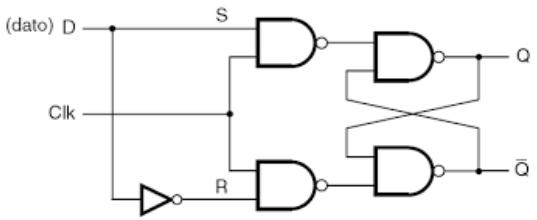


Reti Logiche Sincrone .....	3
Reti Neurali .....	4
Tipologie di Apprendimento nelle Reti Neurali.....	5
Reti Neurali Pattern recognition .....	6
Principali metodi di ottimizzazione esatta.....	7
Mappe e Algoritmo di Karnaugh .....	8
Half Adder (Mezzo Sommatore) .....	9
Multiplexer: Definizione e Funzionamento .....	10
Espressione Booleana: Definizione e Operazioni.....	11
Le Principali Porte Logiche e il loro Funzionamento .....	12
Forme Canoniche nelle Espressioni Booleane .....	13
Principali Assiomi e Teoremi dell'Algebra Booleana.....	14
Circuito RLC: Definizione e Analisi .....	15
Analisi di un Circuito RLC .....	16
Teorema di Shannon.....	17
Algoritmo di Quine-McCluskey (QMC) .....	18
Contatori Mod K Sincroni e Asincroni .....	19
Full Adder: Caratteristiche e Funzionamento .....	21
Addizionatori: Proprietà e Caratteristiche .....	22
Flip-Flop T: Proprietà e Caratteristiche .....	23
Latch SR: Proprietà e Caratteristiche .....	24
Latch SR Sincrono .....	25
Demultiplexer: Proprietà ed Esempi Applicativi.....	26
Reti Neurali Statiche .....	27
Reti Neurali Statiche – Backpropagation.....	28
ROM .....	29
Automi a Stati Finiti (Mealy – Moore).....	30
Latch JK .....	31
Latch D .....	31
Flip-Flop SR .....	32
Flip-Flop JK .....	33
Flip-Flop D .....	33
Registri e Contatori.....	34

EXTRA.....	35
Rete Combinatoria .....	36
Decoder.....	37
Priority Encoder.....	38
Sommatori Carry Look-Ahead e Carry Save.....	39
Reti Neurali Dinamiche .....	40
Reti Neurali Dinamiche – Feedforward .....	41
Reti Neurali Dinamiche – Radial Basis Function Networks (RBFN) .....	42
Benefici delle Reti Neurali .....	43
Benefici - Automazione di Compiti Complessi .....	44
Neurone e Percettrone.....	45
Neurone e Percettrone - Tabella Riassuntiva .....	46

# Reti Logiche Sincrone

Una **rete logica sincrona** è un tipo di rete digitale in cui i cambiamenti dello stato delle variabili di uscita avvengono in momenti specifici, sincronizzati con un **segnale di clock**. Questo segnale di clock, solitamente un'onda quadrata, determina il ritmo con cui le informazioni vengono processate e trasferite all'interno della rete. Ogni cambio di stato avviene solo in corrispondenza di un fronte di clock (ad esempio, quando il clock passa da basso ad alto o viceversa).



## Caratteristiche principali:

- Sincronizzazione con il clock:** Ogni operazione nella rete avviene in sincronia con il segnale di clock, che regola i momenti in cui i segnali logici vengono aggiornati.
- Comportamento deterministico:** Il comportamento della rete è prevedibile, poiché gli eventi sono determinati dal clock e non da altri fattori.
- Commutazione simultanea:** Tutti gli elementi della rete (come porte logiche e flip-flop) si aggiornano allo stesso tempo, rendendo il sistema molto preciso e controllabile.

## Esempio

Immagina una rete logica sincrona che contiene due flip-flop, i quali memorizzano un valore binario (0 o 1). Quando il segnale di clock cambia, i flip-flop possono cambiare il loro stato in base agli ingressi, ma solo al fronte di clock, garantendo che tutti i cambiamenti avvengano contemporaneamente.

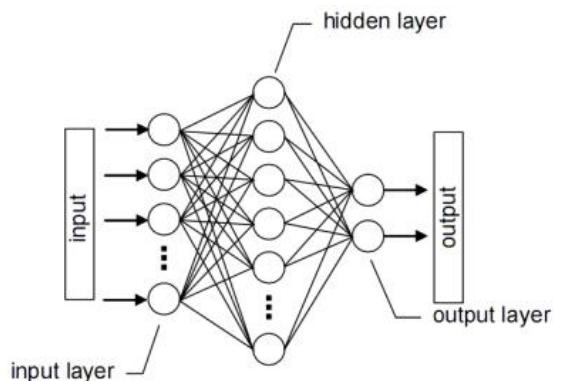
## Tabella della verità per una rete sincrona (esempio con 2 ingressi A, B e un flip-flop):

Clock	Ingresso A	Ingresso B	Uscita Q
0	0	0	0
1	1	0	1
0	1	1	1
1	0	1	0

In questa tabella, il cambiamento dello stato dell'uscita Q avviene solo quando il clock cambia, rispettando la sincronizzazione.

# Reti Neurali

Le **reti neurali** sono fondamentali nell'apprendimento moderno perché permettono di riconoscere schemi complessi, adattarsi ai dati e migliorare le proprie prestazioni attraverso l'esperienza. Sono particolarmente utili in campi come il riconoscimento delle immagini, il linguaggio naturale e la previsione dei dati, grazie alla loro capacità di apprendere da grandi quantità di informazioni senza che sia necessario programmare regole esplicite.



## Caratteristiche principali delle reti neurali

Le reti neurali imitano il funzionamento del cervello umano tramite neuroni artificiali interconnessi, organizzati in **strati** (input, nascosti e output). Tra le caratteristiche principali troviamo:

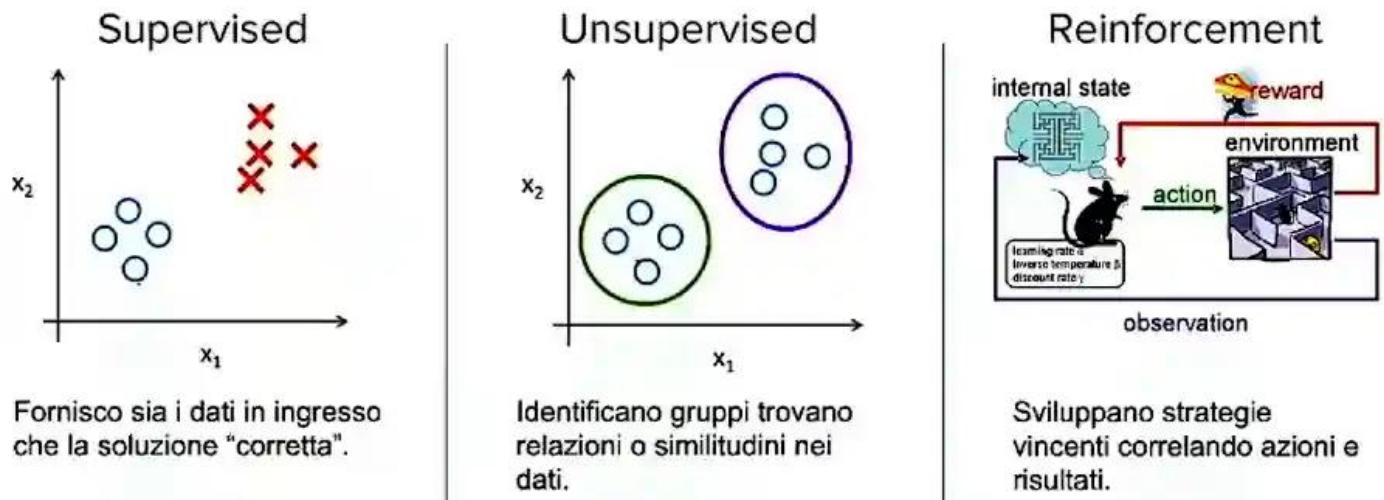
- **Apprendimento automatico:** utilizzano algoritmi come il backpropagation per ottimizzare i pesi delle connessioni.
- **Generalizzazione:** possono adattarsi a dati mai visti, riconoscendo pattern simili a quelli già appresi.
- **Non linearità:** grazie alle funzioni di attivazione (come ReLU o sigmoide), possono modellare relazioni complesse.
- **Parallelismo:** sfruttano unità di elaborazione specializzate (GPU e TPU) per gestire calcoli massivi.

## Limitazioni attuali ed esempi

- **Necessità di grandi quantità di dati:** senza un dataset adeguato, il modello può risultare inefficace.
- **Opacità del modello:** sono spesso considerate "scatole nere", rendendo difficile comprendere il perché di certe decisioni.
- **Consumo di risorse:** l'addestramento richiede una grande potenza computazionale ed energetica.
- **Difficoltà nell'estrapolazione:** se i dati di addestramento sono limitati, il modello fatica a generalizzare su nuovi scenari.

Un esempio classico di rete neurale è un sistema di riconoscimento della scrittura a mano. Il modello riceve immagini di lettere scritte a mano e, dopo l'addestramento su migliaia di esempi, è in grado di riconoscere nuove lettere mai viste prima con una buona accuratezza. Tuttavia, se il sistema viene addestrato solo con un certo stile di scrittura, potrebbe faticare a riconoscere caratteri con calligrafie molto diverse.

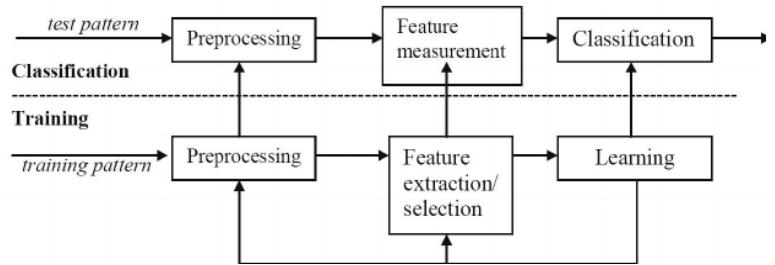
# Tipologie di Apprendimento nelle Reti Neurali



Tipo di Apprendimento	Pro	Contro
<b>Supervisionato</b>	<ul style="list-style-type: none"> <li>- Alta precisione nel riconoscimento di pattern noti.</li> <li>- Adatto per classificazione e regressione.</li> <li>- Utilizzabile in NLP, riconoscimento vocale e visione artificiale.</li> </ul>	<ul style="list-style-type: none"> <li>- Richiede un dataset etichettato, costoso e laborioso da ottenere.</li> <li>- Può soffrire di overfitting con modelli complessi.</li> </ul>
<b>Non Supervisionato</b>	<ul style="list-style-type: none"> <li>- Scopre pattern nascosti nei dati.</li> <li>- Utile per clustering e riduzione della dimensionalità.</li> <li>- Non necessita di etichette.</li> </ul>	<ul style="list-style-type: none"> <li>- Risultati difficili da interpretare.</li> <li>- Generalmente meno preciso rispetto all'apprendimento supervisionato.</li> </ul>
<b>Per Rinforzo</b>	<ul style="list-style-type: none"> <li>- Adatto per problemi complessi come robotica e giochi.</li> <li>- Può apprendere senza dataset etichettati.</li> </ul>	<ul style="list-style-type: none"> <li>- Richiede molte iterazioni per ottenere buoni risultati.</li> <li>- Convergenza lenta e incerta.</li> </ul>
<b>Semi-Supervisionato</b>	<ul style="list-style-type: none"> <li>- Usa pochi dati etichettati per migliorare l'apprendimento.</li> <li>- Meno costoso rispetto al supervisionato puro.</li> </ul>	<ul style="list-style-type: none"> <li>- Qualità dei risultati dipende dalla quantità di dati etichettati disponibili.</li> <li>- Può introdurre errori se i dati non etichettati non sono ben distribuiti.</li> </ul>

# Reti Neurali Pattern recognition

Le **reti neurali** sono strumenti potenti per il **pattern recognition** (riconoscimento di schemi), un processo fondamentale in diversi ambiti, come la visione artificiale, il riconoscimento vocale e l'elaborazione del linguaggio naturale.



Le reti neurali artificiali imitano il funzionamento del cervello umano, utilizzando **neuroni artificiali** connessi tra loro. Il riconoscimento di pattern avviene attraverso un processo di **apprendimento supervisionato o non supervisionato**, in cui la rete viene addestrata su un insieme di dati e impara a identificare le caratteristiche salienti.

Le fasi principali sono:

1. **Pre-elaborazione dei dati** – I dati grezzi vengono normalizzati e trasformati in un formato compatibile con la rete neurale.
2. **Addestramento della rete** – La rete viene esposta a esempi etichettati, utilizzando algoritmi di ottimizzazione come la **discesa del gradiente**.
3. **Apprendimento delle caratteristiche** – I neuroni nei livelli nascosti estraggono pattern dai dati attraverso pesi e funzioni di attivazione.
4. **Classificazione o riconoscimento** – L'output della rete è una probabilità associata a ciascuna classe possibile.

## Esempio: Riconoscimento di Numeri Scritti a Mano

Un classico esempio di **pattern recognition** con reti neurali è il riconoscimento delle cifre scritte a mano nel dataset **MNIST**.

1. **Dataset** – Un insieme di immagini 28x28 pixel di numeri scritti a mano (0-9).
2. **Architettura della Rete** – Una rete neurale convoluzionale (CNN) con:
  - a. **Strato di convoluzione** per estrarre caratteristiche locali (bordi, curve).
  - b. **Strato di pooling** per ridurre la dimensionalità dei dati.
  - c. **Strato completamente connesso** per la classificazione finale.
3. **Apprendimento** – La rete viene addestrata con migliaia di esempi e corregge gli errori tramite **backpropagation**.
4. **Predizione** – Dato un numero scritto a mano, la rete calcola la probabilità che appartenga a ciascuna delle 10 classi (0-9) e restituisce la più alta.

Le reti neurali sono estremamente efficaci nel riconoscimento di pattern, migliorando continuamente grazie a modelli avanzati come **Deep Learning** e **reti convoluzionali**. Questo approccio è oggi alla base di sistemi intelligenti in campi come la medicina, la sicurezza e l'intelligenza artificiale.

# Principali metodi di ottimizzazione esatta

I **metodi di ottimizzazione esatta** nelle reti logiche sono utilizzati per minimizzare le funzioni booleane e ottimizzare il design dei circuiti digitali. Questi metodi garantiscono la soluzione ottimale in termini di numero di porte logiche utilizzate, riduzione del ritardo e minimizzazione dell'area del circuito.

$cx$	$ab$	00	01	11	10
00	-	-	-	1	0
01	0	0	1	-	-
11	1	1	1	-	-
10	-	-	-	0	-

## 1. Metodo delle Tabelle di Karnaugh (K-Map)

Questo metodo grafico permette di semplificare funzioni booleane fino a sei variabili. Consiste nel raggruppare termini adiacenti per ridurre il numero di porte logiche necessarie. È efficace per funzioni con poche variabili, ma diventa poco pratico per problemi più grandi.

## 2. Metodo di Quine-McCluskey

Un approccio tabellare che estende il metodo delle mappe di Karnaugh a un numero maggiore di variabili. Identifica i primi implicanti essenziali e minimizza le espressioni booleane in modo sistematico, risultando adatto anche a problemi più complessi rispetto a K-Map.

## 3. Risoluzione tramite Programmazione a Vincoli

In alcuni casi, l'ottimizzazione delle reti logiche può essere formulata come un problema di soddisfazione di vincoli, dove si cercano configurazioni ottimali delle porte logiche rispettando regole predefinite.

## Confronto tra i metodi

Metodo	Vantaggi	Svantaggi
Mappe di Karnaugh	Facile da applicare con poche variabili	Difficile da gestire oltre 6 variabili
Quine-McCluskey	Metodo sistematico e applicabile a più variabili	Computazionalmente pesante per grandi circuiti
Programmazione a Vincoli	Adatta a problemi complessi con molte regole	Richiede strumenti avanzati per la risoluzione

L'ottimizzazione esatta nelle reti logiche è fondamentale per migliorare l'efficienza dei circuiti digitali, riducendo costi e consumi energetici.

# Mappe e Algoritmo di Karnaugh

L'**algoritmo di Karnaugh** è una tecnica grafica utilizzata per la **semplificazione delle funzioni booleane**. Si basa sulla **Mappa di Karnaugh (K-map)**, una rappresentazione tabellare che consente di individuare velocemente **termini comuni** e di ridurre il numero di operazioni logiche necessarie in un circuito digitale. Questo metodo è particolarmente utile per minimizzare funzioni di **2, 3, 4 e 5 variabili**, facilitando la progettazione di **circuiti logici più efficienti**.

Le K-Map rappresentano una tabella bidimensionale in cui ogni cella corrisponde a un mintermine della funzione booleana. Le variabili di ingresso vengono organizzate in modo tale che ogni cella differisca da quelle adiacenti per un solo bit (proprietà del codice di Gray).

## Struttura della Mappa di Karnaugh

Le mappe possono essere costruite per funzioni con **2, 3, 4 o più variabili**, organizzando i mintermini in una griglia:

- **2 variabili** →  $2 \times 2$
- **3 variabili** →  $2 \times 4$
- **4 variabili** →  $4 \times 4$
- **5+ variabili** → estensione tridimensionale o scomposizione in più K-Map

## Procedura di Semplificazione con le K-Map

1. **Compilare la tabella:** si inseriscono i valori della funzione booleana (0 o 1) nelle celle corrispondenti ai mintermini.
2. **Raggruppare gli 1 adiacenti:** si formano gruppi di **1, 2, 4, 8, ... mintermini** in modo da coprire il massimo numero di valori con il minimo numero di gruppi.
3. **Estrarre la funzione semplificata:** per ogni gruppo si individuano le variabili che rimangono costanti, eliminando quelle che cambiano.

## Esempio di Semplificazione

Data la funzione booleana:

$$F(A,B,C) = \sum_{m(1,2,3,5,6,7)}$$

Inserendo i valori nella K-Map e raggruppando gli 1, otteniamo la funzione semplificata:

$$F(A,B,C) = B'C + AC$$

Le mappe di Karnaugh sono particolarmente utili per semplificare circuiti con poche variabili, ma diventano poco pratiche per più di 5-6 variabili, dove si preferiscono metodi algoritmici come **Quine-McCluskey**.

# Half Adder (Mezzo Sommatore)

L'Half Adder, o mezzo sommatore, è un circuito combinatorio utilizzato per eseguire l'addizione di due bit. È una delle unità fondamentali per la realizzazione di sommatori più complesse, come il Full Adder e gli ALU nei processori.

## Funzionamento

Il mezzo sommatore prende in ingresso due bit, **A** e **B**, e produce due uscite:

1. **Somma (S)**: rappresenta il bit meno significativo della somma.
2. **Riporto (C)**: indica se c'è un riporto al bit successivo.

Le equazioni booleane che descrivono il suo funzionamento sono:

- **Somma:**  $S = A \oplus B$  (XOR)
- **Riporto:**  $C = A \cdot B$  (AND)

## Schema Circuitale

Un **Half Adder** può essere implementato con **una porta XOR e una porta AND**:

- La **porta XOR** genera l'uscita **S**.
- La **porta AND** genera l'uscita **C** (carry).

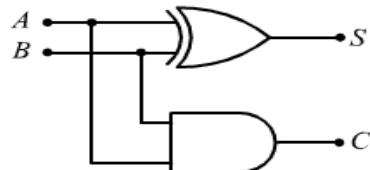
## Proprietà

- **Semplice da implementare** con una porta XOR e una porta AND
- **Non gestisce il riporto in ingresso**, quindi può sommare solo due bit senza portare il risultato a un'altra cifra.

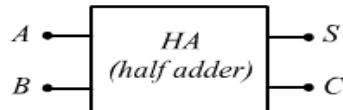
## Limitazioni del Half Adder

L'Half Adder **non può gestire il riporto da un'operazione precedente**, rendendolo inadatto per l'addizione di numeri binari con più bit. Per superare questa limitazione si utilizza il **Full Adder**, che include un ingresso per il riporto precedente.

Tuttavia, il **mezzo sommatore** è utile per operazioni semplici e per costruire sommatori più grandi, come nelle **reti aritmetiche** dei microprocessori.



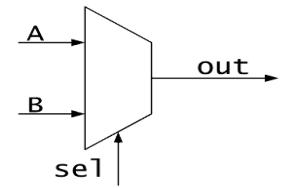
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = A \oplus B$$
$$C = A \cdot B$$

# Multiplexer: Definizione e Funzionamento

Un **Multiplexer (MUX)** è un circuito combinatorio che consente di **selezionare uno tra più ingressi** e inviarlo a un'unica uscita. Funziona come un **commutatore digitale**, controllato da **linee di selezione**, che determinano quale ingresso deve essere trasferito in uscita.



L'operazione del multiplexer è espressa dalla formula:

$$Y = X_i \text{ (dove } i \text{ è determinato dai bit di selezione)}$$

## Struttura del Multiplexer

Un **MUX 2:1** (2 ingressi, 1 uscita) ha:

- **2 ingressi dati** ( $X_0, X_1$ )
- **1 linea di selezione** ( $S$ )
- **1 uscita** ( $Y$ )

### S Uscita Y

S	Uscita Y
0	$X_0$
1	$X_1$

$$Y = (X_0 \cdot \bar{S}) + (X_1 \cdot S)$$

Il concetto si estende a **MUX 4:1, 8:1, 16:1**, con **n linee di selezione** per  $2^n$  **ingressi**.

## Esempi Applicativi

1. **Comunicazioni digitali**: un MUX permette di inviare più segnali su una **singola linea** e decodificarli a destinazione con un **Demultiplexer (DEMUX)**.
2. **Processori e Memorie**: utilizzato per selezionare **registri o blocchi di memoria** da leggere o scrivere.
3. **Sistemi di Controllo**: impiegato nei **sistemi embedded** per scegliere sensori diversi senza necessità di più canali fisici.
4. **Display e Multiplexing temporale**: usato per attivare **segmenti di display a LED** in sequenza, riducendo il numero di connessioni.

Il **Multiplexer** è un componente fondamentale nelle reti logiche, migliorando l'efficienza e riducendo la complessità dei circuiti digitali.

# Espressione Booleana: Definizione e Operazioni

Un'espressione booleana è una combinazione di variabili e operatori logici che assume come risultato solo due possibili valori: **0 (Falso)** o **1 (Vero)**. Le operazioni che compongono un'espressione booleana seguono le regole dell'algebra di Boole, che è alla base del funzionamento delle reti logiche digitali.

A	B	$A \wedge \bar{B}$
0	0	0
0	1	0
1	0	1
1	1	0

Le principali operazioni booleane sono:

- **NOT (Negazione)**: Inverte il valore di una variabile. Se  $A = 1$ , allora  $\bar{A} = 0$ .
- **AND (Moltiplicazione logica)**: Restituisce **1** solo se entrambe le variabili sono **1**.
  - **Esempio:**  $A \cdot B$
- **OR (Somma logica)**: Restituisce **1** se almeno una delle variabili è **1**.
  - **Esempio:**  $A + B$
- **XOR (OR esclusivo)**: Restituisce **1** se le due variabili sono diverse.
  - **Esempio:**  $A \oplus B$

## Differenza tra Espressione e Funzione Booleana

Un'espressione booleana è una scrittura algebrica che descrive una combinazione di operazioni logiche tra variabili. Una funzione booleana, invece, è una relazione che associa a ogni possibile combinazione di variabili un valore di uscita.

In pratica:

- **Espressione booleana** → rappresentazione algebrica di una funzione logica.
- **Funzione booleana** → insieme di coppie ingresso-uscita, spesso rappresentato da una tabella della verità.

### Esempio:

L'espressione  $F(A, B) = A + B$  descrive la funzione booleana che restituisce **1** se almeno una delle variabili è **1**. La tabella della verità mostra come il valore di uscita dipende dalle combinazioni di **A** e **B**.

Le espressioni booleane possono essere semplificate con metodi come le **Mappe di Karnaugh** o l'**algoritmo di Quine-McCluskey**, riducendo la complessità dei circuiti logici.

# Le Principali Porte Logiche e il loro Funzionamento

Le **porte logiche** sono i componenti fondamentali dei circuiti digitali. Esse elaborano segnali binari (0 e 1) secondo le regole dell'algebra di Boole e consentono la realizzazione di operazioni logiche complesse. Combinandole, si possono costruire **sommatori, decoder, multiplexers e memorie**, che sono la base dell'elettronica moderna.

Le principali porte logiche sono:

**NOT, AND, OR, XOR, NAND, e XNOR.**

**NOT (Negazione)**

Porta logica NAND $Y = \overline{A \cdot B}$			Porta logica AND $Y = A \cdot B$			Porta logica OR $Y = A + B$		
A	B	Y	A	B	Y	A	B	Y
0	0	1	0	0	0	0	0	1
0	1	1	0	1	0	1	0	1
1	0	1	1	0	0	1	0	1
1	1	0	1	1	1	1	1	1
Porta logica XOR $Y = A \oplus B$			Porta logica NOR $Y = \overline{A + B}$			Porta logica NOT $Y = \overline{A}$		
A	B	Y	A	B	Y	A	Y	Y
0	0	0	0	0	1	0	1	0
0	1	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	1
1	1	0	1	1	1	0	0	1

La porta **NOT** inverte il valore di ingresso. Se l'ingresso è **1**, l'uscita sarà **0**, e viceversa.

**AND (Moltiplicazione logica)**

La porta **AND** restituisce **1** solo se entrambi gli ingressi sono **1**.

**OR (Somma logica)**

La porta **OR** restituisce **1** se almeno uno degli ingressi è **1**.

**XOR (OR esclusivo)**

La porta **XOR** restituisce **1** se gli ingressi sono diversi.

**NAND (NOT AND)**

La porta **NAND** è l'inverso della porta AND: restituisce **0** solo quando entrambi gli ingressi sono **1**.

**NOR (NOT OR)**

La porta **NOR** è l'inverso della porta OR: restituisce **1** solo se entrambi gli ingressi sono **0**.

**XNOR (XOR Negato)**

La porta **XNOR** è l'inverso della XOR: restituisce **1** se gli ingressi sono uguali.

# Forme Canoniche nelle Espressioni Booleane

Le **forme canoniche** sono rappresentazioni standardizzate delle espressioni booleane che consentono di descrivere in modo univoco una funzione logica, sono essenziali per la progettazione di circuiti logici e la semplificazione delle funzioni logiche, consentendo di rappresentare in modo sistematico una funzione e di ottimizzarne la realizzazione hardware. Le due principali forme canoniche sono:

- **Forma Normale Disgiuntiva (Sum of Products - SOP)**
- **Forma Normale Congiuntiva (Product of Sums - POS)**

## Forma Normale Disgiuntiva (SOP - Sum of Products)

La **SOP** è una somma (**OR**) di termini prodotti (**AND**) delle variabili in forma diretta o negata. Ogni termine rappresenta un **mintermine**, ovvero una combinazione unica delle variabili che rende l'uscita uguale a **1**.

$$F(A,B,C) = A'B'C + AB'C + ABC$$

Questa è una **forma SOP**, perché ogni termine è un prodotto (**AND**) variabili e l'intera espressione è una somma di questi termini.

I **mintermini** corrispondono alle righe in cui  $F(A,B,C)=1$ .

## Forma Normale Congiuntiva (POS - Product of Sums)

A	B	C	<u>F(A, B, C)</u>
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	0
1	1	1	1

La **POS** è un prodotto (**AND**) di termini somma (**OR**) delle variabili in forma diretta o negata. Ogni termine rappresenta un **maxtermine**, ovvero una combinazione che rende l'uscita uguale a **0**.

$$F(A,B) = (A+B)(A'+B)$$

Questa è una **forma POS**, poiché è una moltiplicazione di somme di variabili.

**maxtermini** corrispondono alle righe in cui  $F(A,B)=0$

A	B	<u>F(A, B)</u>
0	0	0
0	1	1
1	0	1
1	1	1

## Proprietà delle Forme Canoniche

- **Univocità:** Ogni funzione booleana può essere espressa in un'unica forma SOP o POS.
- **Convertibilità:** Una funzione in forma SOP può essere trasformata in POS e viceversa tramite la **dualità** dell'algebra booleana.
- **Semplificazione:** Le forme canoniche possono essere semplificate con **Mappe di Karnaugh** o l'**algoritmo di Quine-McCluskey**.

# Principali Assiomi e Teoremi dell'Algebra Booleana

Gli **assiomi e teoremi dell'algebra booleana** sono fondamentali per semplificare espressioni logiche e ottimizzare circuiti digitali. L'uso delle **leggi di De Morgan**, delle **proprietà distributive e commutative**, e dei **teoremi di assorbimento** consente di progettare sistemi più efficienti e con un numero ridotto di componenti logici.

## Assiomi Fondamentali

Gli assiomi sono regole di base che definiscono le operazioni principali dell'algebra booleana (Sono Assiomi fino all'inverso nella tabella sotto)

	<b>AND</b>	<b>OR</b>
<b>Identità</b>	$1 \cdot x = x$	$0 + x = x$
<b>Elemento nullo</b>	$0 \cdot x = 0$	$1 + x = 1$
<b>Idempotenza</b>	$x \cdot x = x$	$x + x = x$
<b>Inverso</b>	$x \cdot \bar{x} = 0$	$x + \bar{x} = 1$
<b>Commutativa</b>	$x \cdot y = y \cdot x$	$x + y = y + x$
<b>Associativa</b>	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$(x + y) + z = x + (y + z)$
<b>Assorbimento</b>	$x \cdot (x + y) = x$	$x + x \cdot y = x$
<b>Distributiva</b>	<i>AND rispetto a OR</i> $x \cdot (y + z) = x \cdot y + x \cdot z$	<i>OR rispetto a AND</i> $x + y \cdot z = (x + y) \cdot (x + z)$
<b>De Morgan</b>	$\overline{x \cdot y} = \bar{x} + \bar{y}$	$\overline{x + y} = \bar{x} \cdot \bar{y}$

# Circuito RLC: Definizione e Analisi

Un **circuito RLC** è un circuito elettrico che contiene una **resistenza (R)**, un **induttore (L)** e un **condensatore (C)**. Questi componenti lavorano insieme per determinare il comportamento della corrente e della tensione nel circuito, in particolare per quanto riguarda fenomeni come l'**oscillazione, la risonanza e lo smorzamento**. È essenziale nell'elettronica analogica e nelle telecomunicazioni. Il suo comportamento dipende dall'interazione tra resistenza, induttanza e capacità, con effetti importanti come l'oscillazione e la risonanza. Grazie alle sue caratteristiche, viene ampiamente utilizzato in filtri, trasmettitori radio e dispositivi di controllo della corrente e della tensione.

## Tipologie di Circuiti RLC

Esistono due principali configurazioni di un circuito RLC:

- **Circuito RLC in serie:** La resistenza, l'induttanza e la capacità sono collegati uno dopo l'altro in sequenza. Questo tipo di circuito è utilizzato nei filtri elettronici e nei circuiti di risonanza.
- **Circuito RLC in parallelo:** Ogni componente è collegato in parallelo agli altri. Questo tipo di circuito è comune nei sistemi di filtraggio dei segnali.

Il comportamento di questi circuiti dipende dalle proprietà di ciascun componente:

- **La resistenza (R)** dissipa energia sotto forma di calore.
- **L'induttore (L)** immagazzina energia in un campo magnetico e si oppone alle variazioni di corrente.
- **Il condensatore (C)** immagazzina energia in un campo elettrico e si oppone alle variazioni di tensione.

## Risposta del Circuito

Un circuito RLC può **oscillare**, smorzandosi nel tempo a seconda del valore della resistenza:

- **Oscillazione sottosmorzata:** La corrente oscilla prima di stabilizzarsi.
- **Smorzamento critico:** Il circuito raggiunge lo stato stabile nel minor tempo possibile senza oscillazioni.
- **Oscillazione sovrasmorzata:** Il circuito ritorna allo stato stabile molto lentamente.

Un aspetto fondamentale è la **risonanza**, che si verifica quando il circuito permette il massimo passaggio di corrente a una certa frequenza. Questa caratteristica viene sfruttata in molte applicazioni elettroniche.

# Analisi di un Circuito RLC

L'**analisi di un circuito RLC** consiste nello studio del comportamento della corrente e della tensione nel tempo, in base ai valori di **resistenza (R)**, **induttanza (L)** e **capacitanza (C)**. Questo processo permette di determinare come il circuito reagisce a una tensione applicata e come si comporta in diverse condizioni operative. E' un processo fondamentale per prevedere il comportamento del circuito in diverse condizioni. Permette di ottimizzare il design elettronico, migliorare l'efficienza energetica e garantire il corretto funzionamento di dispositivi che operano con segnali variabili nel tempo.

## Obiettivi dell'Analisi

- **Determinare il comportamento della corrente:** come varia nel tempo e se presenta oscillazioni.
- **Identificare la frequenza di risonanza:** il punto in cui il circuito risponde con la massima efficienza.
- **Studiare lo smorzamento:** valutare se il circuito perde energia rapidamente o continua a oscillare.
- **Prevedere la risposta a segnali esterni:** come il circuito reagisce a diverse tensioni in ingresso.

## Tipologie di Analisi

L'analisi può essere fatta in diversi modi, a seconda del tipo di circuito e delle informazioni che si vogliono ottenere.

- **Analisi nel dominio del tempo:** studia come variano corrente e tensione nel tempo, utile per capire il comportamento del circuito dopo un impulso o una variazione improvvisa.
- **Analisi nel dominio della frequenza:** si concentra su come il circuito risponde a segnali di diverse frequenze, determinando la frequenza di risonanza.
- **Analisi in regime stazionario:** osserva il comportamento del circuito quando ha raggiunto una condizione di equilibrio, senza più variazioni transitorie.

## Risultati Tipici dell'Analisi

A seconda della configurazione del circuito e dei valori di **R, L e C**, si possono ottenere diversi comportamenti:

- **Oscillazioni smorzate**, se la resistenza è bassa e il circuito continua a oscillare per un po' prima di stabilizzarsi.
- **Risonanza**, quando il circuito raggiunge la massima corrente a una determinata frequenza.
- **Risposta lenta**, se la resistenza è alta e il circuito impiega molto tempo a tornare allo stato stabile.

# Teorema di Shannon

Il **teorema di Shannon**, noto anche come **teorema della decomposizione di Shannon**, è un principio fondamentale dell'**algebra booleana** che permette di semplificare e progettare circuiti logici in modo più efficiente. Il teorema afferma che una **funzione booleana** può essere espressa in termini di due funzioni più semplici, riducendo così la complessità della sua implementazione in hardware. Il teorema di Shannon è particolarmente utile per:

- **Semplificare circuiti logici**, riducendo il numero di porte necessarie.
- **Progettare circuiti sequenziali**, separando il comportamento del circuito in base a una variabile chiave.
- **Progettazione di multiplexer**: permette di strutturare un circuito come una selezione tra due funzioni parziali.
- Permette di trasformare un'espressione booleana arbitraria in una **struttura ad albero**, utile nella realizzazione dei circuiti digitali.

## 1. Definizione del Teorema di Shannon

Il teorema di Shannon afferma che una funzione booleana  $F(X_1, X_2, \dots, X_n)$  può essere decomposta rispetto a una variabile  $X_i$  nel modo seguente:

$$F(X_1, X_2, \dots, X_n) = X_i \cdot F(1, X_2, \dots, X_n) + \overline{X_i} \cdot F(0, X_2, \dots, X_n)$$

Dove:

- $F(1, X_2, \dots, X_n)$  è la funzione valutata ponendo  $X_i = 1$ .
- $F(0, X_2, \dots, X_n)$  è la funzione valutata ponendo  $X_i = 0$ .
- $X_i$  e  $\overline{X_i}$  sono rispettivamente la variabile e il suo complemento.

Questa decomposizione permette di scomporre una funzione complessa in termini più semplici, facilitandone l'implementazione mediante circuiti logici.

## 2. Dimostrazione del Teorema di Shannon

Per dimostrare il teorema, consideriamo una generica funzione booleana  $F(X_1, X_2, \dots, X_n)$ .

Possiamo scrivere:

$$F = (X_i + \overline{X_i}) \cdot F$$

Poiché per le leggi dell'algebra booleana si ha sempre  $X_i + \overline{X_i} = 1$ , questa espressione è logicamente equivalente a  $F$ .

Ora, distribuendo  $F$  sui due termini:

$$F = X_i \cdot F + \overline{X_i} \cdot F$$

A questo punto, sostituiamo i valori di  $F$  nei due casi specifici:

- Se  $X_i = 1$ , allora  $F(1, X_2, \dots, X_n)$ .
- Se  $X_i = 0$ , allora  $F(0, X_2, \dots, X_n)$ .

Sostituendo, otteniamo:

$$F = X_i \cdot F(1, X_2, \dots, X_n) + \overline{X_i} \cdot F(0, X_2, \dots, X_n)$$

Dimostrando così la validità del teorema.

# Algoritmo di Quine-McCluskey (QMC)

L'**algoritmo di Quine-McCluskey (QMC)** è un metodo sistematico per la **minimizzazione delle funzioni booleane**. È una tecnica che, a differenza della **Mappa di Karnaugh**, può essere applicata efficacemente anche a funzioni con un numero elevato di variabili (più di 5 o 6), rendendolo ideale per la progettazione di circuiti logici complessi.

L'algoritmo di QMC è un **metodo tabellare** che utilizza il **concetto di implicanti primi**, ossia i termini minimi da cui è composta la funzione booleana semplificata. L'obiettivo è trovare la **copertura minima degli implicanti primi**, riducendo così il numero di operazioni logiche necessarie.

Si basa su due fasi principali:

1. **Generazione degli implicanti primi** mediante un confronto sistematico tra mintermini.
2. **Selezione della copertura minima** degli implicanti primi per ottenere la forma semplificata della funzione.

## Passaggi dell'Algoritmo di Quine-McCluskey

### Passo 1: Tabella dei Mintermini

- Si elencano tutti i **mintermini** della funzione (valori per cui la funzione vale 1).
- Si ordinano in base al numero di bit a 1.

### Passo 2: Confronto tra Mintermini e Generazione degli Implicanti

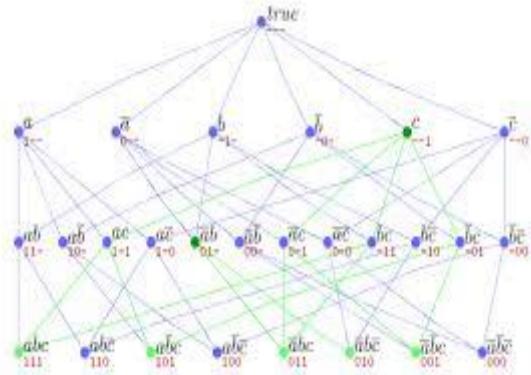
- Si confrontano i mintermini che differiscono **di un solo bit**, sostituendo la posizione variabile con un "-".
- Il processo viene ripetuto fino a quando non è più possibile combinare ulteriori termini.
- I termini che non possono più essere combinati diventano **implicanti primi**.

### Passo 3: Costruzione della Tabella degli Implicanti Primi

- Si crea una tabella che collega ogni mintermine agli implicanti primi che lo coprono.
- Si identificano gli **implicanti primi essenziali**, ovvero quelli che coprono almeno un mintermine in modo unico.

### Passo 4: Selezione della Copertura Minima

- Si sceglie il numero minimo di implicanti primi necessario per coprire tutti i mintermini



# Contatori Mod K Sincroni e Asincroni

I **contatori mod K** sono circuiti sequenziali utilizzati per **contare eventi digitali** e si dividono in **sincroni** e **asincroni**, a seconda di come i flip-flop interni ricevono il segnale di clock.

Un **contatore mod K** ha **K stati distinti** prima di tornare allo stato iniziale. Per implementarlo, il numero di flip-flop richiesto è  $n \geq \log_2(K)$ , dove  $n$  è il numero di bit necessari.

I contatori mod K sono elementi fondamentali della **logica sequenziale**, utilizzati in una vasta gamma di dispositivi elettronici. I **contatori asincroni** sono più semplici, ma soffrono di ritardi di propagazione, mentre i **contatori sincroni** sono più complessi ma offrono prestazioni migliori. La scelta tra i due dipende dalle esigenze specifiche del sistema in cui devono essere implementati.

## Contatori Asincroni (Ripple Counter)

Nei **contatori asincroni**, il segnale di clock viene applicato **solo al primo flip-flop**, mentre i successivi vengono attivati dal cambiamento di stato del flip-flop precedente. Questo causa un **ritardo di propagazione** tra i diversi flip-flop, rendendoli più **lenti** rispetto ai contatori sincroni.

### Struttura

- Utilizza **flip-flop T o JK** in cascata.
- Il primo flip-flop è sincronizzato con il clock, mentre i successivi cambiano stato in base all'uscita del flip-flop precedente.
- Si dice "**ripple**" (**a cascata**) perché l'aggiornamento degli stati si propaga con un ritardo accumulato.

#### Vantaggi:

- **Facile da realizzare** con pochi componenti.
- Richiede **meno risorse hardware** rispetto ai contatori sincroni.

#### Svantaggi:

- **Ritardo di propagazione** accumulato tra i flip-flop.
- **Non adatto per frequenze elevate**, perché gli errori aumentano con la velocità del clock.

## Contatori Sincroni

Nei **contatori sincroni**, tutti i flip-flop ricevono il segnale di clock simultaneamente, eliminando i ritardi di propagazione tipici dei contatori asincroni. Questo li rende più **veloci e affidabili**, specialmente per contatori ad alta frequenza.

### Struttura

- Tutti i flip-flop sono sincronizzati con il clock principale.
- La logica combinatoria tra i flip-flop determina quando devono cambiare stato.
- Utilizza **gate logici** per controllare l'attivazione dei flip-flop in base allo stato corrente.

#### Vantaggi:

- **Maggior velocità** grazie all'assenza di ritardo di propagazione.
- **Affidabile per alte frequenze di clock**.

#### Svantaggi:

- Richiede **più componenti hardware**, aumentando la complessità del circuito.

## Confronto tra Contatori Sincroni e Asincroni

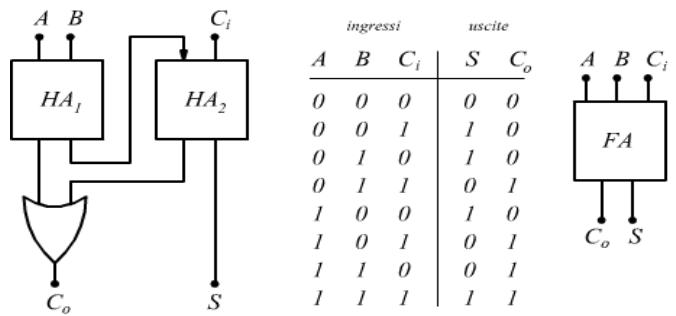
Caratteristica	Contatore Asincrono	Contatore Sincrono
Clock	Solo al primo flip-flop	Sincronizzato su tutti i flip-flop
Ritardo	Accumulato, aumenta con il numero di bit	Minimo, tutti i flip-flop cambiano contemporaneamente
Velocità	Lento per alte frequenze	Più veloce e affidabile
Hardware richiesto	Minimo, semplice da implementare	Più complesso, richiede più componenti
Applicazioni	Sistemi semplici, bassa frequenza	Sistemi digitali avanzati, alta velocità

## Applicazioni Pratiche

- **Contatori asincroni:**
  - Timer digitali semplici.
  - Contatori di eventi a bassa frequenza.
  - Generatori di segnali di temporizzazione.
- **Contatori sincroni:**
  - CPU e registri nei computer.
  - Sistemi di controllo a microprocessore.
  - Generazione di segnali PWM per il controllo di motori.

# Full Adder: Caratteristiche e Funzionamento

Il Full Adder è un elemento fondamentale delle reti logiche combinatorie, in grado di sommare tre bit binari e generare un riporto utilizzabile per addizioni multi-bit. È un'evoluzione dell'Half Adder e costituisce la base degli **addizionatori multi-bit** presenti in CPU e dispositivi digitali. Il **Full Adder** è un circuito combinatorio in grado di sommare **tre bit binari**:



- **Due bit di ingresso** ( $A$  e  $B$ ).
- **Un riporto in ingresso** ( $Cin$ , carry-in), che proviene dall'addizionatore precedente in un'operazione multi-bit.

L'uscita del Full Adder è composta da:

- **Somma (S)** → il risultato della somma tra i tre bit di ingresso.
- **Riporto in uscita (Cout)** → il riporto generato dalla somma, che verrà passato allo stadio successivo.

## Funzionamento del Full Adder

Il Full Adder può essere visto come una combinazione di due **Half Adder**.

1. Il primo Half Adder calcola la somma tra i bit di ingresso  $A$  e  $B$ .
2. Il secondo Half Adder somma il risultato ottenuto con il bit di **carry-in (Cin)**.
3. Infine, un'operazione OR tra i riporti generati determina l'uscita **Cout**.

- ◆ **Formula per la somma (S):**  $S = A \oplus B \oplus Cin$
- ◆ **Formula per il riporto (Cout):**  $Cout = (A \cdot B) + (Cin \cdot (A \oplus B))$

## Schema Logico del Full Adder

- **2 porte XOR** per il calcolo della somma.
- **2 porte AND** per determinare i riporti parziali.
- **1 porta OR** per combinare i riporti.

## Vantaggi del Full Adder rispetto all'Half Adder

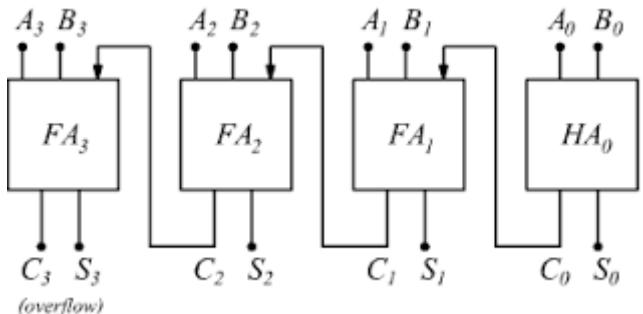
**Supporta il carry-in**, permettendo di concatenare più full adder per creare **addizionatori multi-bit**.

**Versatilità**, utilizzato nei processori e nelle unità aritmetico-logiche (ALU).

**Gestisce il riporto in ingresso**, permettendo il collegamento tra più bit per addizioni multi-bit.

# Addizionatori: Proprietà e Caratteristiche

Gli **addizionatori** sono circuiti combinatori fondamentali nelle reti logiche digitali, utilizzati per eseguire l'**operazione di somma binaria**. Sono alla base delle **unità aritmetico-logiche (ALU)** dei processori e vengono impiegati in vari dispositivi digitali. Gli **half adder** sono semplici e adatti a operazioni base, mentre i **full adder** permettono la costruzione di **addizionatori multi-bit** più complessi. Gli **addizionatori avanzati**, come il **Carry Lookahead Adder**, migliorano la velocità di calcolo e sono fondamentali nei moderni processori.



Esistono due tipi principali di addizionatori:

1. **Half Adder (Addizionatore Semplice)**
2. **Full Adder (Addizionatore Completo)**

Inoltre, combinando più full adder, si possono costruire **addizionatori a più bit** e addizionatori più avanzati, come l'**addizionatore con riporto anticipato (Carry Lookahead Adder)**.

**Calcolatrici Digitali** → Utilizzano addizionatori per eseguire somme binarie.

**Sistemi di Controllo Digitale** → Utilizzati nei microcontrollori per operazioni logiche.

**Circuiti per Elaborazione del Segnale** → Necessari in applicazioni audio e video.

## Addizionatore Parallello

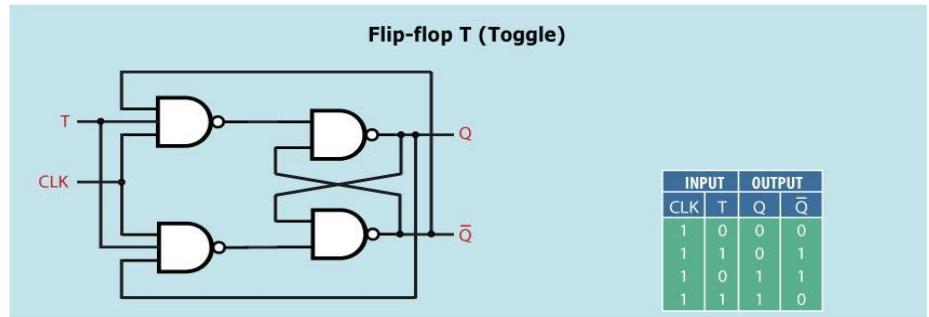
Un addizionatore parallelo è un circuito combinatorio che somma due numeri binari di  $n$  bit trattando ogni bit simultaneamente. È composto da una serie di **Full Adder (FA)** collegati in cascata, con il carry che viene propagato da uno stadio all'altro.

Pro	Contro
Somma eseguita simultaneamente su tutti i bit	Il ritardo dipende dalla propagazione del carry
Implementazione semplice per numeri di piccola dimensione	Non efficiente per numeri molto grandi

L'addizionatore parallelo è una soluzione efficace per eseguire operazioni aritmetiche in circuiti digitali. Tuttavia, per ridurre il ritardo dovuto alla propagazione del carry, vengono spesso utilizzati addizionatori più avanzati come il **Carry Look-Ahead Adder (CLA)**.

# Flip-Flop T: Proprietà e Caratteristiche

Il Flip-Flop T (Toggle Flip-Flop) è un tipo di flip-flop che cambia stato ad ogni impulso di clock quando il segnale di ingresso è attivo. È un elemento fondamentale nei circuiti sequenziali ed è spesso utilizzato nei contatori e nei divisori di frequenza.



In tabella **Q(n)** è lo stato attuale - **Q(n+1)** è lo stato successivo dopo il fronte di clock.

## Funzionamento del Flip-Flop T

Il Flip-Flop T ha un ingresso **T** e un segnale di **clock (CLK)**. Il suo comportamento è il seguente:

- Se **T = 0**, il flip-flop mantiene lo stato precedente.
- Se **T = 1**, il flip-flop inverte lo stato precedente ad ogni impulso di clock (toggle).

## Realizzazione del Flip-Flop T

Il Flip-Flop T può essere realizzato utilizzando un **Flip-Flop JK**, ponendo entrambi gli ingressi J e K a 1 ( $J = K = T$ ).

## Proprietà del Flip-Flop T

- **Memorizzazione dello stato** → mantiene l'ultimo valore se  $T = 0$ .
- **Toggle a ogni impulso di clock** → utile nei contatori e nei divisori di frequenza.
- **Realizzazione semplice** → può essere costruito a partire da Flip-Flop JK o D.

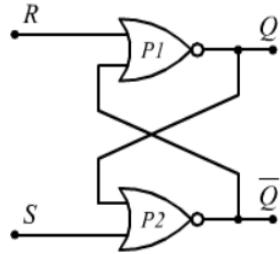
## Conclusioni

Il **Flip-Flop T** è un elemento chiave nei circuiti sequenziali, particolarmente utile per realizzare **contatori e divisori di frequenza**. Grazie alla sua capacità di cambiare stato a ogni impulso di clock, è impiegato in circuiti di temporizzazione e controllo digitale, rendendolo essenziale nei sistemi elettronici.

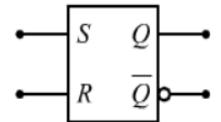
# Latch SR: Proprietà e Caratteristiche

## Il Latch SR (Set-Reset Latch)

è un circuito sequenziale bistabile utilizzato per memorizzare uno stato logico. È il tipo più semplice di latch e rappresenta la base per la costruzione di flip-flop e altri elementi di memoria nei circuiti digitali.



S	R	$Q_{n+1}$	$\bar{Q}_{n+1}$	
0	0	$Q_n$	$\bar{Q}_n$	memoria
0	1	0	1	reset
1	0	1	0	set
1	1	X	X	non ammesso



## Funzionamento del Latch SR

Il Latch SR ha due ingressi principali:

- **S (Set)** → Imposta l'uscita a 1.
- **R (Reset)** → Imposta l'uscita a 0.

Lo stato dell'uscita viene mantenuto fino a quando non si verifica un nuovo cambiamento negli ingressi.

- **$Q(n+1)$**  è lo stato successivo dopo l'applicazione degli ingressi.
- La combinazione  **$S = 1, R = 1$**  è **indefinita**, poiché forzerebbe entrambe le uscite a valori complementari in modo incoerente.

## Proprietà del Latch SR

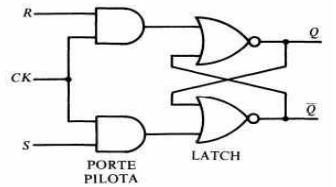
- **Memorizzazione dello stato** → Mantiene l'ultimo valore quando  $S = 0$  e  $R = 0$ .
- **Set e Reset esplicativi** → Può impostare direttamente lo stato a 1 o 0.
- **Semplice da implementare** → Utilizza solo due porte logiche.
- **Stato indefinito** → Se entrambi gli ingressi sono attivati contemporaneamente ( $S = 1, R = 1$  per NOR o  $S' = 0, R' = 0$  per NAND), il comportamento non è definito.

## Applicazioni del Latch SR

- **Memoria elementare** → Base per flip-flop e registri.
- **Controllo di eventi asincroni** → Utilizzato nei sistemi di interruzione e nei buffer di ingresso.
- **Sincronizzazione di segnali** → Adottato nei circuiti di temporizzazione.

# Latch SR Sincrono

Il **Latch SR sincrono** introduce un **segnale di clock (CLK)** per controllare quando il circuito può cambiare stato. Questo significa che le modifiche agli ingressi **S e R** vengono prese in considerazione solo quando **CLK è attivo** (tipicamente alto).



## Schema di Funzionamento

- Se **CLK = 0**, il latch mantiene il suo stato precedente, ignorando le variazioni di **S e R**.
- Se **CLK = 1**, il circuito si comporta come un normale Latch SR asincrono, rispondendo agli ingressi.

## Vantaggi del Latch SR Sincrono

- **Maggiore stabilità:** evita aggiornamenti non desiderati dovuti a glitch sugli ingressi.
- **Sincronizzazione con altri circuiti:** essenziale nei sistemi sequenziali complessi.
- **Base per i Flip-Flop:** molti Flip-Flop derivano dal latch SR sincronizzato.
- **Non sensibile ai glitch:** non commuta in modo indesiderato se gli ingressi cambiano troppo rapidamente.

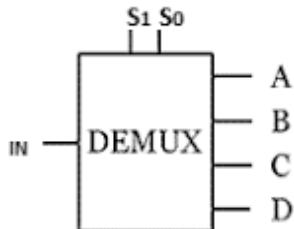
## Schema di Latch SR Sincrono con Porte NAND

Per realizzare un Latch SR sincrono, si aggiunge un **AND** tra **S e CLK** e tra **R e CLK**, in modo che il circuito reagisca solo quando il segnale di clock è attivo.

Caratteristica	Latch SR Asincrono	Latch SR Sincrono
<b>Clock</b>	Non presente	Necessario
<b>Aggiornamento dello stato</b>	Ogni volta che S o R cambiano	Solo quando il clock è attivo
<b>Rischio di glitch</b>	Elevato	Ridotto
<b>Utilizzo</b>	Buffer, memorie semplici	Registri, Flip-Flop

# Demultiplexer: Proprietà ed Esempi Applicativi

Il **demultiplexer (DEMUX)** è un circuito combinatorio utilizzato per indirizzare un singolo ingresso verso una delle molteplici uscite disponibili, in base ai valori di controllo. È l'opposto del **multiplexer (MUX)**, che invece seleziona un ingresso tra molti per trasmetterlo a un'unica uscita. Il suo utilizzo è fondamentale in sistemi digitali complessi, dove è necessario dirigere dati o segnali a destinazioni specifiche in base a criteri logici ben definiti.



S <sub>1</sub>	S <sub>0</sub>	A	B	C	D
0	0	IN	0	0	0
0	1	0	IN	0	0
1	0	0	0	IN	0
1	1	0	0	0	IN

## Struttura e Funzionamento del Demultiplexer

- **Un solo ingresso dati**
- **n ingressi di selezione** che determinano quale uscita attivare
- **2<sup>n</sup> uscite**

L'uscita selezionata riceve il valore dell'ingresso, mentre tutte le altre restano a 0.

## Tabella della verità per un DEMUX 1→4

Dove **IN** è il dato in ingresso, che viene instradato all'uscita corrispondente alla combinazione di **S1 S0**.

$$A = IN \cdot S1 \cdot S0$$

$$B = IN \cdot S1 \cdot S0$$

$$C = IN \cdot S1 \cdot S0$$

$$D = IN \cdot S1 \cdot S0$$

## Proprietà del Demultiplexer

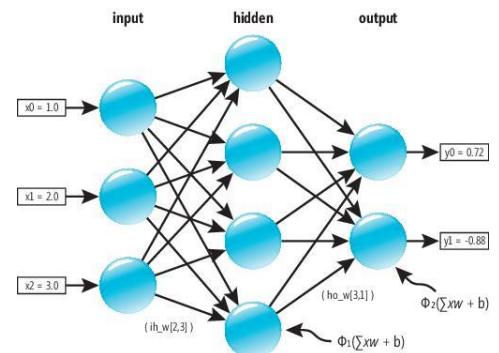
- **Indirizzamento diretto** → Il DEMUX invia un segnale a una sola uscita per volta.
- **Velocità di commutazione** → Determinata dal tempo di propagazione delle porte logiche.
- **Scalabilità** → Si possono combinare più demultiplexer per ottenere versioni più grandi (es. DEMUX 1→8 o 1→16).

## Esempi Applicativi del Demultiplexer

- **Sistemi di comunicazione** → Smistamento di segnali su diverse linee di trasmissione.
- **Memorie digitali** → Selezione di celle di memoria in dispositivi di archiviazione.
- **Display a segmenti** → Controllo dell'attivazione di singoli segmenti in un display.
- **Controllo di dispositivi** → Attivazione di componenti elettronici specifici (es. motori, LED, altoparlanti).

# Reti Neurali Statiche

Le **reti neurali statiche** sono un tipo di rete neurale artificiale in cui il flusso di informazioni avviene in un'unica direzione, senza retroazioni o stati interni che cambiano nel tempo. Questo le distingue dalle **reti neurali dinamiche**, che invece hanno connessioni ricorrenti e memoria. Le reti statiche sono particolarmente utilizzate per problemi di **classificazione, regressione e pattern recognition**, grazie alla loro capacità di apprendere funzioni complesse a partire da dati di input. Le reti neurali statiche sono fondamentali per molte applicazioni pratiche in **intelligenza artificiale e apprendimento automatico**.



## Hopfield Network

Le **reti di Hopfield** sono reti neurali ricorrenti usate come **memorie associative**. Ogni neurone è connesso a tutti gli altri e il sistema converge in stati stabili, che rappresentano i pattern memorizzati.

- Funzionano come una rete di memoria con stati discreti.
- L'aggiornamento avviene tramite **trasformazioni lineari** e **rappresentazione matriciale**.
- Sono limitate dal numero di pattern memorizzabili.

## Apprendimento Supervisionato e Regola di Hebb

L'**apprendimento supervisionato** avviene quando una rete viene addestrata con coppie input-output desiderate. La **regola di Hebb** stabilisce che la forza della connessione tra due neuroni aumenta se sono attivati contemporaneamente.

Viene usata in **memorie associative** e in alcuni modelli di apprendimento biologico.

## Widrow-Hoff Learning (Adaline e LMS)

L'**algoritmo Widrow-Hoff** è utilizzato nelle reti **Adaline (Adaptive Linear Neuron)**, basate su un modello lineare per la classificazione.

- La **rete Adaline** utilizza il **Mean Square Error (MSE)** per valutare l'errore tra output e target.
- L'**algoritmo Least Mean Squares (LMS)** minimizza l'errore regolando i pesi. È simile alla regola di **backpropagation**, ma viene applicato a reti neurali più semplici o a sistemi dinamici.

# Reti Neurali Statiche – Backpropagation

L'**algoritmo di Backpropagation** (retropropagazione dell'errore) è una tecnica di apprendimento supervisionato utilizzata per addestrare le **reti neurali artificiali** multilivello (**MLP - Multi-Layer Perceptron**). Il suo scopo è **aggiornare i pesi** della rete per ridurre l'errore tra output desiderato e output reale, usando il metodo della **discesa del gradiente**.

## Fasi del Backpropagation

### 1. Forward Pass:

- L'input viene propagato attraverso i **neuroni nascosti** fino all'**output**.
- Viene calcolato l'output effettivo della rete.

### 2. Calcolo dell'errore:

- Si misura la differenza tra output reale e desiderato, utilizzando una funzione di errore (tipicamente **Mean Square Error - MSE**).

### 3. Backward Pass:

- Si calcola il **gradiente dell'errore** rispetto ai pesi della rete.
- Si aggiornano i pesi utilizzando la **discesa del gradiente** dove viene calcolato il tasso di apprendimento.

## Perché la backpropagation è efficace?

- Permette di **apprendere in modo efficiente**: invece di dover ripetere l'intero processo per ogni neurone, si aggiorna solo in base all'errore calcolato, rendendo l'algoritmo molto più veloce ed efficace.
- È in grado di ottimizzare una rete complessa con **molti strati** di neuroni, consentendo l'apprendimento di rappresentazioni **non lineari**.

## Conclusioni

Dopo più iterazioni, i pesi continueranno ad aggiustarsi, minimizzando l'errore. Il **backpropagation** è fondamentale per il training delle reti neurali profonde e viene utilizzato in applicazioni come **riconoscimento vocale, visione artificiale e analisi dei dati**

Le **reti neurali statiche** come le **reti di Hopfield, le memorie associative e i modelli di apprendimento supervisionato** hanno applicazioni fondamentali in intelligenza artificiale e pattern recognition. Tecniche come **Adaline, Widrow-Hoff Learning e Backpropagation** permettono di addestrare reti in modo efficiente, minimizzando l'errore e migliorando la capacità di generalizzazione.

# ROM

La **ROM (Read-Only Memory)** è una memoria digitale non volatile utilizzata per memorizzare dati in modo permanente. Nella progettazione di circuiti digitali, la ROM è spesso implementata tramite matrici di transistor o reti di porte logiche. A differenza della RAM, il contenuto della ROM non può essere modificato durante il normale funzionamento del sistema.

La ROM è una componente essenziale per la memorizzazione permanente di dati e istruzioni. La sua implementazione varia da semplici matrici di porte logiche a memorie avanzate basate su transistor MOSFET.

## Struttura e Funzionamento

Dal punto di vista elettronico, la ROM è organizzata in una matrice di **celle di memoria**. Ogni cella può rappresentare un bit di dati, che può essere programmato durante la produzione o successivamente, a seconda del tipo di ROM.

Una ROM è composta da:

- **Linee di indirizzo** → Determinano la posizione del dato richiesto.
- **Decoder di indirizzo** → Seleziona la riga corrispondente all'indirizzo fornito.
- **Matrice di memoria** → Contiene i dati programmati (bit 0 o 1).
- **Circuiti di uscita** → Restituiscono il valore memorizzato all'uscita.

Quando viene fornito un indirizzo, il **decoder** attiva una riga nella matrice di memoria, facendo emergere i dati memorizzati che vengono inviati ai circuiti di uscita.

## Implementazione della ROM come Circuito Combinatorio

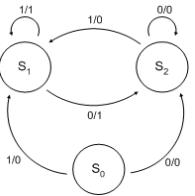
La ROM può essere realizzata con una rete combinatoria di porte logiche per implementare funzioni predefinite. Ad esempio, consideriamo una ROM con 2 bit di indirizzo e 3 bit di uscita:

Indirizzo (A1, A0)	Uscita (D2, D1, D0)
00	101
01	110
10	011
11	000

Questa tabella della verità può essere implementata con una rete di porte **AND**, **OR** e **NOT**, oppure con una matrice di transistor MOSFET programmabili.

# Automi a Stati Finiti (Mealy – Moore)

Gli **automi a stati finiti (Finite State Machines, FSM)** sono modelli matematici utilizzati in elettronica digitale per rappresentare sistemi con un numero finito di stati. Si dividono principalmente in due categorie:



1. **Automa di Moore**: l'output dipende solo dallo stato corrente.
2. **Automa di Mealy**: l'output dipende sia dallo stato corrente che dall'ingresso.

## Schema Logico

- **Automa di Moore**

- Output definito solo dallo stato attuale.
- Più stabile, ma con possibile ritardo nell'aggiornamento dell'output.
- Diagramma: Stato → Output
  - $q_0 \rightarrow Y_0$
  - $q_1 \rightarrow Y_1$
  - $q_2 \rightarrow Y_2$

- **Automa di Mealy**

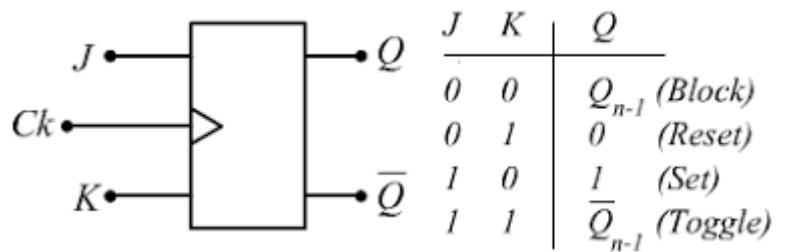
- Output dipendente dallo stato attuale e dall'ingresso.
- Più veloce nella risposta ai cambiamenti dell'ingresso.
- Diagramma: Stato + Ingresso → Output
  - $(q_0, X_0) \rightarrow Y_0$
  - $(q_1, X_1) \rightarrow Y_1$
  - $(q_2, X_2) \rightarrow Y_2$

Tipo di Automa	Pro	Contro
Moore	Maggiori stabilità	Maggior latenza nell'output
Mealy	Risposta più veloce	Output più sensibile ai cambiamenti dell'ingresso

Gli automi a stati finiti sono essenziali per il controllo di dispositivi digitali, dai processori ai protocolli di comunicazione. La scelta tra Moore e Mealy dipende dall'applicazione: Moore è più stabile e prevedibile, mentre Mealy è più reattivo e efficiente

## Latch JK

Il **Latch JK** è un circuito bistabile derivato dal **Latch SR** (Set-Reset) con l'aggiunta di una logica che evita lo stato instabile. Ha due ingressi, **J** e **K**, e un ingresso di abilitazione (**Enable o Clock, a seconda della configurazione**).



L'uscita del latch JK è determinata come segue:

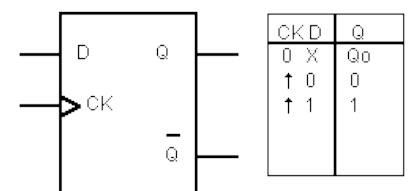
- Se  $J=0$  e  $K=0$  - Mantiene lo stato precedente.
- Se  $J=0$  e  $K=1$  -  $Q=0$  Reset
- Se  $J=1$  e  $K=0$  -  $Q=1$  Set
- Se  $J=1$  e  $K=1$  → Comutazione dello stato (toggle).
- 

Pro	Contro
Evita lo stato proibito del latch SR	Sensibile ai glitch nel clock
Può operare in modalità toggle	Può entrare in oscillazione con alcuni segnali

Il latch JK è una versione migliorata del latch SR, usata in circuiti di memoria e controllo. La sua capacità di commutare lo stato (toggle) lo rende utile in contatori e registri.

## Latch D

Il **Latch D (Data o Delay Latch)** è un circuito bistabile che memorizza un bit di dati e lo mantiene finché non viene aggiornato da un segnale di controllo (Enable o Clock). È una versione modificata del **Latch SR**, progettata per evitare lo stato proibito eliminando la necessità di ingressi separati per **Set** e **Reset**.

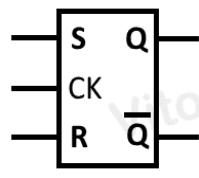


Pro	Contro
Semplice da implementare	Può introdurre ritardi nei segnali
Evita lo stato proibito del latch SR	Sensibile a glitch nell'Enable

Il Latch D è fondamentale nei registri e nei buffer di memoria, poiché permette di sincronizzare e trattenere i dati in un sistema digitale. Viene spesso usato come base per il **Flip-Flop D**, che introduce il controllo tramite clock per un comportamento più stabile.

# Flip-Flop SR

Il **Flip-Flop SR (Set-Reset)** è una versione **sincrona** del **Latch SR**, ovvero aggiorna il suo stato solo in corrispondenza di un segnale di **clock**. Viene utilizzato per memorizzare un bit e fa parte delle **reti sequenziali** digitali.



S	R	Q
0	0	memoria
0	1	0
1	0	1
1	1	non ammesso

È composto da due ingressi:

- **S (Set)** → Imposta l'uscita  $Q=1$
- **R (Reset)** → Porta l'uscita  $Q=0$
- **CLK (Clock)** → Controlla quando il flip-flop può cambiare stato.

L'uscita cambia solo quando il **clock è attivo**, tipicamente sul **fronte di salita** o **fronte di discesa** del segnale di clock.

CLK	S	R	Q (stato successivo)
↑	0	0	Q (mantiene lo stato)
↑	0	1	0 (reset)
↑	1	0	1 (set)
↑	1	1	Stato proibito

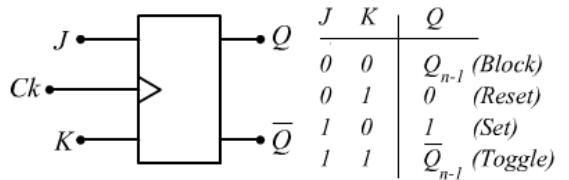
**Nota:** La condizione  $S=1, R=1$  è **indefinita**, quindi nei circuiti reali si preferisce utilizzare il **Flip-Flop JK**, che risolve questo problema.

Pro	Contro
Sincrono, quindi più stabile di un latch	Stato proibito per $S=1, R=1$
Usato nei circuiti sequenziali e registri	Struttura più complessa rispetto al latch

Il Flip-Flop SR è il punto di partenza per la memorizzazione nei sistemi digitali, ma la sua limitazione dello stato proibito lo rende meno usato rispetto ad altre varianti come il **Flip-Flop JK** o **Flip-Flop D**.

## Flip-Flop JK

Il **Flip-Flop JK** è un'evoluzione del **Flip-Flop SR**, progettato per risolvere il problema dello **stato proibito** ( $S=1, R=1$ ). È un flip-flop **sincrono**, il che significa che aggiorna il suo stato solo quando riceve un impulso di **clock (CLK)**.



Ha due ingressi principali:

- **J (Set)** → Funziona come *S* nel Flip-Flop SR.
- **K (Reset)** → Funziona come *R* nel Flip-Flop SR.
- **CLK (Clock)** → Controlla il momento in cui avviene il cambio di stato.

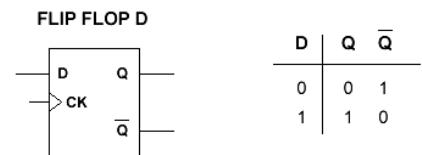
**Nota:** Quando  $J=1$  e  $K=1$ , lo stato di  $Q$  si **inverte** rispetto a quello precedente.

Pro	Contro
Risolve il problema dello stato proibito del Flip-Flop SR	Più complesso rispetto ad altri flip-flop
Può funzionare come Flip-Flop SR, D o T	Il toggle può creare instabilità se non controllato bene
Usato in registri, contatori e circuiti sequenziali	-

Il Flip-Flop JK è una delle versioni più versatili ed è utilizzato in **contatori, registri e circuiti sequenziali**, grazie alla sua capacità di **toggle** e alla rimozione dello stato proibito.

## Flip-Flop D

Il **Flip-Flop D (Delay o Data Flip-Flop)** è un tipo di flip-flop **sincrono** utilizzato per **memorizzare un bit di dati**. La sua caratteristica principale è che l'uscita segue semplicemente il valore dell'ingresso **D** in corrispondenza del **fronte del segnale di clock (CLK)**.

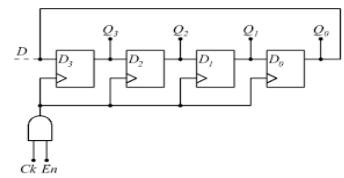


Viene utilizzato per **registrare e sincronizzare** segnali digitali ed è una base per la realizzazione di **registri a scorrimento, contatori e memorie sequenziali** ma non permette il toggle come il Flip-Flop JK.

Il Flip-Flop D **copia il valore di D** sull'uscita  $Q$  al **fronte di salita (o discesa) del clock** ed è ampiamente utilizzato in **registri, buffer e pipeline di processori**, poiché garantisce che i dati vengano trasferiti e memorizzati in modo preciso e controllato dal segnale di **clock**.

# Registri e Contatori

I **registri** sono circuiti sequenziali costituiti da un insieme di **flip-flop** che memorizzano più bit di informazione. Ogni flip-flop memorizza un bit, quindi un registro a **n bit** è composto da **n flip-flop** collegati in parallelo.



## Tipologie di registri

1. **Registro a scorrimento (Shift Register)** → I dati vengono spostati a destra o sinistra tra i flip-flop, utili per la **conversione seriale-parallela** e viceversa.
2. **Registro con caricamento parallelo** → Tutti i bit vengono caricati contemporaneamente tramite un segnale di clock.
3. **Registro bidirezionale** → Può far scorrere i dati sia a destra che a sinistra.
4. **Registro a carica controllata** → I dati vengono memorizzati solo quando un segnale di controllo è attivo.

Pro	Contro
Permettono la memorizzazione temporanea di dati	Necessitano di un clock sincronizzato
Utili per sincronizzare segnali digitali	Possono occupare spazio nei circuiti complessi

I **contatori** sono circuiti sequenziali che incrementano o decrementano un valore in base agli impulsi di clock. Sono formati da una serie di **flip-flop** collegati tra loro.

## Tipologie di contatori

1. **Contatori asincroni (Ripple Counter)** → Ogni flip-flop è attivato dall'uscita del precedente, introducendo un ritardo di propagazione.
2. **Contatori sincroni** → Tutti i flip-flop cambiano stato simultaneamente, garantendo maggiore velocità e stabilità.
3. **Contatori modulo-k (Mod-k Counter)** → Contano fino a un valore massimo  $k$  e poi si azzerano.
4. **Contatori UP/DOWN** → Possono contare sia in avanti che all'indietro.

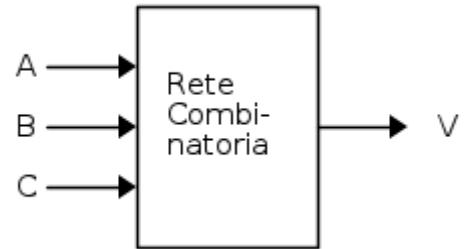
Pro	Contro
Utilizzati per il conteggio e la generazione di segnali temporizzati	I contatori asincroni possono avere ritardi
Essenziali nei sistemi digitali per la gestione di eventi	I contatori sincroni richiedono più logica di controllo

I registri e i contatori sono due elementi fondamentali nei **circuiti sequenziali digitali**. I registri vengono usati per la **memorizzazione e il trasferimento** di dati, mentre i contatori servono per **conteggi, temporizzazioni e generazione di impulsi** in molti dispositivi elettronici.

**EXTRA**

# Rete Combinatoria

Una **rete logica combinatoria** è un circuito digitale in cui l'uscita dipende esclusivamente dallo stato attuale degli ingressi, senza l'uso di elementi di memoria. Questo significa che ogni combinazione degli ingressi produce un'uscita ben definita in base a una funzione logica predefinita.



## Caratteristiche Principali

- **Nessuna memoria:** l'uscita è determinata solo dagli ingressi presenti in quell'istante.
- **Tempo di propagazione:** il tempo necessario affinché una variazione dell'ingresso si rifletta sull'uscita dipende dai ritardi dei gate logici.
- **Realizzabile con porte logiche:** qualsiasi funzione combinatoria può essere costruita utilizzando le porte AND, OR, NOT e, in alcuni casi, NAND, NOR, XOR.
- **Funzionamento deterministico:** per ogni combinazione di ingressi, l'uscita è univocamente definita.

## Rappresentazione con Tabelle della Verità ed Espressioni Booleane

Le reti combinatorie sono descritte mediante:

- **Tabella della verità**, che elenca tutte le possibili combinazioni degli ingressi e le corrispondenti uscite.
- **Espressioni booleane**, che rappresentano il comportamento del circuito attraverso l'algebra booleana.

## Tipologie di Reti Combinatorie

- **Sommatore e Sottrattore:** circuiti che eseguono operazioni aritmetiche di base.
- **Multiplexer:** seleziona uno tra più ingressi e lo inoltra all'uscita.
- **Demultiplexer:** distribuisce un segnale di ingresso su più linee di uscita.
- **Codificatori/Decodificatori:** convertitori di segnali da un formato all'altro.

# Decoder

Un **decoder** è un circuito logico combinatorio che converte un codice di ingresso binario in un'uscita univoca. Viene utilizzato per selezionare una tra più linee di uscita in base alla combinazione di bit presenti all'ingresso.

A	B	Z <sub>0</sub>	Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder 2 - 4

## Caratteristiche Principali

- **Numero di ingressi e uscite:** Un decoder con  $n$  ingressi ha  $2^n$  uscite, in quanto ogni combinazione degli ingressi corrisponde a una linea di uscita attivata.
- **Funzionamento combinatorio:** L'uscita è determinata unicamente dagli ingressi attuali, senza memoria.
- **Espansione della logica:** Utilizzato per convertire codici binari in segnali di controllo o per attivare dispositivi specifici.

## Tipologie di Decoder

1. **Decoder binario a decimale:** Converte un numero binario in un'uscita corrispondente al suo valore decimale.
2. **Decoder di indirizzi:** Utilizzato nelle memorie per selezionare una specifica cella.
3. **Decoder 3-to-8 e 4-to-16:** Espandono il numero di linee selezionabili.
4. **BCD-to-7-Segment Decoder:** Converte un codice binario in segnali per pilotare un display a 7 segmenti.

## Applicazioni

- **Memorie RAM e ROM:** Selezione di celle di memoria.
- **Microprocessori:** Decodifica degli indirizzi nelle architetture di calcolo.
- **Display e indicatori:** Conversione di numeri binari in codici visivi.
- **Sistemi di controllo industriali:** Selezione di dispositivi o attuatori specifici.

## Conclusione

I decoder sono circuiti essenziali nell'elettronica digitale per la gestione della logica combinatoria e della selezione di linee di segnale. Grazie alla loro capacità di trasformare ingressi binari in uscite univoche, trovano applicazione in molteplici sistemi, dalla memoria dei computer ai display digitali.

# Priority Encoder

Un **priority encoder** (encoder con priorità) è un circuito logico combinatorio che converte un ingresso attivo tra più linee in un codice binario, assegnando priorità agli ingressi più significativi. Questo significa che, se più ingressi sono attivi contemporaneamente, viene codificato solo quello con la priorità più alta.

## Caratteristiche Principali

- **Selezione della priorità:** Gli ingressi con indice più alto hanno precedenza su quelli con indice più basso.
- **Numero di ingressi e uscite:** Un priority encoder con  $2^n$  ingressi produce n uscite.
- **Gestione di ingressi multipli:** Ignora ingressi attivi con priorità inferiore.
- **Bit di validità (V):** Indica se almeno un ingresso è attivo.

## Applicazioni

- **Gestione delle interruzioni nei processori:** Determina quale dispositivo ha priorità in una richiesta di interrupt.
- **Multiplexing dei segnali:** Seleziona il segnale più rilevante in base alla priorità.
- **Decoder di tastiere:** Identifica quale tasto è stato premuto in base a una priorità predefinita.

Il priority encoder è essenziale nei sistemi digitali per la gestione delle priorità tra più segnali in ingresso. Grazie alla sua capacità di assegnare importanza agli ingressi più significativi, viene utilizzato nei microprocessori, nei sistemi embedded e nelle periferiche digitali.

# Sommatori Carry Look-Ahead e Carry Save

I sommatori sono circuiti combinatori utilizzati per eseguire l'addizione di numeri binari. Due delle architetture più avanzate sono il Carry Look-Ahead Adder (CLA) e il Carry Save Adder (CSA), progettati per migliorare l'efficienza rispetto ai sommatori tradizionali come il Ripple Carry Adder (RCA).

## Carry Look-Ahead Adder (CLA)

Il **sommatore Carry Look-Ahead** è progettato per ridurre il ritardo di propagazione del riporto (carry), calcolando anticipatamente quali bit genereranno e propagheranno il riporto.

### Vantaggi

- Maggiore velocità rispetto al Ripple Carry Adder (RCA).
- Riduce il ritardo di propagazione del riporto.

### Svantaggi

- Più complesso e richiede più porte logiche.
- Il numero di porte aumenta con la dimensione dei numeri.

## Carry Save Adder (CSA)

Il **Carry Save Adder** viene utilizzato per eseguire l'addizione di più numeri binari senza propagare immediatamente il riporto. Questo metodo è particolarmente utile nei moltiplicatori binari.

### Vantaggi

- Riduce il numero di operazioni sequenziali necessarie.
- Molto efficiente nei moltiplicatori hardware.

### Svantaggi

Richiede un passo di somma finale con un altro tipo di sommatore.

- Il **Carry Look-Ahead Adder (CLA)** è ideale per operazioni aritmetiche rapide in unità logico-aritmetiche (ALU).
- Il **Carry Save Adder (CSA)** è più efficiente quando si devono sommare più numeri contemporaneamente, come nei moltiplicatori.

# Reti Neurali Dinamiche

Le **reti neurali dinamiche** si differenziano da quelle statiche perché includono **connessioni retroattive e dipendenza temporale** nello stato dei neuroni. Ciò le rende adatte a modellare fenomeni sequenziali e processi con memoria, come il riconoscimento del parlato o il controllo di sistemi dinamici.

## Layered Digital Dynamic Networks (LDDN)

Le **Layered Digital Dynamic Networks (LDDN)** sono architetture neurali con strati multipli, caratterizzate da una propagazione digitale e strutturata dell'informazione. In queste reti, ogni livello elabora l'input secondo una dinamica predefinita, spesso regolata da **equazioni differenziali** o **automi finiti**.

### Caratteristiche:

- Ogni strato ha **connessioni temporali**, che permettono la persistenza dell'informazione.
- Possono essere **feedforward** (senza retroazione) o **recurrent** (con loop interni).
- Utilizzate in **controllo automatico e modellazione di sistemi digitali complessi**.

## Backpropagation Through Time (BPTT) e Backpropagation Order

L'**algoritmo Backpropagation Through Time (BPTT)** è un'estensione del classico **Backpropagation** per reti neurali dinamiche. Serve per addestrare **reti ricorrenti** unwrapping la rete su una finestra temporale e applicando la discesa del gradiente su ogni istante di tempo.

### Backpropagation Order:

- **Ordine 0:** Addestramento senza propagazione temporale (equivalente al feedforward).
- **Ordine 1:** Considera un solo passo temporale passato per la correzione dei pesi.
- **Ordine N:** Propaga l'errore per **N istanti precedenti**, aumentando l'efficacia dell'apprendimento ma anche la complessità computazionale

## Reti Neurali Dinamiche – Feedforward

Sono un'estensione delle reti feedforward classiche, ma con connessioni dipendenti dal tempo. Questo consente alla rete di adattarsi in funzione dell'evoluzione dell'input.

### Applicazioni:

- **Filtraggio di segnali dinamici.**
- **Analisi di dati con dipendenza temporale.**

## Reti Neurali Dinamiche Ricorrenti (RNN)

Le **reti ricorrenti (Recurrent Neural Networks - RNN)** introducono **connessioni retroattive**, che permettono la memorizzazione di informazioni passate.

### Tipologie:

1. **Simple Recurrent Networks (SRN):** Un solo strato nascosto con connessioni ricorrenti.
2. **Long Short-Term Memory (LSTM):** Introducono **celle di memoria** per evitare il problema della vanishing gradient.
3. **Gated Recurrent Units (GRU):** Variante più efficiente delle LSTM.

### Applicazioni:

- **Elaborazione del linguaggio naturale (NLP).**
- **Riconoscimento di sequenze.**

## Simple Associative Network e Instar Rule

Le **Simple Associative Networks** sono reti neurali progettate per **memorizzare e riconoscere pattern associativi**. L'**Instar Rule** è un metodo di apprendimento usato per modificare i pesi sinaptici in base alla correlazione tra input e output.

## Simple Recall Network e Output Rule

Le **Simple Recall Networks** sono reti che memorizzano pattern e li recuperano quando ricevono input simili. L'**Output Rule** regola il comportamento della rete durante la fase di **richiamo** di un pattern precedentemente appreso.

# Reti Neurali Dinamiche – Radial Basis Function Networks (RBFN)

Le **Radial Basis Function Networks** sono reti neurali con **funzioni di attivazione gaussiane**, usate principalmente per **applicazioni di interpolazione e classificazione**.

## Caratteristiche:

- Struttura **feedforward**.
- I neuroni nascosti usano **funzioni radiali gaussiane** invece della classica funzione sigmoide.
- Utilizzate in **riconoscimento vocale e controllo robotico**.

## Algoritmo K-Means e sue varianti per RBFN

L'**algoritmo K-Means** è una tecnica di clustering usata per **inizializzare i centri dei neuroni nascosti** nelle RBFN. Varianti di K-Means per le RBFN includono:

- **K-Means adattivo**, che aggiorna dinamicamente il numero di cluster.
- **K-Means fuzzy**, che assegna un grado di appartenenza a ciascun cluster.

## Conclusioni

Le **reti neurali dinamiche** sono essenziali per applicazioni in cui il **tempo** e la **memoria delle informazioni passate** giocano un ruolo chiave. Grazie agli algoritmi di addestramento come il **Backpropagation Through Time** e l'uso di tecniche avanzate come le **RBFN**, possono essere applicate con successo in **previsione, analisi sequenziale e controllo automatico**.

# Benefici delle Reti Neurali

Le **reti neurali artificiali (ANN - Artificial Neural Networks)** offrono numerosi vantaggi nelle applicazioni moderne, grazie alla loro capacità di apprendere da grandi quantità di dati e di adattarsi a problemi complessi.

Di seguito vengono evidenziati i principali benefici:

## Capacità di Apprendimento e Generalizzazione

Le reti neurali possono **apprendere autonomamente** da set di dati di addestramento, migliorando le proprie prestazioni nel tempo. Inoltre, riescono a generalizzare le conoscenze acquisite per **fare previsioni su nuovi dati non visti durante l'addestramento**.

**Esempio:** Un sistema di riconoscimento delle immagini può identificare volti anche se l'illuminazione o l'angolazione della foto cambiano.

## Adattabilità e Autonomia

Una rete neurale può adattarsi dinamicamente a nuovi dati senza bisogno di riprogettare completamente il sistema. Questo la rende ideale per **applicazioni in ambienti mutevoli**.

**Esempio:** Algoritmi di **trading automatico** che aggiornano le loro strategie basandosi su nuovi dati di mercato.

## Parallelismo e Capacità Computazionale

Le ANN sfruttano **operazioni parallele** che permettono loro di elaborare enormi quantità di dati simultaneamente, rendendole molto efficaci in contesti come **elaborazione di immagini e analisi predittiva**.

**Esempio:** In **medicina**, le reti neurali analizzano milioni di immagini radiologiche per rilevare anomalie in pochi secondi.

## Capacità di Riconoscere Pattern Complessi

Grazie alla loro struttura stratificata, le reti neurali sono particolarmente efficaci nel riconoscere pattern nascosti nei dati, anche quando questi non sono immediatamente evidenti agli esseri umani.

**Esempio:** **Sistemi di rilevamento delle frodi bancarie**, che identificano transazioni sospette analizzando anomalie nei dati.

# Benefici - Automazione di Compiti Complessi

Le reti neurali consentono di **automatizzare attività** che richiederebbero molto tempo e risorse se eseguite manualmente, migliorando efficienza e riducendo errori umani.

**Esempio:** Chatbot intelligenti che comprendono e rispondono alle domande dei clienti in linguaggio naturale.

## Miglioramento Continuo con l'Aumento dei Dati

A differenza di molti algoritmi tradizionali, le reti neurali **migliorano le proprie prestazioni man mano che ricevono nuovi dati**, grazie alla possibilità di ri-addestramento.

**Esempio:** Algoritmi di **motori di ricerca** che affinano i risultati sulla base delle interazioni degli utenti.

## Tabella Riassuntiva

Beneficio	Descrizione	Esempio
<b>Apprendimento e Generalizzazione</b>	Apprende dai dati e fa previsioni su nuovi input.	Riconoscimento facciale.
<b>Adattabilità</b>	Si aggiorna automaticamente con nuovi dati.	Trading automatico.
<b>Parallelismo Computazionale</b>	Elabora grandi dataset in poco tempo.	Diagnosi medica da immagini.
<b>Riconoscimento Pattern Complessi</b>	Individua relazioni nei dati anche non ovvie.	Rilevamento frodi bancarie.
<b>Automazione</b>	Sostituisce il lavoro manuale in compiti ripetitivi.	Chatbot intelligenti.
<b>Miglioramento Continuo</b>	Più dati portano a maggiore accuratezza.	Algoritmi di ricerca.

# Neurone e Percettrone

Il **neurone artificiale** è il modello matematico ispirato al funzionamento del neurone biologico. È l'unità fondamentale delle **reti neurali artificiali (ANN - Artificial Neural Networks)** e opera trasformando ingressi numerici in un'uscita attraverso un processo di **pesatura, somma e attivazione**.

## Struttura del neurone artificiale:

- **Ingressi ( $x_1 \dots x_n$ )**: Valori numerici che rappresentano le informazioni in ingresso.
- **Pesi sinaptici ( $w_1 \dots w_n$ )**: Coefficienti che determinano l'importanza di ciascun ingresso.
- **Sommatore**: Calcola la somma pesata degli ingressi
- **Funzione di attivazione (f)**: Applica una trasformazione all'output per decidere se il neurone si attiva o meno.

Il **percettrone** è il primo modello di neurone artificiale introdotto da **Frank Rosenblatt nel 1958**. Si tratta di un classificatore binario che separa i dati in due classi mediante una funzione di attivazione a soglia.

## Funzionamento del percettrone:

1. Riceve più **ingressi ( $x_1 \dots x_n$ )**
2. Moltiplica ciascun ingresso per un peso  **$w_i$**
3. Somma i prodotti e aggiunge un **bias**.
4. Applica una **funzione di attivazione** (tipicamente una funzione a soglia).
5. Genera un **output binario: 1 (attivato) o 0 (non attivato)**.

## Limiti del Percettrone

Il percettrone è efficace solo per problemi **linearmente separabili** (come AND e OR), ma **non può risolvere problemi più complessi** come il **problema XOR**.

Per superare questo limite, sono stati sviluppati modelli più avanzati, come il **Percettrone Multistrato (MLP)** con funzioni di attivazione non lineari e **reti neurali profonde (Deep Learning)**.

## Neurone e Percettrone - Tabella Riassuntiva

Caratteristica	Neurone Artificiale	Percettrone
Definizione	Modello matematico ispirato ai neuroni biologici.	Algoritmo di classificazione basato su un neurone artificiale.
Componenti	Ingressi, pesi, somma, funzione di attivazione.	Ingressi, pesi, somma, funzione a soglia.
Output	Valore continuo o discreto.	Valore binario (0 o 1).
Applicazioni	Reti neurali complesse, deep learning.	Classificazione di dati linearmente separabili.
Limiti	Nessuno, dipende dalla struttura della rete.	Non gestisce problemi non linearmente separabili (es. XOR).