3 - Algebra di Boole

Per poter descrivere i circuiti combinando porte logiche occorre definire un nuovo tipo di algebra dove, considerato un insieme di elementi S:

- Ognuno dei suoi elementi può assumere uno di due valori, 0 e 1,
- Nella quale esiste almeno una coppia di elementi X e Y appartenenti a S ma sono diversi tra di loro ($X, Y \in S, X \neq Y$)
- Per i quali esiste una legge di composizione detta **somma logica(+)** tale che Z è uguale alla somma di X e Y($Z = X + Y$) con X,Y e Z appartenenti all'insieme S ($X, Y, Z \in S$)
- Per i quali esiste una legge di composizione detta **prodotto logico($\cdot$)** tale che Z è uguale al prodotto di X e Y($Z = X + Y$) con X,Y e Z appartenenti all'insieme S ($X, Y, Z \in S$)
- Che contenga un elemento neutro rispetto alla somma(zero) indicato dal simbolo **0**,tale che la somma di una variabile rispetto a lui restituisca la stessa variabile ($X + 0 = X$)
- Per il quale valgono le proprietà commutative delle operazioni ($+$ e $\cdot$),tali che $X + Y = Y + X$ e $X \cdot Y = Y \cdot X$
- Per le quali valgono le proprietà distributive dell'operazione $+$ rispetto alla operazione $\cdot$ e viceversa $\{X + (Y \cdot Z) = (X + Y) \cdot (X + Z)\}$,$\{ X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)\}$

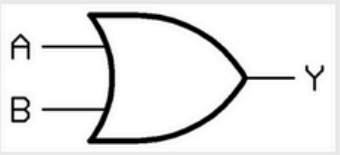
Questa proprietà definiscono la struttura che si chiama **Algebra di Boole**,un algebra che si basa su delle funzioni che hanno una o più variabili di input e generano un valore di output in base alla che dipende dal valore delle variabili.

Operatori

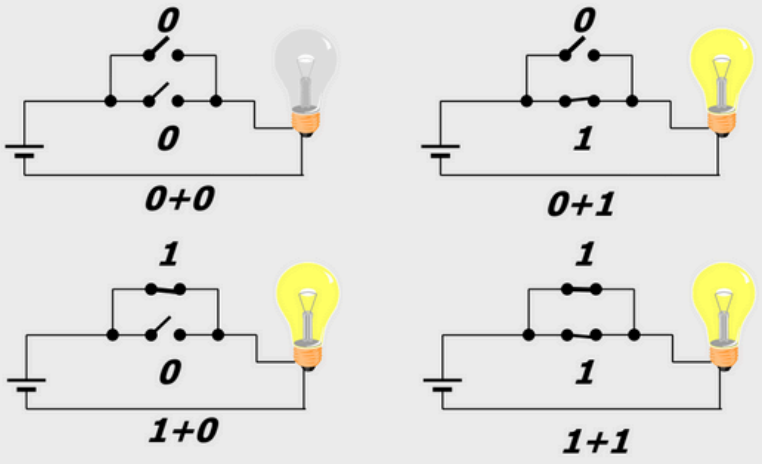
Nei circuiti ci sono 3 operatori fondamentali per effettuare operazioni sui valori booleani:

Operatore OR

L'operatore OR (o somma logica) è un operatore dove una solo condizione deve essere vera per il quale l'output sia vero ,quindi il valore della somma logica è 1 se il valore di almeno uno dei suoi operandi è 1.
Opera su almeno due variabili e le operazioni sono in parallelo



A	B	A+B A∪B A.OR.B
0	0	0
0	1	1
1	0	1
1	1	1

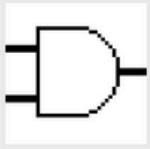


Date n variabili booleane indipendenti la loro somma logica (OR) è
$x_1 + x_2 + \ldots + x_n =$ {1 se almeno una x_i vale 1,altrimenti 0}

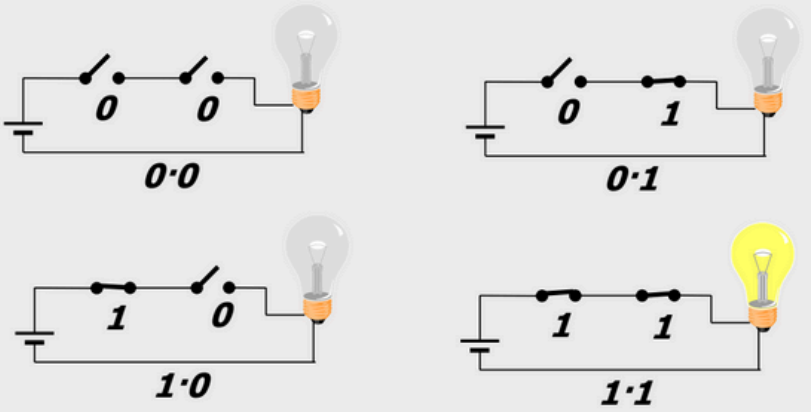
Operatore AND

L'operatore AND è un operatore dove tutte e due le condizioni devono essere vere per il quale l'output sia vero, quindi il valore della somma logica è 1 solo se il valore di tutti gli operandi è 1. In questo caso le condizioni sono in serie.

- **Prodotto logico** (AND): il valore del prodotto logico è 1 se il valore di tutti gli operandi è 1.



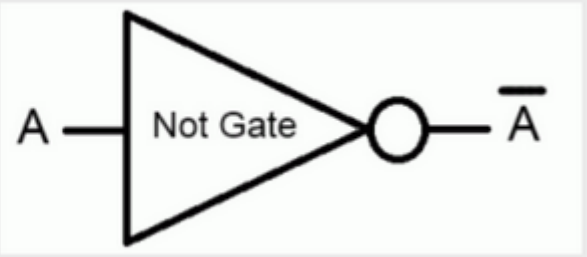
X	Y	X·Y X∩Y X.AND.Y
0	0	0
0	1	0
1	0	0
1	1	1



Operatore NOT

L'operatore NOT calcola il complemento di una condizione, nega l'ingresso (contrario della variabile inserita) ed è un operatore unario (lavora su un unico ingresso)

A	$\overline{A}$!A NOT(A)
0	1
1	0



Funzioni logiche

Una variabile y è una funzione delle n variabili indipendenti x_1,x_2,....,x_n, se esiste un criterio che fa corrispondere in modo univoco ad ognuna delle 2^n configurazioni delle x un valore di y
$y = F(x_1, x_2, x_n)$
Per rappresentare una funzione dell'algebra booleana si fa la **tabella di verità**,dove si elenca tutte le possibili combinazioni di x_1, x_n con associato il valore di y
La tabella di verità racconta quindi cosa la macchina restituisce dati dei valori a delle variabili

$F(\mathrm{a}, \mathrm{b}, \mathrm{c}) = \mathrm{a} \cdot \mathrm{b} + \mathrm{c}$

a	b	c	a·b	a·b + c
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Per costruire la tabella della verità di un espressione booleana occorre:

- Semplificare, se possibile, l'espressione mediante i teoremi dell'algebra booleana
- Calcolare i termini parziali della funzione riducendoli alle operazioni fondamentali

Proprietà

Idempotenza

Un’operazione è **idempotente** se applicarla più volte a un valore non cambia il risultato rispetto all’applicarla una sola volta
(A or $A = A$)
(A and $A = A$)

$$A + A = A$$

Infatti:

A	A	A + A
0	0	0
1	1	1

$$A \cdot A = A$$

A	A	A · A
0	0	0
1	1	1

Esistenza dell'elemento neutro per OR e AND

L’elemento neutro è un valore che, quando combinato con un altro valore tramite un’operazione logica, non cambia l’altro valore

Proprietà dell'elemento neutro per OR e AND:

$$A + 0 = A \qquad A \cdot 1 = A$$

Infatti:

A	0	A + 0
0	0	0
1	0	1

A	1	A · 1
0	1	0
1	1	1

Esistenza dell'elemento nullo per OR e AND

L'elemento nullo è un valore che, quando combinato con un altro valore tramite un'operazione logica, produce sempre l'elemento nullo stesso
Per OR è 0,per AND è 1

Proprietà dell'elemento nullo per OR e AND:

$$A + 1 = 1 \qquad A \cdot 0 = 0$$

Infatti:

A	1	A + 1
0	1	1
1	1	1

A	0	A · 0
0	0	0
1	0	0

Esistenza dell'elemento complementare per OR e AND

L’elemento complementare di un valore è il valore opposto

Proprietà dell’elemento complementare per OR e AND:

$$A + \overline{A} = 1 \qquad A \cdot \overline{A} = 0$$

Infatti:

A	$\overline{A}$	A + $\overline{A}$
0	1	1
1	0	1

A	$\overline{A}$	A · $\overline{A}$
0	1	0
1	0	0

Proprietà OR e AND

Le porte OR e AND hanno diverse proprietà:

Proprietà commutativa:

L’ordine degli operandi non influisce sul risultato dell’operazione
$A + B = B + A, A \cdot B = B \cdot A$

Proprietà associativa:

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

Assorbimento

Combinando una variabile con un’espressione che include quella variabile, il risultato è la variabile stessa
Formula:$A \cdot (A + B) = A (1) \quad A + (A \cdot B) = A (2)$

Dimostrazione (1):$A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A$
Dimostrazione (2):$A + (A \cdot B) = (A + A) \cdot (A + B) = A \cdot (A + B) = A + (1 \cdot B) = A + 1 + A$

Distributiva:

Una variabile combinata con una somma o un prodotto di altre variabili può essere distribuita su ciascuna delle variabili

Formula:

AND:$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

OR:$A + (B \cdot C) = (A + B) \cdot (A + C)$

Dimostrazione:

$[(A + B) \cdot (A + C) = A \cdot A + A \cdot C + B \cdot A + B \cdot C = A + A \cdot C + B \cdot A + B \cdot C = A(1 + C) + B \cdot A$

Legge di De Morgan

Prima legge di De Morgan: $\neg(A \cdot B) = \neg A + \neg B$

Questa legge afferma che la negazione di una congiunzione (AND) è equivalente alla disgiunzione (OR) delle negazioni

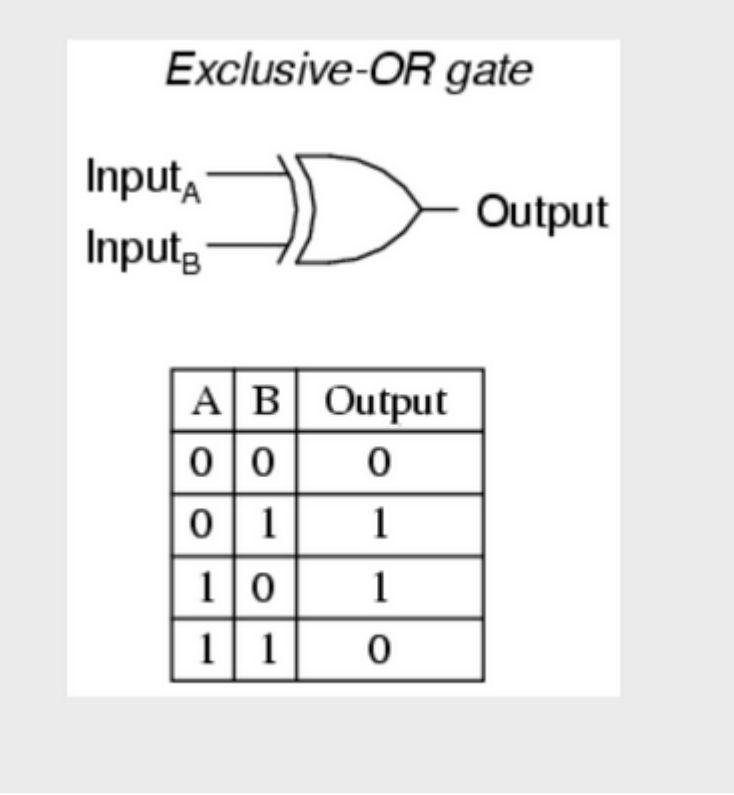
Seconda legge di De Morgan: $\neg(A + B) = \neg A \cdot \neg B$

Questa legge afferma che la negazione di una disgiunzione (OR) è equivalente alla congiunzione (AND) delle negazioni.

Porte derivate

XOR

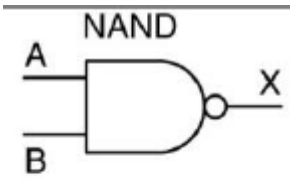
La porta XOR prende tutte le condizioni di ingresso e restituisce 1 solo e solo se gli ingressi sono diversi, serve per effettuare la verifica di disuguaglianza di due variabili



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Porta NAND

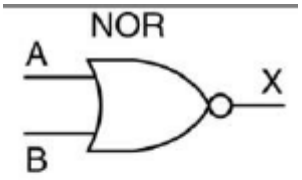
La porta NAND prende tutte le condizioni ingresso e le nega nel caso gli ingressi siano positivi, quindi restituisce 0 solo se entrambi gli ingressi sono 1,in tutti gli altri casi restituisce 1.



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

Porta NOR

La porta NOR restituisce 1 solo se entrambi gli ingressi sono 0,in tutti gli altri casi restituisce 0



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

Esercizi

Output:

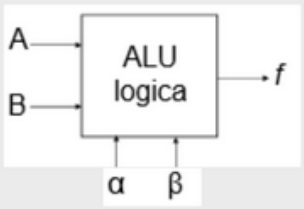
- P (passaggio):
  - 0 non passa
  - 1 passa

  Input:
- U (Uniti satelliti e nave):
  - 0 non sono uniti
  - 1 sono uniti
- I (Pressione):
  - 0 non stessa pressione

- 1 stessa pressione
- S (pile solari)
  - 0 non funzionano
  - 1 funzionano
- C (controllo da terra)
  - 0 nessun controllo
  - 1 controllo

Proposizione (operazioni): $P = (U \cdot I + \text{Not}U \cdot T) \cdot S \cdot C$

Un problema così complesso può essere risolto con una riga di codice, la parte più difficile è realizzarli
Bisogna sempre individuare variabili di ingresso e d'uscita, procedimento e tabella di verità
Avendo 1 bit si possono ottenere due configurazioni, con due entrate per operazione (una sola per il NOT, l'altro ingresso avrà la massa)
Per poter dire all'ALU quali operazione fare (NOT,XOR,AND,OR) bisogna creare una regola, dare qualche variabile come alfa e beta che se messi ad un determinato valore vengono tradotti in quell'operazione, si usa il logaritmo per poter capire quante variabili devono essere assegnate per le operazioni e le combinazioni di bit(1 bit=4 configurazioni $log4 = 2$)



$f = AB$ se $\alpha=0$ $\beta=0$
$f = A+B$ se $\alpha=0$ $\beta=1$
$f = \bar{A}$ se $\alpha=1$ $\beta=0$
$f = A \text{ xor } B$ se $\alpha=1$ $\beta=1$

$$f = \bar{\alpha}\bar{\beta}AB + \bar{\alpha}\beta\bar{A}B + \bar{\alpha}\beta A\bar{B} + \bar{\alpha}\beta AB + \\ + \alpha\bar{\beta}\bar{A}\bar{B} + \alpha\bar{\beta}\bar{A}B + \alpha\beta\bar{A}B + \alpha\beta A\bar{B}$$

Error Correcting Codes

I codici di rilevazione e/o correzione degli errori sono codici che consentono la rilevazione e/o la correzione degli errori in una parola
Questi codici sono usati per controllare una parola, un bit… per controllare se nelle trasmissioni non vi sia nulla modificato per vari motivi come il cambiamento di tensione nei chip, accoppiamento tra le piste etc.

Distanza di Hamming

Se tra due parole di codice vi è una distanza di Hamming pari a d, allora saranno necessari d errori singoli per trasformare una parole nell'altra

A	B	H(A,B)
101	111	1
1100	0011	4
100011	100101	2

La distanza di Hamming gioca un ruolo chiave nella rilevazione e correzione di errori in un codice:

- Per rilevare d errori singoli è necessario un codice con distanza di Hamming $d+1$ (infatti in questo modo non esiste alcun modo in cui d errori singoli possono cambiare una parola valida in un'altra parola valida);
- Per correggere d errori singoli è necessario un codice con distanza di Hamming $2d+1$ (infatti in questo modo anche con d cambiamenti la parola di codice originaria continua ad essere "più vicina" rispetto a tutte le altre non esiste alcun modo in cui d errori singoli possono cambiare una parola valida in un'altra parola valida);

Bit di parità

Questo algoritmo prevede un bit di parità in un messaggio. Il bit di parità viene scelto in modo tale che il numero di bit 1 nella parola codice sia pari (o dispari).

Supponiamo di voler trasmettere il byte 10011010 utilizzando un bit di parità pari:

1. Calcolo del bit di parità:
   - Contiamo il numero di bit 1 nel byte: 10011010 ha 4 bit 1,poiché il numero di bit 1 è pari, il bit di parità sarà 0 per mantenere la parità pari.
2. Trasmissione:
   - Il byte trasmesso sarà 100110100.
3. Ricezione e verifica:
   - Il ricevitore riceve 100110100 e controlla il numero di bit 1: ci sono 4 bit 1 più il bit di parità 0, quindi la parità è corretta.

Se durante la trasmissione un bit cambia, ad esempio 100110100 diventa 100110110, il ricevitore rileverà un errore perché il numero di bit 1 sarà dispari 1.

CRC

Il CRC è un altro algoritmo che prevede:

- Alla fine del messaggio il bit di controllo
- Il calcolo deve essere fatto da mittente e destinatario
- Il resto del mittente viene inviato al destinatario e deve essere 0
  Al suo interno viene utilizzato sempre lo XOR

Supponiamo di voler trasmettere il messaggio 11010011101100 utilizzando il polinomio generatore 1011.

1. **Preparazione del messaggio:**
   - Aggiungiamo tre zeri (numero di bit del polinomio generatore meno uno) al messaggio: 11010011101100000.
2. **Divisione polinomiale:**
   - Eseguiamo la divisione binaria del messaggio esteso per il polinomio generatore:

```
11010011101100000
1011
----
0111 (resto)
```

3. **Aggiunta del resto:**
   Il resto 0111 viene aggiunto al messaggio originale:
   11010011101100 `diventa` 11010011101100 0111.
4. **Trasmissione e verifica:**
   Il ricevitore esegue la stessa divisione e verifica che il resto sia `000`, confermando che non ci sono errori

Hamming

Utilizza bit di parità posizionati in modo strategico nei dati. Ogni bit di parità controlla una combinazione specifica di bit di dati. Se si verifica un errore, i bit di parità indicano la posizione esatta del bit errato, permettendo la correzione automatica

Supponiamo di voler trasmettere il messaggio `1011` utilizzando il codice di Hamming (7,4).

1. **Posizionamento dei bit di parità:**
   - I bit di parità vengono posizionati nelle posizioni 1, 2, 4: `_ _ 1 _ 0 1 1`.
   - Calcoliamo i bit di parità:
     - Parità 1 (posizione 1): copre i bit 1, 3, 5, 7 → `1 ^ 1 ^ 1 = 1`
     - Parità 2 (posizione 2): copre i bit 2, 3, 6, 7 → `0 ^ 1 ^ 1 = 0`
     - Parità 4 (posizione 4): copre i bit 4, 5, 6, 7 → `0 ^ 1 ^ 1 = 0`

2. **Messaggio codificato:**
    - Il messaggio trasmesso sarà `1010101`.
3. **Ricezione e correzione:**
    - Se il ricevitore riceve `1010001` (con un errore nel bit 5), calcola i bit di parità e trova che la posizione 5 è errata, correggendola automaticamente