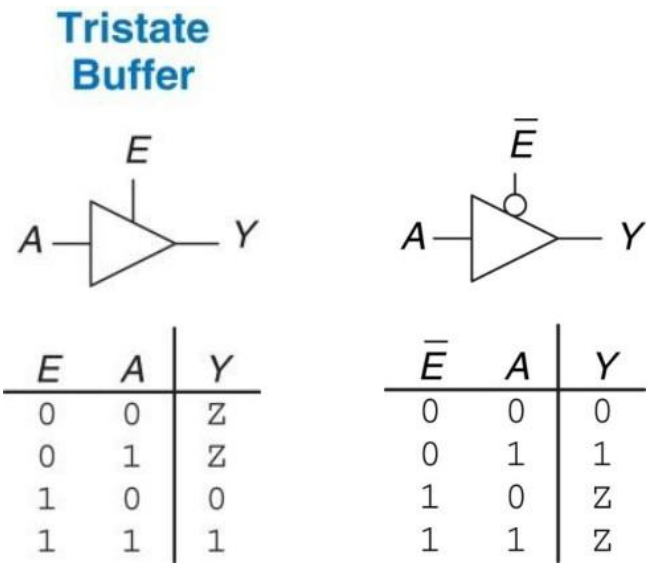
Gli stati di alta impedenza vengono utilizzati per disconnettere una parte di un circuito dal resto. Un circuito che realizza tale funzione è il tristate buffer

Tristate buffer

Il buffer tristate, possiede tre possibili stati d'uscita:
ALTO (1), BASSO (0) e alta impedenza (Z).

Ha un ingresso A, un'uscita Y, e un'abilitazione **E (Enable).**
Quando il segnale di abilitazione **E=1**, il buffer lavora come un **buffer normale** e trasferisce il valore dell'ingresso all'uscita.
Quando invece il segnale di abilitazione **E=0**, l'uscita assume il valore di **alta impedenza (Z)**

il segnale di abilitazione E può essere attivo alto oppure attivo basso E*, se è attivo basso quando E=1 avremo lo stato di alta impedenza



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

$\overline{E}$	A	Y
0	0	0
0	1	1
1	0	Z
1	1	Z

I buffer tristate vengono solitamente utilizzati sui bus che connettono più chip tra loro.
Per esempio, un microprocessore, un controllore video o un controllore Ethernet possono tutti aver bisogno di comunicare con la memoria, a cui sono collegati col medesimo bus.
In questi casi, per evitare stati illegali, solo un componente alla volta può «immettere» segnali sul bus per comunicare con la memoria.

Per questo ogni chip viene collegato al bus di memoria condiviso tramite un buffer tristate.
Quando un componente comunica con la memoria gli altri componenti i devono essere temporaneamente disconnessi tramite lo stato di alta impedenza del tristate buffer

Mappe di Karnaugh

Le mappe di Karnaugh (K-map) sono un metodo grafico di semplificazione di espressioni booleane in forma SOP. Esse consentono di rilevare più facilmente implicati che possono essere semplificati

- Quindi alla base delle mappe di Karnaugh c'è il solito principio:

$$PA + P\overline{A} = P$$

Ogni quadrato della mappa di Karnaugh corrisponde a una riga della tabella delle verità e contiene il valore, corrispondente a quella riga, dell'uscita Y

Nella mappa si inseriscono tutte le combinazioni possibili delle variabili booleane presenti nelle espressioni, ogni riquadro della mappa differisce per un singolo cambiamento di una variabile booleana

le combinazioni delle variabili compaiono in un ordine particolare: 00, 01, 11, 10. Questo ordine è chiamato **codice Gray**. è utile proprio perché gli elementi adiacenti differiscono per una sola variabile. scrivere le combinazioni seguendo l'ordine binario ordinario non avrebbe prodotto come risultato la proprietà utile delle mappe di Karnaugh dei riquadri adiacenti con un'unica variabile diversa.

Le mappe di Karnaugh ci permettono di eseguire la semplificazione graficamente, cerchiando gli 1 nei riquadri adiacenti. Per ogni cerchio si deve poi scrivere l'implicante corrispondente. Bisogna utilizzare il minor numero possibile di cerchi e includendo in ogni cerchio il maggior numero possibile di riquadri

Regole di minimizzazione con K-maps

- Usare il minimo numero di cerchi/bolle necessari per ricoprire tutti gli 1
- Tutte le caselle in una bolla contenere un 1
- Ogni cerchio deve ricoprire un blocco di caselle che è una potenza di 2: 1,2,4,…
- Ogni cerchio deve essere il più largo possibile
- Un cerchio può estendersi oltre i bordi e comprendere le estremità della K-map (struttura toroidale)
- Una casella può essere ricoperta da più cerchi se questo permette un numero inferiore di cerchi

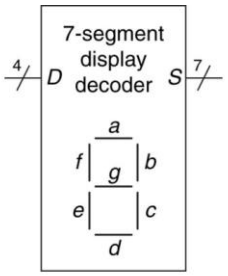
Don't care e K-map

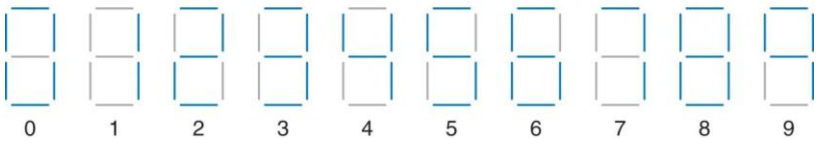
In un circuito valori «don't care» (X) si possono avere anche in output allorché per una data configurazione degli input il valore dell'output è irrilevante

In particolare in una K-map sostituiremo i valori X don't care, se li abbiamo, con degli 1 se questo consente di avere un numero inferiore di cerchi o cerchi più larghi

Display a 7 segmenti

Un display a sette segmenti riceve un dato di ingresso a 4 bit D3:0 e produce sette uscite per controllare 7 diodi emettitori di luce (LED, Light Emitting Diode) che visualizzano una cifra da 0 a 9. Le sette uscite sono spesso chiamate segmenti da a a g, o Sa–Sg





$D_{3:0}$	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

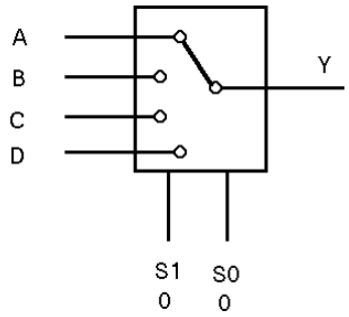
nel display a sette segmenti per gli input «illegali» 10-15 l'output può essere di tipo X, don't care

BLOCCHI COSTITUTIVI COMBINATORI

La logica combinatoria viene spesso raggruppata in blocchi costitutivi più ampi per costruire sistemi più complessi. Questa è chiaramente un'applicazione del principio dell'astrazione, che nasconde i dettagli non necessari a livello delle porte logiche per enfatizzare la funzione del blocco costitutivo.

Multiplexer

Un multiplexer è essenzialmente un selettore di linea, ossia un circuito che è in grado di selezionare un'uscita a partire da un certo numero di ingressi possibili , basandosi sul valore di un segnale di selezione. I multiplexer sono anche chiamati mux.



In generale è costituito da N ingressi (dove N è una potenza di 2), 1 uscita, e log in base 2 di N linee di selezione che indicano a quale ingresso deve corrispondere l'output

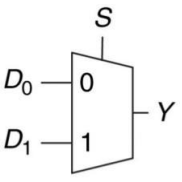
Multiplexer 2:1

Un mux 2:1 presenta due ingressi D0 e D1, un ingresso per il segnale di abilitazione S e un'uscita Y.

Il multiplexer sceglie tra i due ingressi a seconda del valore assunto da S:
-se S = 0, Y = D0

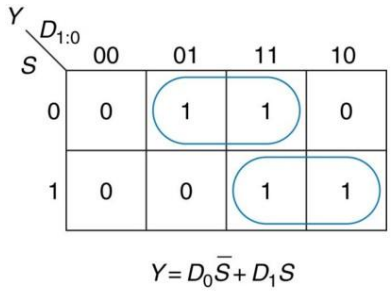
-se S = 1, Y = D1.

S viene anche chiamato **segnale di controllo o di abilitazione** proprio perché controlla la scelta del multiplexer.

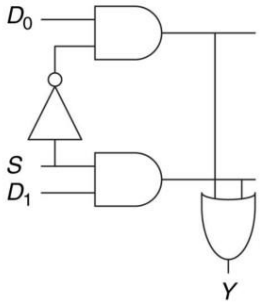


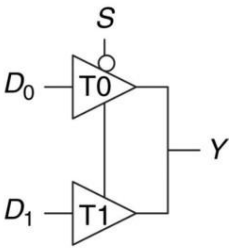
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Un multiplexer 2:1 può essere costruito a partire dalla logica SOP costruendo il circuito sulla base dell'espressione SOP minimizzata.
In alternativa, i multiplexer possono essere realizzati con **buffer tristate**



$Y = D_0\overline{S} + D_1 S$

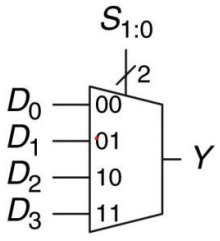


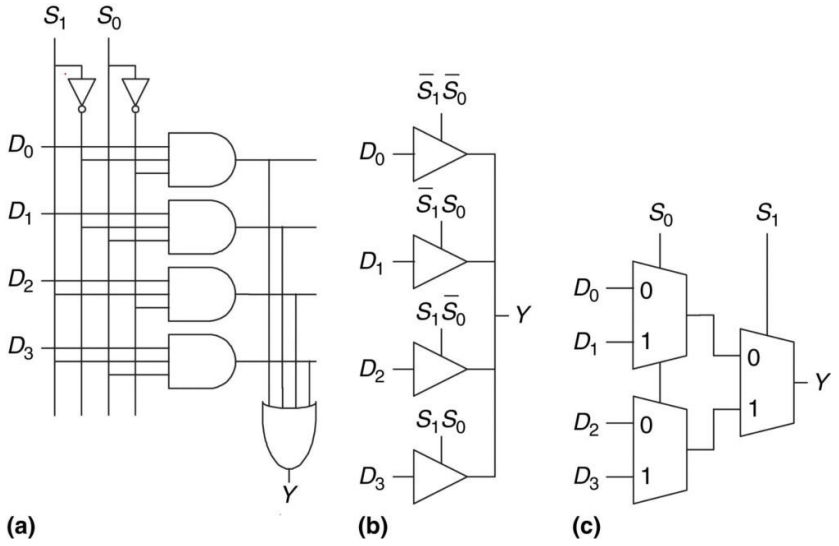


$Y = D_0\overline{S} + D_1 S$

le abilitazioni dei buffer sono organizzate in maniera tale che, in ogni momento, solo un buffer tristate è attivo. Quando S = 0, viene attivato il tristate T0, permettendo così a D0 di passare in Y; quando invece S = 1, viene attivato il tristate T1, che permette a D1 di passare in Y

Multiplexer 4:1
Un multiplexer 4:1 possiede quattro ingressi e un'uscita. In questo caso sono necessari due segnali di selezione per scegliere tra i quattro ingressi in input.
Il multiplexer 4:1 può essere costruito utilizzando la logica sop, i tristate,
o alcuni multiplexer 2:1 in cascata

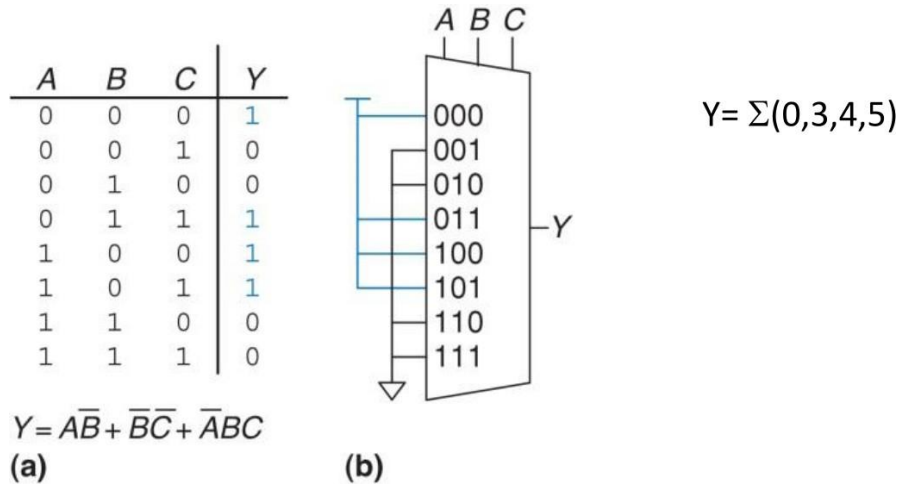




Sintetizzare funzioni booleane con mux

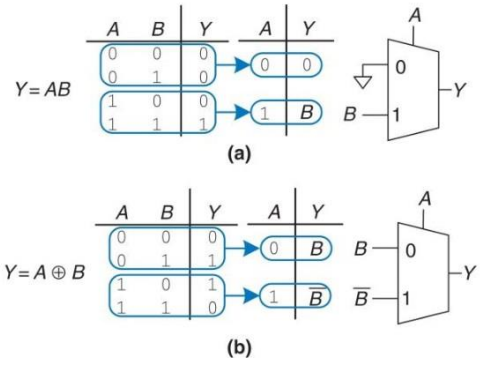
I multiplexer possono essere usati anche per sintetizzare delle funzioni booleane

Sintetizzare una funzione di m variabili con un mux a 2^m linee è molto semplice: le variabili saranno linee di selezione. Data una certa configurazione delle variabili, la linea di ingresso corrispondente sarà posta al valore della funzione in quella configurazione, ossia 1(VDD) o GRD (0). Di fatto le linee di ingresso riproducono la tabella di verità della funzione

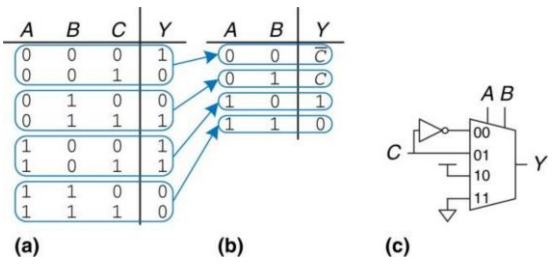


$Y = \Sigma(0,3,4,5)$

- E' possibile utilizzare un mux con 2^{m-1} ingressi per sintetizzare un funzione ad m variabili
- Le prime m-1 variabili saranno linee di selezione, mentre le linee di ingresso possono essere poste a 0,1, oppure all'ultima variabile (positiva o negata)



In ingresso possiamo avere anche delle variabili per sintetizzare la funzione booleana e per avere una tavola di verità più succinta

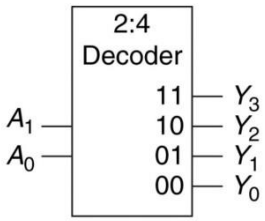


Decoder

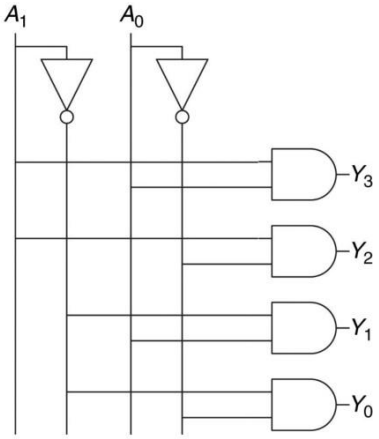
Un decoder ha N linee di ingresso e 2^N linee di uscita e attiva una delle sue uscite a seconda della combinazione dei valori in ingresso.

se n il è numero rappresentato dagli input allora solo l'n-esima linea di uscita èpari a 1 mentre tutte le altre sono a 0

Le uscite sono dette one hot proprio perché solo un'uscita alta in ogni momento



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Anche i decoder possono essere usati per sintetizzare funzioni booleane
Basta mettere in OR tutte le linee di uscita che occorrono nell'espressione della funzione da sintetizzare