

INTRODUZIONE

Nell'elettronica digitale, un circuito logico è una rete di dispositivi elettronici.

Ogni circuito è provvisto di un'insieme di **INPUT** che vengono processati all'interno di esso, restituendo uno o più **OUTPUT**. Oltre ai suoi input e ai suoi output, ogni circuito viene descritto da una **SPECIFICA FUNZIONALE** (il modo in cui il circuito agisce sugli input per generare gli output), e una **SPECIFICA TEMPORALE** (tempo impiegato dal circuito per processare gli input e generare gli output).

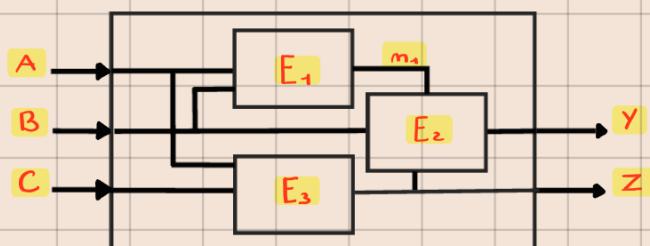


I circuiti logici possono essere visti come delle **SCATOLE NERE** dove bisogna tener conto degli input forniti e degli output che vengono restituiti.

Al loro interno possiamo trovare **NODI E ELEMENTI** (E_1, E_2, E_3, \dots), che spesso sono a loro volta anche essi dei circuiti logici.

I modi, invece, sono dei fili elettrici il cui voltaggio trasporta un valore discreto, i modi possono essere:

- **NODI DI INPUT** (A, B, C, \dots);
- **NODI DI OUTPUT** (Y, Z, \dots);
- **NODI INTERNI AL CIRCUITO STESSO** (m_1, \dots).



I circuiti logici possono essere divisi in due macro-categorie:

• **CIRCUITI COMBINATORI** che sono senza memoria, quindi il valore degli output dipende solo dagli input e dal modo in cui vengono elaborati al suo interno;

• **CIRCUITI SEQUENZIALI** che sono dotati di memoria, quindi il valore degli output dipende sia dal valore degli input nel momento precedente all'elaborazione, sia dal valore che hanno assunto nelle precedenti elaborazioni.

Per poter essere definito come combinatorio, un circuito logico deve rispettare determinate caratteristiche:

- OGNI ELEMENTO DEL CIRCUITO DEVE ESSERE A SUA VOLTA COMBINATORIO;
- OGNI NODE È UN INPUT OPPURE SI CONNETTE AD ESATTAMENTE UN OUTPUT;
- NEL CIRCUITO NON SONO PRESENTI DEI COLLEGAMENTI CICLICI.

PORTE LOGICHE

La funzione delle porte logiche è quella di ricevere in input una o più valori binari per poi produrre un output a sua volta binario.

Tra i circuiti digitali il valore assunto in input ed in output è descritto da due semplici relazioni:

• SE C'È PASSAGGIO DI CORRENTE;

• SE NON C'È PASSAGGIO DI CORRENTE.

La relazione tra gli input e gli output delle porte logiche vengono descritte tramite delle **TABELLE DELLA VERITÀ** e tramite le **EQUAZIONI BOOLEANE**.

Esistono vari tipi di porte logiche rappresentate graficamente e sono:

• **SINGLE INPUT LOGIC GATES** e sono:



$$Y = A$$



$$Y = \bar{A}$$

A	Y
0	0
1	1

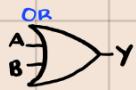
A	Y
0	1
1	0

• **TWO INPUT LOGIC GATES** e sono:



$$Y = AB$$

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



$$Y = A + B$$

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



$$Y = A \oplus B$$

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



$$Y = \overline{AB}$$

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



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

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



$$Y = \overline{A \oplus B}$$

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

• **MULTIPLE INPUT LOGIC GATES** e sono:



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

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



$$Y = ABC$$

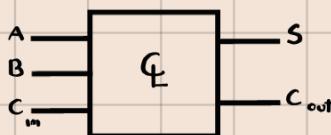
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

EQUAZIONI BOOLEANE

Il seguente circuito logico (CL) è dotato di 3 input (A, B e C_{in}) e 2 output (S, C_{out}) e può essere descritto sia in forma scritta sia in forma grafica.

$$S = F(A, B, C_{im})$$

$$C_{out} = f(A, B, C_{in})$$



Per descrivere le specifiche funzionali del circuito, è necessario definire le equazioni booleane che mettono in relazione gli input con gli output.

In questo esempio ci sono le 2 funzioni che generano gli output S e Cout come:

$$S = A \oplus B \oplus C_m$$

$$C_{out} = AB + AC_{im} + BC_{im}$$

In aggiunta, possiamo anche definire la **TABELLA DELLA VERITÀ** corrispondente all'equazione descritta:

Senza saperlo, abbiamo descritto completamente un **SOMMATORE AD 1 BIT CON RIPORTO**, dove s corrisponde al valore ottenuto dalla somma dei bit A, B e Cin , ossia il possibile riporto di una precedente somma, mentre $Cout$ corrisponde al riporto generato dalla somma stessa.

Prima di poter parlare dettagliatamente di equazioni booleane, è necessario introdurre una serie di terminologie:

COMPLEMENT: l'opposto stretto di una variabile $\Rightarrow \bar{A}$ è il complemento di A ;

LITERAL: una variabile o un suo complemento $\Rightarrow A, \bar{A}, B, \bar{B}$;

IMPICANT: un prodotto tra letterali $\Rightarrow \bar{A}\bar{B}, \bar{A}\bar{C}, ABC$;

MINTERM: un prodotto tra tutte le variabili in input $\Rightarrow ABC, A\bar{B}C, \bar{A}\bar{B}C$;

MAXTERM: una somma tra tutte le variabili in input $\Rightarrow A+B+C, A+\bar{B}+C, \bar{A}+\bar{B}+C$.

SONDA DI PRODOTTI(SOP) E PRODOTTO DI SOMME(POS)

Due ulteriori modi per rappresentare le equazioni booleane è attraverso le forme canoniche di **SUM-OF-PRODUCTS(SOP)** e **PRODUCT-OF-SUMS(POS)**.

Nella **FORMA SOP**: ad ogni riga della tabella della verità, descrivente il rapporto tra input e output, corrisponde un **MINTERM** tra le variabili della funzione, dove gli 0 assumono la forma di **COMPLEMENTARE**, mentre gli 1 non vengono negati.

Infine, la funzione descritta dall'equazione booleana viene rappresentata come la somma dei minterm il cui output è 1.

Nella **FORMA POS**: ad ogni riga della tabella della verità corrisponde un **MAXTERM** tra le variabili della funzione, dove gli 1 assumono la forma di **COMPLEMENTARE**, mentre gli 0 non vengono negati.

Infine, la funzione descritta dall'equazione booleana viene rappresentata come il prodotto dei maxterm il cui output è 0.

Proviamo a descrivere l'equazione $Y = A \oplus B$ prima in forma SOP e poi in forma POS:

A	B	MINTERM	MAXTERM	Y
0	0	$\bar{A} \cdot \bar{B}$	$A+B$	0
0	1	$\bar{A} \cdot B$	$A+\bar{B}$	1
1	0	$A \cdot \bar{B}$	$\bar{A}+B$	1
1	1	$A \cdot B$	$\bar{A}+\bar{B}$	0

$$\text{FORMA SOP} \Rightarrow Y = F(A,B) = \bar{A}B + A\bar{B} = \Sigma(1,2)$$

$$\text{FORMA POS} \Rightarrow Y = F(A,B) = (A+B)(\bar{A}+\bar{B}) = \Pi(0,3)$$

ALGEBRA DI BOOLE

L'algebra introdotta dal matematico George Boole, seppur molto poco utilizzata nel periodo in cui venne introdotta, trovò un'applicazione estremamente efficiente nel campo dell'elettronica digitale, egli introdusse una serie di **ASSIOMI** e **TEOREMI** che dettano il rapporto tra i valori discreti 0 ed 1.

Tramite questi assiomi e teoremi è possibile semplificare notevolmente le equazioni booleane descriventi i circuiti logici.

La caratteristica fondamentale di essi è la loro **DUALITÀ**, ossia la completa intercambiabilità tra AND(\cdot) ed OR($+$) e tra 1 e 0 , senza alterare la veridicità dell'assioma o del Teorema.

• ASSIOMI BOOLEANI

Nome	Axioma	Duale
CAMPB BINARIO	$B=0 \Leftrightarrow B \neq 1$	$B=1 \Leftrightarrow B \neq 0$
NOT	$\bar{0} = 1$	$\bar{1} = 0$
AND/OR	$0 \cdot 0 = 0$	$1 + 1 = 1$
AND/OR	$1 \cdot 1 = 1$	$0 + 0 = 0$
AND/OR	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 + 0 = 0 + 1 = 1$

• TEOREMI BOOLEANI

Nome	Teorema	Duale
IDENTITÀ	$B \cdot 1 = B$	$B + 0 = B$
ELEMENTO NULLO	$B \cdot 0 = 0$	$B + 1 = 1$
IDEM - POTENZA	$B \cdot B = B$	$B + B = B$
INVOLUZIONE	$\bar{\bar{B}} = B$	$\overline{\overline{B}} = B$
COMPLEMENTARE	$\overline{B} \cdot \overline{B} = 0$	$\overline{B} + \overline{B} = 1$
P. COMMUTATIVA	$B \cdot C = C \cdot B$	$B + C = C + B$
P. ASSOCIAТИVA	$(B \cdot C) \cdot D = B \cdot (C \cdot D)$	$(B + C) + D = B + (C + D)$
P. DISTRIBUTIVA	$B(C+D) = (B \cdot C) + (B \cdot D)$	$B + (C \cdot D) = (B+C)(B+D)$
1° T. ASSORBIIMENTO	$B \cdot (B+C) = B$	$B + (B \cdot C) = B$
2° T. ASSORBIIMENTO	$(B \cdot C) + (B \cdot C) = B$	$(B+C) \cdot (B+C) = B$
T. DI DEMORGAN	$\overline{B_0 \cdot B_1 \cdot B_2 \dots} = \overline{B_0} + \overline{B_1} + \overline{B_2} \dots$	$\overline{B_0 + B_1 + B_2 \dots} = \overline{B_0 \cdot B_1 \cdot B_2 \dots}$

SEMPLIFICARE LE EQUAZIONI

Proviamo per esempio a semplificare l'equazione $y = \bar{A}BC + \bar{A}$:

1) applico la proprietà **DISTRIBUTIVA** in modo inverso $\Rightarrow \bar{A}(BC + 1)$

2) applico l'assioma dell'**ELEMENTO NULLO** $\Rightarrow \bar{A}(1) \Rightarrow \bar{A}$

3) applico il teorema dell'**IDENTITÀ** $\Rightarrow \bar{A}$

Possiamo quindi riscrivere l'equazione y come $y = \bar{A}BC + \bar{A} = \bar{A}$

Per verificare che la semplificazione sia valida possiamo usare la tabella della verità rappresentante entrambe le due forme dell'equazione:

A	B	C	$ABC + \bar{A}$	\bar{A}
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0

$$\begin{array}{ccc|c|c} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{array}$$

Proviamo ora a semplificare l'equazione $y = (A+BC)(A+DE)$. In questo caso possiamo usare la proprietà distributiva in 2 modi:

1) DUALE DELLA P.DISTRIBUTIVA IN MODO INVERSO:

$$\bullet (A+BC)(A+DE) = A + (BC \cdot DE) = A + BCDE$$

2) DISTRIBUISCO $(A+BC)$ IN $(A+DE)$ PER Poi APPLICARE IL DUALE DEL PRIMO TEOREMA DELL'ASSORBIENTO:

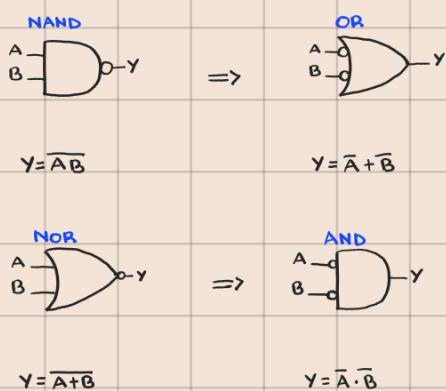
$$\bullet (A+BC)(A+DE) = AA + ADE + ABC + BCDE$$

$$\bullet AA + ADE + ABC + BCDE = A + ABC + BCDE = A + BCDE$$

IL TEOREMA DI DEHORGAN E IL BUBBLE PUSHING

Il TEOREMA DI DEMORGAN stabilisce che il COMPLEMENTO DEI PRODOTTI equivale alla SOMMA DEI COMPLEMENTI e che di conseguenza il suo duale afferma che il COMPLEMENTO DELLA SOMMA equivale al PRODOTTO DEI COMPLEMENTI.

Questo teorema ha anche una **FORMA GRAFICA**:

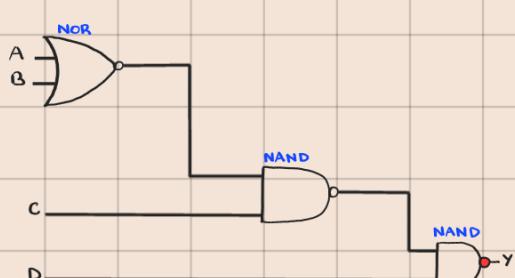


Come possiamo notare l'applicazione del teorema di DeMorgan corrisponde all'invertire AND con un OR (o viceversa) e allo spostare il cerchio del

NOT DALL'OUTPUT AGLI INPUT (o viceversa)

Per via di quest'ultima impostazione a livello grafico, nell'ambito della progettazione dei sistemi digitali è nata una convenzione, chiamata **BUBBLE PUSHING**, dove si cerca di spostare tutte le bolle NOT presenti nel circuito da sinistra verso destra (**FORWARD BUBBLE PUSHING**) oppure a destra verso sinistra (**BACKWARDS BUBBLE PUSHING**), rendendo più fluido il processo di semplificazione.

Proviamo a semplificare il seguente circuito utilizzando il **backwards bubble pushing**.



1) Partendo da destra utilizziamo il BUBBLE PUSHING sull'outout della porta NAND alla fine:





2) successivamente, possiamo **SEMPLIFICARE** il NOT in uscita dalla porta NAND centrale con il NOT in entrata alla porta OR appena modificata.

Inoltre, possiamo applicare nuovamente il **BUBBLE PUSHING** sull'output della porta NOR iniziale



3) una volta effettuati i passaggi, abbiamo ottenuto una versione semplificata del circuito, dove abbiamo applicato il **TEOREMA DI DEMORGAN** in forma grafica.

COMPLETEZZA DELLE PORTE

Una volta visti tutti gli assiomi e teoremi fondanti dell'algebra di Boole, è possibile notare una sua particolare caratteristica, ossia la **COMPLETEZZA FUNZIONALE**

posseduta dalle porte **NAND** e **NOR**, poiché tutte le porte sono realizzabili tramite l'uso di solo queste 2 semplici porte e l'applicazione dei teoremi su di essi:

PORTA	Soli NAND	Soli NOR
\bar{A}	$\bar{A} \cdot \bar{A}$	$\bar{A} + \bar{A}$
$A \cdot B$	$\overline{\overline{A} \cdot \overline{B} \cdot \overline{A} \cdot \overline{B}}$	$\overline{\overline{A} + \overline{A} + \overline{B} + \overline{B}}$
$A + B$	$\overline{\overline{A} \cdot \overline{A} \cdot \overline{B} \cdot \overline{B}}$	$\overline{\overline{A} + \overline{B} + \overline{A} + \overline{B}}$
...

DALLA LOGICA AI CIRCUITI

Nell'ambito dei circuiti, esistono alcune **REGOLE DI LETTURA GENERALI** adottate come standard:

- gli **INPUT** partono da **SINISTRA** (o dall'alto);
- gli **OUTPUT** vanno verso **DESTRA** (verso il basso);
- le **PORTE** vanno da **DESTRA** a **SINISTRA**;
- si preferisce l'uso di **FILI NON CURVI** per facilitare la lettura dei collegamenti;
- i **FILI** che formano un **INCROCIO A T** sono commessi tra loro;
- i **FILI** che formano un **INCROCIO NORMALE** non sono commessi tra loro;
- i **FILI** che formano un **INCROCIO NORMALE** ed hanno un **PALLINO** al centro sono commessi tra loro.

OUTPUT MULTIPLI, X E Z

In elettronica digitale oltre agli output singoli esistono anche gli **OUTPUT MULTIPLO**.

In particolare, di seguito vedremo il funzionamento di un **CIRCUITO PRIORITARIO**, il quale, dati **N INPUT**, restituisce **N OUTPUT**, dove però solo un output assume valore 1, mentre tutti gli altri assumono valore 0. Quale tra gli N output sarà vero viene stabilito in base al **VALORE VERO PIÙ SIGNIFICATIVO** assunto dagli input.

Immaginando un circuito composto da input A_3, A_2, A_1, A_0 e come output y_3, y_2, y_1, y_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	0	1
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

PRIORITY CIRCUIT

Come vediamo nella Tabella sopra nel caso A_3 sia **VERO** allora l'output y_3 sarà **VERO**, mentre tutti gli altri saranno **FALSI**, poiché A_3 è il **VALORE VERO PIÙ SIGNIFICATIVO** in input (dunque ha la precedenza sugli altri).

Nel caso A_3 sia **FALSO**, il controllo sul valore più significativo verrà effettuato sull'input successivo nell'ordine di **PRECEDENZA**, ossia A_2 , rendendo l'output y_2 vero

(e tutti gli altri falsi) nel caso in cui A_2 sia vero. Questo processo di selezione dell'output in base alla precedenza viene effettuato su ogni input, fino a raggiungere quello con precedenza più bassa (nel nostro caso A_0).

Per via del modo in cui vengono stabiliti i valori degli output, possiamo direttamente **IGNORARE** il valore assunto da tutti gli **INPUT MENO SIGNIFICATIVI** rispetto all'input con il valore vero più significativo. In questo caso, i valori ignorati vengono definiti con una **x** e prendono il nome di **DON'T CARE** (di non interesse).

Possiamo quindi riscrivere la Tabella indicando anche quali siano i valori don't care presenti nella logica del circuito, per poi semplificare la in una versione più ristretta:

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	A ₃	A ₂	A ₁	A ₀	Y ₃	Y ₂	Y ₁	Y ₀
0	0	1	x	0	0	1	0								
0	-	x	x	0	1	0	0					0	0	0	0
0	-	x	x	0	1	0	0					0	0	0	0
0	-	x	x	0	1	0	0					0	0	0	1
0	-	x	x	0	1	0	0					0	0	1	0
1	x	x	x	1	0	0	0					0	0	1	0
1	x	x	x	1	0	0	0					0	1	0	0
1	x	x	x	1	0	0	0					1	x	x	1
1	x	x	x	1	0	0	0					0	1	0	0
1	x	x	x	1	0	0	0					1	x	x	0
1	x	x	x	1	0	0	0					0	0	0	0
1	x	x	x	1	0	0	0					0	0	0	0

=>

• OCCHIO ALLE X E ALLE Z!

Nell'ambito dei circuiti, una x non viene utilizzata solo nel caso in cui ci si trovi davanti ad un valore **DON'T CARE**, bensì viene utilizzata per indicare la presenza di un

VALORE CONTESO (o ignoto) tra uno 0 ed un 1, ossia la presenza di un errore nella progettazione di un circuito.

• **CONTESO**: il circuito tenta di generare un output sia uno 0 che un 1;

• il vero valore assunto è tra 0 e 1, ma potrebbe essere interpretato come uno 0, un 1 oppure una zona proibita;

• il risultato potrebbe **Cambiare** in base a voltaggio, temperatura, tempo e rumore magnetico;

• spesso causa un'eccessiva dissipazione di energia.



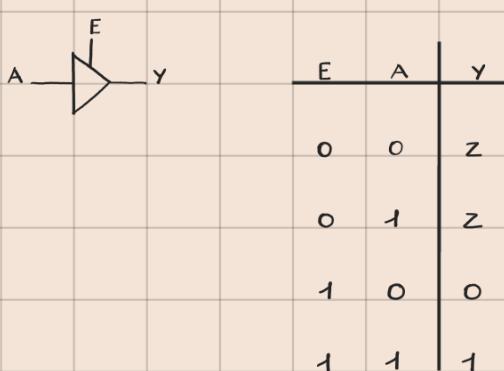
Tuttavia, esiste anche **UN'ALTRA CASISTICA** in cui un output può assumere un valore pari a 0, ad 1 o Tra 0 ed 1, ossia nel caso in cui si presenti

un **ALTA INDEPENDENZA (o floating value)** che viene indicato con **Z**.

A differenza del valore assunto da una z può essere cambiato nel caso in cui ci sia un **ALTRO SEGNALE** elettrico connesso ad essa chiamato **E**.

In questo caso si parla di un **BUFFER TRISTATE (o TRIPLO STATO)**, dove l'output può assumere valore 0, 1 o z a seconda dello **STATO DEL SECONDO SEGNALE**.

SEGNALE:



In un buffer tristate, quindi, nel caso in cui il segnale di controllo E **NON SIA ATTIVO**, allora si avrà un **VALORE Z** come output, nel caso in cui E **SIA ATTIVO**, allora l'output sarà uguale al **VALORE IN INPUT** (ossia A).

LE MAPPE DI KARNAUGH (K-MAPS)

Per facilitare l'applicazione dei 2 **TEOREMI DELL'ASSORBIMENTO** previsti dall'algebra booleana, dove, seguendo delle semplici regole, è possibile semplificare ai minimi termini quest'ultima. Questa rappresentazione grafica viene chiamata **Mappa di Karnaugh (o K-maps)**.

La struttura della K-map prevede la **TRASLACIONE** degli output di una funzione, descritti all'interno della sua Tabella della verità, in una mappatura che mette a confronto i **TERMINI SIMILI A LORO**.

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

AB		AB	
00	01	11	10
1	0	0	0
1	0	0	0

\Rightarrow c 0

AB		AB	
00	01	11	10
\overline{ABC}	\overline{ABC}	\overline{ABC}	\overline{ABC}
\overline{ABC}	\overline{ABC}	ABC	$A\overline{BC}$

\Rightarrow c 0

Normalmente potremmo riscrivere questa funzione in **FORMA SOP (o POS)** utilizzando i mintermini (o maxtermi) descritti nella sua Tabella della verità, ottenendo l'equazione $y = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C$.

Successivamente potremmo applicare il **TEOREMA DELL'ASSORBIMENTO** per semplificare l'equazione ottenuta, ottenendo l'equazione finale $y = \overline{A} \cdot \overline{B}$.

Guardando le K-map descritte sopra, possiamo notare come l'unica **DIFERENZA TRA ESSI** sia il literal definito da c. Possiamo quindi **RAGGRUPPARE** graficamente sulla mappa questi due mintermini e considerare solo la loro **PARTE IN COMUNE**, ossia l'**IMPPLICANTE** $\overline{A} \cdot \overline{B}$.

Poiché esso è l'unico implicante presente nella mappa, possiamo direttamente scrivere l'equazione minima come $y = \overline{A} \cdot \overline{B}$. In questo modo abbiamo applicato il

Teorema dell'assorbimento in modo estremamente semplificato, permettendoci di ottenere direttamente la **FORMA MINIMA POSSIBILE** con cui è possibile rappresentare l'equazione descritta nella tabella della verità.

AB		AB	
00	01	11	10
1	0	0	0
1	0	0	0

\Rightarrow

AB		AB	
00	01	11	10
\overline{ABC}	\overline{ABC}	\overline{ABC}	\overline{ABC}
\overline{ABC}	\overline{ABC}	ABC	$A\overline{BC}$

• REGOLE PER I RAGGRUPPAMENTI

• Ogni 1 deve essere raggruppato **ALMENO UNA VOLTA**, anche se solo con se stesso (implicante ad 1 variabile);

Ogni raggruppamento può estendersi **SOLO** per una quantità di quadrati pari ad una **POTENZA DI 2** (1, 2, 4, ...) in ogni direzione;

Ogni raggruppamento deve essere **IL PIÙ GRANDE POSSIBILE**, in modo da poter ridurre ai minimi termini possibili. Inoltre, il raggruppamento più grande presente in una K-map

viene chiamato **IMPONENTE PRIMO**.

I raggruppamenti possono estendersi **OLTRE I BORDI**, ad esempio in una mappa a 3 valori è possibile raggruppare le celle $(AB=0, C=0)$ e $(AB=10, C=0)$ se entrambe presentano

degli 1;

Un valore **DON'T CARE(X)** può essere raggruppato, ma solo se è utile per minimizzare l'equazione, altrimenti può essere ignorato.

Le K-map sono **DUALI**, dunque possono essere utilizzate non solo per le forme **SOP**, ma anche per le **FORME POS**: i raggruppamenti vengono effettuati tra gli **0** ed ogni quadrato

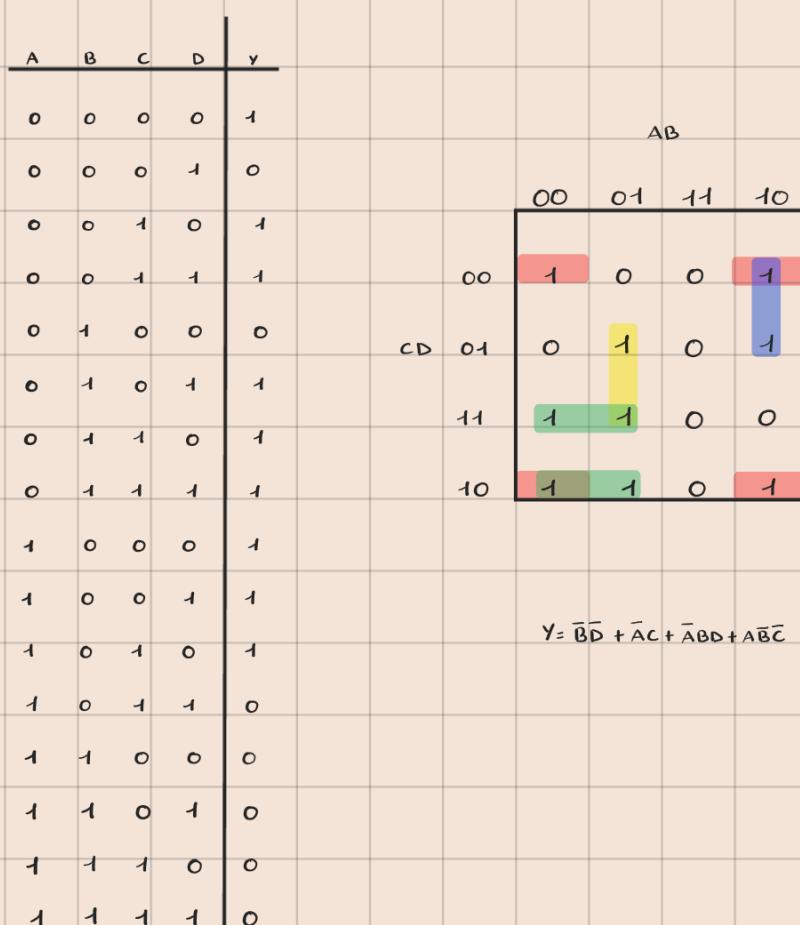
rappresenta un **MAXTERM**.

ESEMPI DI SEMPLIFICAZIONI TRAMITE LE K-MAP

Ovviamente, applicare manualmente il teorema dell'assorbimento nell'esempio precedentemente visto risulta molto banale, dunque utilizzare le K-map potrebbe essere visto

come una perdita di tempo. Tuttavia, esse risultano **ESTREMAMENTE COHODE** nel caso in cui ci si trovi dinanzi ad una funzione con molti output

possibili, come nel seguente esempio:



Per comprendere meglio i passaggi effettuati per ottenere gli implicanti che sostituiscono l'equazione finale in base ai raggruppamenti effettuati, analizziamo questi ultimi e i loro termini comuni:

RAGGRUPPAMENTO 1: ognuno dei 4 implicanti raggruppati contiene \bar{B} e \bar{D} , dunque l'implicante minimo corrispondente sarà $\bar{B}\bar{D}$:

AB

		00	01	11	10
		$\bar{A}\bar{B}$	0	0	$A\bar{B}$
$\bar{B}\bar{D}$	00	1	0	0	1
	01	0	1	0	1

CD	01	0	1	0	1	\Rightarrow tutti e 4 contengono sia \bar{B} che \bar{D}
	11	1	1	0	0	
	10	1	1	0	1	

RAGGRUPPAMENTO 2: ognuno dei 4 implicanti raggruppati \bar{A} e c , dunque l'implicante minima corrispondente sarà $\bar{A}c$:

AB						
CD	00	01	11	10		
01	0	1	0	1		
11	1	1	0	0		
10	1	1	0	1		

RAGGRUPPAMENTO 3: ognuno dei 2 implicanti raggruppati contiene \bar{A} , B e D , dunque l'implicante minima corrispondente sarà $\bar{A}BD$

AB						
CD	00	01	11	10		
01	0	1	0	1		
11	1	1	0	0		
10	1	1	0	1		

RAGGRUPPAMENTO 4: ognuno dei 2 implicanti raggruppati contiene A , \bar{B} e \bar{C} , dunque l'implicante minima corrispondente sarà $A\bar{B}\bar{C}$

AB						
CD	00	01	11	10		
01	0	1	0	1		
11	1	1	0	0		
10	1	1	0	1		

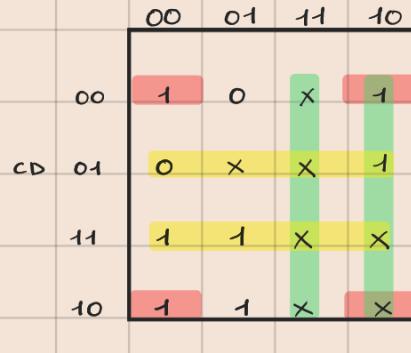
OK-MAP CON DON'T CARE

Vediamo ora una situazione simile alla precedente, dove però alcuni output sono stati sostituiti da un valore **DON'T CARE(X)**:

A	B	C	D	y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1

AB

0	1	0	0	0
0	1	0	1	x
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x



$$y = \overline{B} \overline{D} + A + C$$

Notiamo come una x in questo caso non sia stata raggruppata. Il motivo è semplice: nel caso in cui venisse raggruppata si andrebbe ad un termine eccessivo, poiché tutti gli 1 sono già stati coperti.

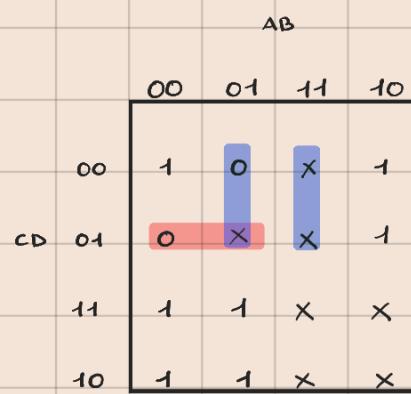
• K-MAP CON MAXTERMINI

Le K-map possono essere utilizzate non solo per semplificare un'equazione in forma minima SOP, ma anche in FORMA MINIMA POS. Le regole sono le stesse ma a punti invertite: VENGONO RAGGRUPPATI GLI 0 e gli elementi in comune vengono riscritti SOTTO FORMA DI MAXTERMINI.

Nei casi in cui vi è una grande quantità di 1 all'interno della Tavola della verità, è conveniente utilizzare questa versione in forma POS delle K-map, poiché PIÙ RAPIDA.

ESEMPIO:

A	B	C	D	y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	x
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x



$$y = (\overline{B} + C)(A + C + \overline{D})$$

1 1 1 1 X

RAGGRUPPAMENTO 1:

		AB			
		00	01	11	10
CD	00	1	0	X	1
	01	0	X	X	1
11	1	1	X	X	
10	1	1	X	X	

=> Tutti e 4 contengono sia \bar{B} che C

RAGGRUPPAMENTO 2:

		AB			
		00	01	11	10
CD	00	1	0	X	1
	01	0	X	X	1
11	1	1	X	X	
10	1	1	X	X	

=> Tutti e 2 contengono A, C e \bar{B}

!ATTENZIONE!: la forma POS e SOP sono EQUIVALENTI, quindi scegliere quali delle due "tecniche" applicare è indifferente sul risultato.

Tuttavia, notiamo che provando a convertire la forma POS ricavata in SOP, NON OTTENIAMO LA STESSA SOP ricavata precedentemente:

$$(\bar{B}+C)(A+C+\bar{B}) = \bar{B}A + \bar{B}C + \bar{B}\bar{B} + CA + C + C\bar{B} = \bar{B}A + \bar{B}C + C$$

L'incongruenza che vediamo è dovuta all'uso di alcuni DON'T CARE diversi tra i due raggruppamenti. Il funzionamento

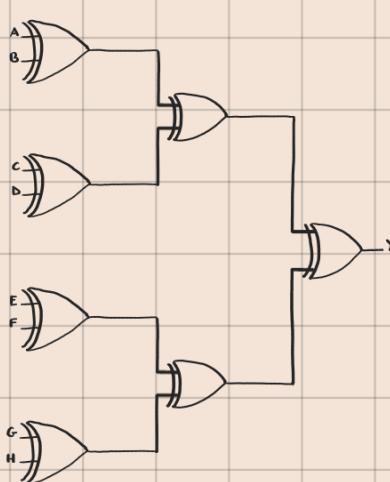
dell'equazione però è la stessa.

LOGICA COMBINATORIALE MULTI-LIVELLO

Nonostante siano molto semplici da realizzare, i circuiti con una LOGICA A DUE LIVELLI (ossia riducibili a forme SOP e POS) di grandi dimensioni richiedono molto più hardware per essere realizzati, dunque più costosi, rispetto a dei circuiti con LOGICA MULTI-LIVELLO.

Basti immaginare come per costruire uno XOR AD 8 INPUT servano 7 XOR a 2 input interconnessi tra loro, dove (ricordando che $Y = A \oplus B = \bar{A}B + A\bar{B}$) ognuno di questi ultimi è composto

a sua volta da DUE AND ED UN OR. Per realizzare un simile componente, dunque, la quantità di elementi necessari sarebbe ELEVATA e poco gestibile.



$$Y = A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H$$

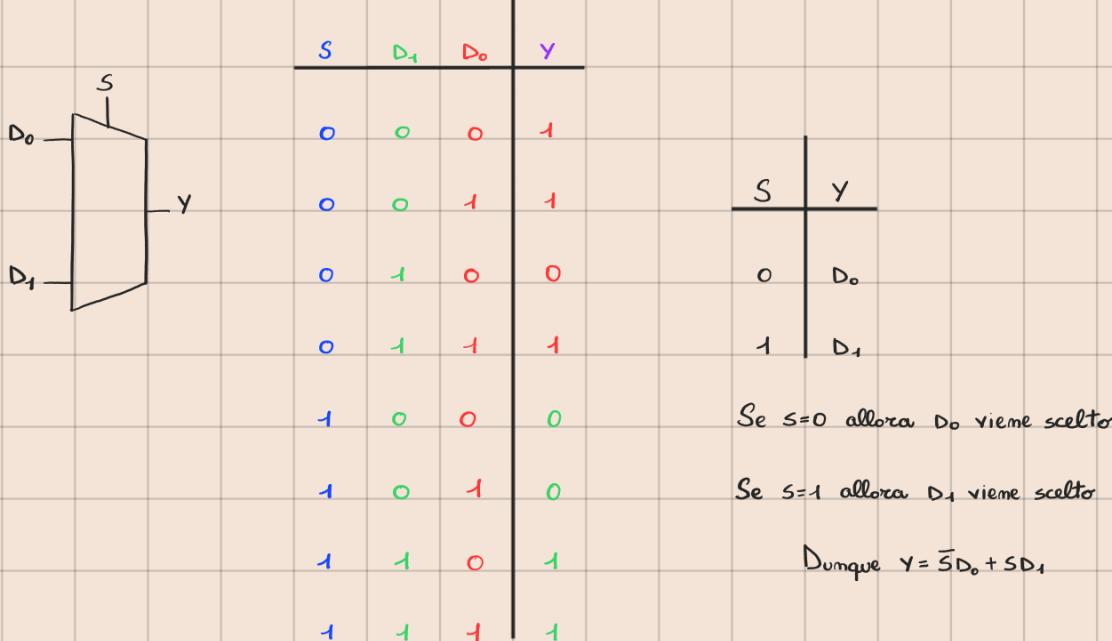
Nei circuiti con **LOGICA MULTI-LIVELLO**, i due componenti principali utilizzati che vedremo sono i **MUX** e il **DECODER**.

MUX

Il **MUX** è un componente che si occupa di **SELEZIONARE SOLO UNO DI N INPUT**, collegandolo direttamente all'output. La selezione dell'input avviene tramite

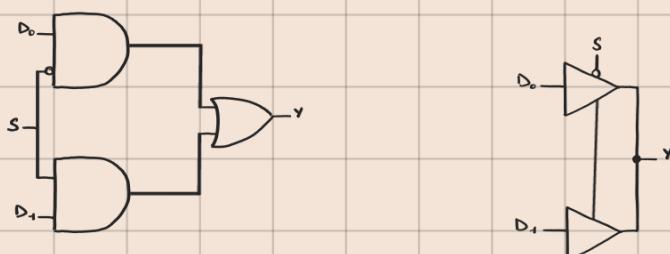
un ulteriore **INPUT DI CONTROLLO**, dove in base al suo stato viene deciso il valore assunto dall'output. Per capire meglio, vediamo il funzionamento di un **MUX 2:1**.

ESEMPIO:

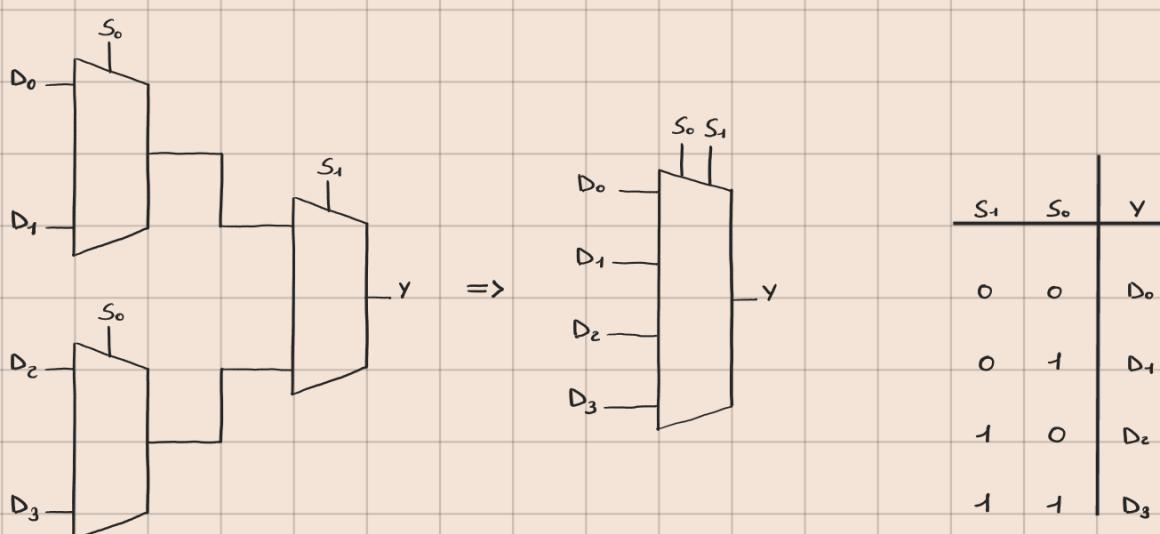


Come abbiamo visto è possibile implementare un MUX 2:1 utilizzando **3 PORTE LOGICHE**. Tuttavia, è possibile ridurre ancora il numero di componenti necessari,

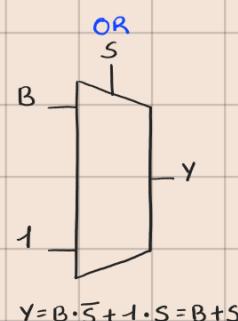
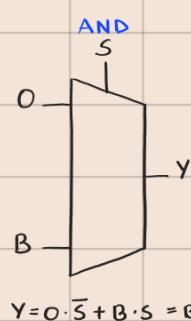
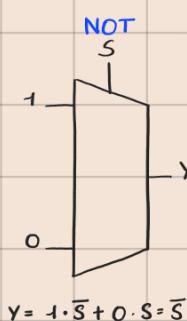
realizzando lo stesso MUX tramite **2 BUFFER TRISTATE**.



Per via della sua semplicità come componente base, utilizzando tre MUX 2:1, possiamo realizzare ad esempio un MUX 4:1, dunque con 4 input ed 1 solo output.



Un'altra importante caratteristica intrinseca al MUX, sempre poiché componente base, è la sua **COMPLETEZZA** analoga a quella delle porte NAND e NOR. Dunque è possibile realizzare ogni singola porta logica utilizzando un MUX:



• DECODER

Il **DECODER** è un componente che si occupa di **SELEZIONARE SOLO UNO TRA 2ⁿ OUTPUT (CODIFICA ONE-HOT)** generati da n valori in input. Al contrario del MUX, la selezione non avviene tramite un aggiuntivo segnale di controllo, bensì essa è intrinseca al decoder stesso. Per capire meglio, vediamo il funzionamento di un **DECODER 2:4**:

	A ₁	A ₀	Y ₃	Y ₂	Y ₁	Y ₀
A ₀	0	0	0	0	0	1
A ₁	0	1	0	0	1	0
	1	0	0	1	0	0
	1	1	1	0	0	0

$$\begin{aligned} Y_0 &= \overline{A}_1 \cdot \overline{A}_0 \\ Y_1 &= \overline{A}_1 \cdot A_0 \\ Y_2 &= A_1 \cdot \overline{A}_0 \\ Y_3 &= A_1 \cdot A_0 \end{aligned}$$

Possiamo notare come gli output del corrispondano tutti a **MINTERMINI** ottenibili con le variabili A₁ e A₀. Per questo motivo, potremmo

potenzialmente collegare una porta OR a due di questi output ed ottenere tutte le porte logiche (**COMPLETEZZA PARZIALE**).

$$\text{ESEMPIO: } A_1 \oplus A_0 = Y_1 + Y_2 = \overline{A}_1 \cdot A_0 + A_1 \cdot \overline{A}_0$$

• TEOREMA DI SHANNON

Nell'ambito dei circuiti combinatori multilivello, il **TEOREMA DI SHANNON** afferma che data una funzione booleana f di n variabili, la seguente equivalenza risulta valida:

$$f(x_1, x_2, \dots, x_n) = x_1 \cdot f(1, x_2, \dots, x_n) + \overline{x}_1 \cdot f(0, x_2, \dots, x_n)$$

Questo teorema può essere utilizzato per riscrivere una funzione di n variabili come **2 FUNZIONE DA N-1 VARIABILI**, in modo da poter implementare la funzione originale

tramite un **HUX 2:4**.

• ANALISI E SINTESI

Giunti verso la fine del capitolo sui circuiti combinatori, è opportuno precisare due **TERMINOLOGIE**, in modo che non vengano confuse tra di loro:

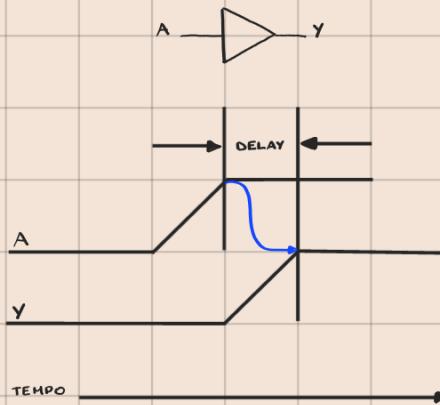
* per **ANALISI** di un circuito combinatorio si intende l'individuazione della funzione logica realizzata dal circuito, formata come espressione booleana;

• per **SINTESI** di un circuito combinatorio si intende il disegno di un circuito combinatorio a partire da un'espressione booleana (o dalla tabella di verità)

• TIMING DEI CIRCUITI

Fino ad ora abbiamo analizzato i vari circuiti combinatori solo da un punto di vista non del tutto realistico, ignorando un fattore molto importante, ossia il **TEMPO**.

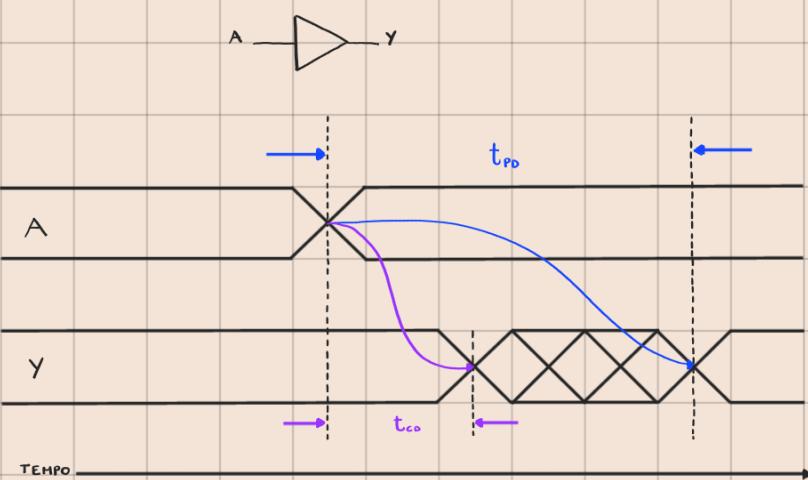
In ogni componente di un circuito, è possibile identificare un **DELAY**, ossia il tempo trascorso dal variazione dell'output. Questo delay può dipendere da effetti dell'ambiente esterno, come i limiti della velocità della luce, o dal mondo in cui è stato costruito il componente stesso, come la sua capacità e la sua resistenza.



Esistono 2 tipi di delay:

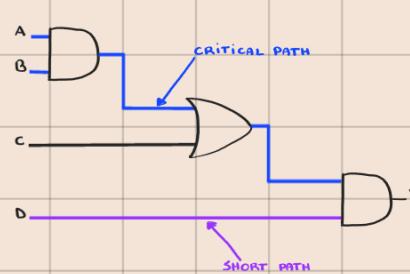
• **DELAY DI CONTAMINAZIONE (t_{co})**: corrisponde al delay **MINIMO** che deve trascorrere affinché l'output possa essere considerato come variato, dunque non sappiamo ancora se sia cambiato;

• **DELAY DI PROPAGAZIONE (t_{pd})**: corrisponde al delay **MASSIMO** che deve trascorrere affinché l'output sia considerabile variato, dunque è sicuro che sia cambiato.



Durante la progettazione dei circuiti vengono favoriti gli **SHORT PATH**, ossia i percorsi dove vanno ad accumularsi il minor numero di delay, in modo che il circuito possa essere il più veloce possibile.

ESEMPIO:



Come possiamo vedere A e B passano per due porte aggiuntive rispetto a D, risultando in **CRITICAL PATH (O LONG PATH)**:

$$\text{CRITICAL PATH} = 2 \cdot t_{PD\ AND} + t_{PB\ OR}$$

$$\text{SHORT PATH} = t_{PD\ AND}$$

