

## Docente/i

Annalisa Massini / Salvatore Pontarelli

## Appunti e esami (De Cristo)

### Twiki Gorla

<https://twiki.di.uniroma1.it/twiki/view/Architetture1/AL/WebHome>

### Verilog

<https://hdlbits.01xz.net/>

---

## La salvezza

### Boolean Expression Solver

<https://www.emathhelp.net/en/calculators/discrete-mathematics/boolean-algebra-calculator>

### Float Toy (IEEE 754)

<https://evanw.github.io/float-toy/>

### IEEE 754 Calculator

In fondo al sito si può cambiare tra 16/32/64/128 bit

<http://weitz.de/ieee/>

### Karnaugh map

<http://www.32x8.com/var2.html>

---

## Rappresentazione di un informazione

Un informazione si rappresenta utilizzando:

- Un alfabeto di supporto
- Parole (sequenze) finite di simboli dell'alfabeto

Un informazione è codificata utilizzando una funzione che associa l'informazione dell'insieme B a parole nell'insieme C (chiamato codice)

Se ho più parole che codici, ho una codifica ridondante, altrimenti una ambigua

Una rappresentazione deve essere:

- Economica (minor numero di caratteri possibili)
  - Semplice da codificare e decodificare
  - Semplicità di elaborazione (data da una buona rappresentazione)
- 

## Operazioni binarie

### Complemento a 2

OVERFLOW

L'overflow è valutato in base al segno.

- Se sommo due termini positivi e ottengo un risultato negativo
- Se sommo due numeri negativi e ottengo un risultato positivo

### Sottrazione

Per sottrarre due numeri  $X$  e  $Y$  in base 2 ( $X-Y$ ):

- cambiare il segno di  $Y$ ; OBJ
  - Se non hanno lo stesso numero di bit, aggiungere  $n = n_1 - n_2$  con (  
 $n_1, n_2 = \text{lunghezza in bit di primo e secondo numero}$ ) 1 a destra (che per la proprietà delle potenze lascia il numero invariato);
  - eseguire la somma tra i due numeri.
- 

## Virgola fissa

La virgola fissa è una metodologia di rappresentazione basata sull'avere  $n$  cifre e di vedere le prime  $m$  come parte intera e le successive  $d$  come parte decimale.

### Problematiche

l'intervallo di numeri rappresentabili è molto piccolo e le approssimazioni sono grossolane.

Ad esempio avendo a disposizione 32 bit e assegnandone 20 per la P.I. (in Ca2) e 12 per la P.F. si ha come intervallo di rappresentazione  $\{-524.287, \dots, 524.287\}$

Chiaramente possiamo giocare con la dimensione di P.I e P.F per aumentare la precisione della parte decimale (a scapito dell'ampiezza dell'intervallo).

Non è una notazione adatta ai calcoli scientifici reali

---

## Virgola mobile (IEEE 754)

Notazione in virgola mobile usata nei calcolatori formata dalla tripla di valori

$< \text{segno}; \text{esponente}; \text{mantissa} >$ .

Il segno se vale 0 rappresenta + altrimenti -.

Nella rappresentazione prima della virgola si mette sempre 1, e dopo si riporta la mantissa.

L'uno davanti è dato per scontato esserci e questo ci fa risparmiare un bit quando andiamo a rappresentare il valore in IEEE 754

Si ispira alla notazione scientifica

**Nota:** Il numero scritto in forma 1,... rappresenta il valore della mantissa e viene usato per fare i calcoli.

### Video utile

 IEEE Floating Point Representation

### Forme

Precision	Bit	Esponente	Mantissa
Half-Precision	16	5	10
Single-Precision	32	8	23
Double-Precision	64	11	52

### Numeri speciali

Tipo	Esponente	Significando	Mantissa
Zeri	0	1	0
Numeri	0	0	$m \neq 0$

denaturalizzati			
Infiniti	$2^e - 1$ (sequenza di 1)	1	0
NAN	$2^e - 1$ (sequenza di 1)	1	$m \neq 0$
Numeri normali	$[1, 2^e - 2]$	1	Qualunque

## Numeri rappresentabili

tipo	valore	binario
numero subnormale positivo più piccolo	$2^{-14} \times (0 + \frac{1}{1024}) \approx 0.000000059604645$	0 00000 0000000001
numero subnormale più grande	$2^{-14} \times (0 + \frac{1023}{1024}) \approx 0.000060975552$	0 00000 1111111111
numero normale positivo più piccolo	$2^{-14} \times (1 + \frac{0}{1024}) \approx 0.00006103515625$	0 00001 0000000000
numero normale più grande	$2^{15} \times (1 + \frac{1023}{1024}) \approx 65504$	0 11110 1111111111

- precisione della rappresentazione: distanza tra due numeri adiacenti
- ampiezza della rappresentazione: caratterizzata dal valore assoluto del più grande/piccolo numero rappresentabile

Sarebbe ideale sia una maggior precisione che una maggior ampiezza, queste dipendono dal numero di bit dedicati a mantissa ed esponente.

## Decimale

$$(-1)^{\text{segno}} * 1 + \text{mantissa in decimale} * 2^{\text{esponente (non biased)}}$$

### Esempio con negativo

$$1\ 01111\ 0100000000 = (-1)^1 * 1 + 0,25 * 2^0 = -1,25$$

poiché la mantissa è 0100000000 = 0,25

e l'esponente è 01111 = 15 - 15 = 0

### Esempio con positivo

$0\ 10000\ 0010000000 = (-1)^{10} * 1 + 0,125 * 2^1 = 2,25$   
 poiché la mantissa è  $0010000000 = 0,125$   
 e l'esponente è  $01111 = 16 - 15 = 1$

## Ottenere la mantissa

- Moltiplico la parte decimale per la base b
- Itero sulla parte decimale finché:
  - Ottengo 0 come risultato
  - Ottengo il numero di bit necessari
- Potrei trovare un pattern (ripetizioni) da cui deduco che i bit dopo sono uguali a quel pattern

La rappresentazione è data dalle parti intere nell'ordine in cui le abbiamo ottenute.

## Esempio (base 2)

$13,75 \Rightarrow 0,75 * 2 = 1,5 \Rightarrow 0,5 * 2 = 1,0$   
 Quindi  $0,75_2 = 11$

## Somma

$$\langle s_1, m_1, e_1 \rangle + \langle s_2, m_2, e_2 \rangle = \langle s', m', e' \rangle$$

con

$$s = 0 \text{ se } 0, m_1 \geq 0, m_2$$

$$s = 1 \text{ altrimenti}$$

1. Sia  $e_1 = e_2$

$m, e$  sono le normalizzazioni di  $m', e'$

$$e' = e_1 (= e_2)$$

$$m' = 0, m_1 + 0, m_2 \text{ se } s_1 = s_2$$

$$m' = 0, m_1 - 0, m_2 \text{ se } s_1 \neq s_2 \text{ e } 0, m_1 \geq 0, m_2$$

$$m' = 0, m_1 - 0, m_2 \text{ altrimenti}$$

2. Sia  $e_1 < e_2$

a. Slitto a destra  $m_1$  di  $e_2 - e_1$  posizioni (inserendo 0 a sinistra)

(N.B.: in questo passaggio intermedio, il primo operando non è più normalizzato!!)

(N.B.: ci può essere perdita di cifre in coda a  $m_1$ , al limite  $m_1$  si potrebbe azzerare!!)

- b. Così il primo operando avrà come esponente  $e_2$
  - c. Procedo come nel punto 1
3. Sia  $e_1 > e_2$
- a. Come il punto 2, ma porta il secondo operando all'esponente del primo

### Se uno dei due numeri è negativo

Dovrò fare il  $\bar{C}a_2$  della mantissa del numero negativo, eseguire la somma delle mantisse dei due numeri e poi normalizzare il risultato, se serve, eseguendo il complemento a 2 di nuovo (nel caso uscisse negativo quando deve essere positivo (la mantissa deve essere sempre positiva in quanto il segno è dato dalla rappresentazione)).

### Esempio

$$x = 0\ 10000\ 00100000000$$

$$y = 1\ 01111\ 01000000000$$

$$e_b^x = 10000 = 16, e^x = 16 - 15 = 1$$

$$e_b^y = 01111 = 15, e^y = 15 - 15 = 0$$

L'esponente di  $x$  è più grande,  $1 - 0 = 1$  quindi shifto  $y$  di una posizione

$$y = 1,0100000000 * 2^0 \Rightarrow 0,1010000000 * 2^1$$

$$x = 1,0010000000 * 2^1$$

Eseguo la sottrazione, poiché  $y$  è negativo

$y$  ha una mantissa con magnitudo più alto, lo metto per primo

$$\begin{array}{r} 1,1010000000 * 2^1 \\ - 1,0010000000 * 2^1 \\ \hline 0,1000000000 * 2^1 \end{array}$$

Devo normalizzare il risultato, dovrò quindi shiftare di un bit a sinistra, decrementando così l'esponente

$$res = 0,1000000000 * 2^1 \Rightarrow 1,0000000000 * 2^0$$

Riscrivo il risultato in notazione IEEE 754

Il segno è positivo perché viene 0, e perché effettivamente abbiamo sottratto ad un positivo più grande un negativo più piccolo

L'esponente è

$$e^{res} = 0 + 15 = 15 = 01111$$

La mantissa è composta da soli 0 come vediamo nel risultato

Dunque

$$res = \langle 0; 01111; 0000000000 \rangle$$

## Sottrazione

Equivale a dire

$$\langle s_1, m_1, e_1 \rangle - \langle s_2, m_2, e_2 \rangle \Rightarrow \langle s_1, m_1, e_1 \rangle + \langle \overline{s_2}, m_2, e_2 \rangle$$

Con  $\overline{s_2}$  intendiamo  $m_2$  va portata in complemento a 2 e poi sommata con la mantissa  $m_1$

## Moltiplicazione

$$\langle s_1, m_1, e_1 \rangle * \langle s_2, m_2, e_2 \rangle = \langle s, m, e \rangle$$

con

$$s = 0 \text{ se } s_1 = s_2$$

$$s = 1 \text{ altrimenti}$$

$m, e$  sono la mantissa e l'esponente normalizzati di

$$1, m_1 * 1, m_2 * b^{e_1 + e_2}$$


**Nota:** Si eseguono prendendo le mantisse fino agli ultimi 1 e senza andare ad apparare il numero di bit delle due mantisse

L'esponente si ricava facendo la somma (**NON** binaria) degli esponenti.

La mantissa si ricava facendo la moltiplicazione binaria delle mantisse. Per capire dove va messa la virgola bisogna sommare i numeri che ci sono dopo la virgola in  $m_1$  e  $m_2$  (**NON** in binario).

Facendo attenzione all'overflow degli esponenti

## Video utile

 IEEE Floating Point Addition & Multiplication

## Divisione

$$\langle s_1, m_1, e_1 \rangle \div \langle s_2, m_2, e_2 \rangle = \langle s, m, e \rangle$$

con

$$s = 0 \text{ se } s_1 = s_2$$

$$s = 1 \text{ altrimenti}$$

$m$ ,  $e$  sono la mantissa e l'esponente normalizzati di

$$(0, m_1 \div 0, m_2) * b^{e_1 - e_2}$$

## Tutte le operazioni

Il risultato deve essere normalizzato

- Portare il risultato in forma 1,...
- Regolare l'esponente di conseguenza
- Se la mantissa del risultato è negativa, complementarla e inserire la negazione nel primo bit della rappresentazione
- Il segno del risultato va valutato in base a segno e magnitudo delle mantisse degli operandi (quello da riportare in notazione IEEE 754)

## Overflow

L'overflow si verifica se l'esponente è troppo grande o troppo piccolo.

## Hex

Raggruppo i bit 4 a 4 e li trasformo in HEX

$$0100000010000000 = 0100\ 0000\ 0100\ 0000 = 4080$$

---



# Rappresentazioni

## BCD

Rappresentazione di un decimale cifra per cifra.

La rappresentazione non è efficiente perché sprechiamo rappresentazione (con 4 bit potrei rappresentare da 0 a 15) ma è una rappresentazione comoda per lavorare su singole cifre.

## Esempio

3827 = 0011 1000 0010 0111

## ASCII

È una rappresentazione composta da 7 bit.

- 3 bit di prefisso identificano il tipo (es.: 000 e 001 sono i caratteri speciali, 011 le cifre decimali, 100 e 101 le lettere maiuscole, etc.)
- 4 bit rappresentazione codificano il carattere in maniera monotona (se c'è un ordinamento naturale)

Ed è usata per rappresentare i simboli.

Si completa la rappresentazione con un ulteriore bit (all'inizio) e può essere usato come bit di parità (Parity Bit).

*Parity Bit* → Calcolato in base al numero di 1 nella sequenza di bit

Vale:

- 1 se il numero di 1 nella sequenza è pari
- 0 se il numero di 1 nella sequenza è dispari

Viene usato per trovare errori di trasmissione. Stabilito tra mittente e destinatario che PARITY BIT usare, il destinatario calcola il numero di 1 nella sequenza ricevuta e vede se l'informazione combacia con il PARITY BIT.

---

## Boole

Nella logica booleana i due simboli assumono il significato di vero (V) e falso (F)

George Boole dimostrò come la logica possa essere ridotta ad un sistema algebrico molto semplice, che utilizza solo un codice binario

Il codice binario fu trovato particolarmente utile nella teoria della commutazione (Claude Shannon) per descrivere il comportamento dei circuiti digitali (1=chiuso, 0=aperto).

## Espressione booleana

Un'espressione booleana è una sequenza composta da operatori booleani, parentesi, costanti e variabili booleane

## Funzione Booleana

È una legge che, in base ai valori delle variabili, restituisce in maniera univoca un valore booleano

## Tavola di verità

Una funzione booleana può essere rappresentata mediante una tavola di verità che descrive completamente l'associazione tra gli elementi del dominio e quelli del codominio.

Date  $n$  variabili, una tavola di verità è composta da 2 parti:

- Nella parte sinistra elenca ordinatamente tutte le  $2^n$  combinazioni possibili di valori binari assegnabili alle variabili
- Nella parte destra, contiene una colonna di 0 e 1 tale che il valore nella riga  $i$  sia il valore assunto dalla funzione in corrispondenza dell' $i$ -esima npla di valori booleani assegnati alle variabili

## Assiomi

Regola dello XOR	$A \oplus B = (A \cdot \bar{B}) + (\bar{A} \cdot B)$	$\overline{A \oplus B} = AB + \bar{A} \bar{B}$
Commutativa	$X + Y = Y + X$	$YX = XY$
Associativa	$X + (Y + Z) = (X + Y) + Z$	$X(YZ) = (XY)Z$
Distributiva	$X(Y + Z) = (XY) + (XZ)$	$X + (YZ) = (X + Y)(X + Z)$
Elemento neutro	$X + 0 = X$	$X * 1 = X$
Complemento	$X + \bar{X} = 1$	$X\bar{X} = 0$
Involuzione	$\overline{\bar{X}} = X$	
Elemento Annullatore	$X * 0 = 0$	

Idempotenza	$XX = X$	$X + X = x$
Assorbimento	$x + xy = x$	$x(x + y) = x$
Assorbimento 2	$x + \bar{x}y = x + y$	
De Morgan	$\overline{x + y} = \bar{x} \bar{y}$	$\overline{xy} = \bar{x} + \bar{y}$
Consenso	$xy + y\bar{z} + xz$	$y\bar{z} + xz$

## Uguaglianza

E1 ed E2 sono equivalenti se hanno lo stesso valore a fronte dello stesso assegnamento di valori booleani alle loro variabili.

Per verificare l'uguaglianza:

- tramite dimostrazioni formali
- Usiamo il metodo dell'induzione perfetta, cioè costruiamo la tavola di verità di entrambe le espressioni dimostrando che i valori sono uguali

## Dualità

Un'espressione  $E_1$  è la duale di un'altra  $E_2$  se differisce per gli operatori (in  $E_1$  compare +, in  $E_2$  compare \*) e per i valori delle costanti (in  $E_1$  compare 0, in  $E_2$ ).

## Complementare

È la funzione complementata quindi  $\bar{f}$  è la funzione complementata di  $f$ .

## SOP

Forma canonica disgiuntiva, composta da mintermini

Dalle SOP si sviluppa una rete AND-TO-OR

### Trasformare in canonica

Moltiplico per  $(x + \bar{x})$  i termini dove manca x

## POS

Forma canonica congiuntiva da maxtermini

Dalle POS si sviluppa una rete OR-TO-AND

## Trasformare in canonica

Sommo  $(x * \bar{x})$  ai termini dove manca  $x$

## Mintermine

È un termine prodotto in cui compaiono tutti i letterali

## Maxtermine

È un termine somma in cui compaiono tutti i letterali

## Forma Canonica SOP

Tutti i termini sono mintermini

## DA NORMALE A CANONICA

- Moltiplico il prodotto dove manca la variabile  $x$  per  $(x + \bar{x})$
- Applico la proprietà distributiva
- Elimino i termini ripetuti

## Forma Canonica POS

Tutti i termini sono maxtermini

## DA NORMALE A CANONICA

- Moltiplico il prodotto dove manca la variabile  $x$  per  $(x\bar{x})$
- Applico la proprietà distributiva
- Elimino i termini ripetuti

## SOP a POS con dualità

- Trovare la duale di  $f$
- Svolgere le moltiplicazioni
- Togliere i termini duplicati / Assorbimento
- Trovare la duale della nuova funzione

## Da SOP a POS con distributiva (3 mintermini)

$$xy + xz + ab = (x+a)(x+b)(y+z+a)(y+z+b)(x+y+a)(x+y+b)(x+z+a)(x+z+b)$$

**Nota:** il risultato di una trasformazione con  $n$  mintermini dovrà generare  $2^n$  fattori.

**Spiegazione:**

- sommo  $a$  e  $b$  ad  $x$  ( $x$  è il literal presente più di una volta)

$(x+a)(x+b)$   
sommo a e b a  $(y+z)$   
 $(y+z+a)(y+z+b)$   
sommo a e b a  $xy$   
 $(x+y+a)(x+y+b)$   
sommo a e b a  $xz$   
 $(x+z+a)(x+z+b)$

## Da POS a SOP con distributiva

- Eseguire le moltiplicazioni
- Applicare assorbimento / eliminare termini ripetuti

## Espressione Minimale

Espressione con il minimo numero di termini (somme nelle SOP, moltiplicazioni nelle POS) e a parità di termini minor numero di letterali.

## Minimizzazione

Il problema di ricavare una espressione minima deriva dalla necessità di ridurre il numero di porte logiche necessarie per realizzare una rete combinatoria.

Questo ha conseguenze in termini di:

- TEMPO DI ATTRAVERSAMENTO: Il tempo di risposta di una rete combinatoria dipende dal numero di porte logiche attraversate: ridurre tale numero può avere effetti importanti in termini di prestazioni

## Mappe di K

Modo di scrivere una Funzione Booleana diverso dalle tavole di verità

Queste mappe ordinano i punti dello spazio booleano  $\{0, 1\}^n$  in modo che i punti a distanza unitaria (cioè le cui coordinate differiscono per un solo bit) siano adiacenti sulla mappa. Funzionano solo fino a  $n = 4$  poi servono metodi più complessi

## Funzioni non completamente specificate

- Non tutte le combinazioni in ingresso sono ammissibili
- Non tutti i valori di uscita sono ammissibili
- Entrambi i casi precedenti
- Il valore della funzione è DON'T CARE ( $\delta$ )

I DON'T CARE non sono obbligatori da prendere nei raccoglimenti delle mappe di Karnaugh, vanno presi solo se possono servire a fare dei raggruppamenti più grandi e possono valere sia 0 che 1.

---

## Circuiti combinatori

Una rete combinatoria è un circuito elettronico digitale in grado di calcolare in modo automatico una funzione booleana.

Un circuito elettronico è un sistema costituito da blocchi elementari (porte) interconnesse tra loro in maniera aciclica.

Gli ingressi di porte che non sono uscite di altre porte sono gli input. Le uscite di porte che non sono ingressi di altre porte sono gli output

### Procedimento di Sintesi

- Descrizione Verbale
- Ricavare un'espressione booleana
- Tavola di verità
- Mappe di K per ottenere espressione minimale
- Disegno del circuito

### Procedimento di Analisi

- Circuito dato
- Espressioni booleane in uscita
- Tavola di verità
- Mappe di K per ottenere espressione minimale
- Verifica di ottimalità
- Descrizione verbale
  - Dipende dalle informazioni che ho, se so quello che rappresentano le variabili e cosa realizza quel circuito

### Schema circuitale

Uno schema circuitale è un collegamento di porte (rappresentate in maniera grafica) tramite linee. Identifichiamo tre tipi di linee:

1. linee di ingresso, ognuna etichettata con una delle  $n$  variabili booleane
2. linee di uscita, ognuna etichettata con una delle  $m$  variabili di uscita
3. linee interne, ciascuna delle quali collega l'uscita di una porta con l'ingresso di un'altra porta

## Porta Logica

Elemento di un circuito che presi dei valori binari in input restituisce dei valori binari in uscita

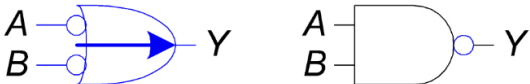
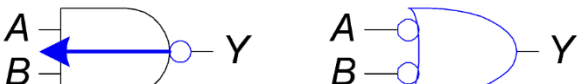
## Circuito Minimale

Circuito con il minor numero di porte e, a parità di porte, con il minor numero di fun-in (ingressi delle porte).

## Bubble pushing

Il Bubble Pushing è una tecnica per applicare il teorema di De Morgan direttamente al diagramma logico.

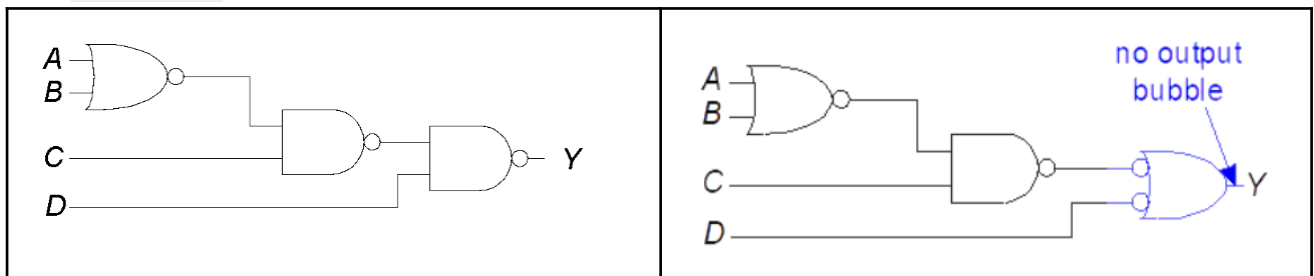
- Cambiare la porta logica (AND in OR e OR in AND).
- Aggiungi bolle agli input e agli output dove non ce n'erano e rimuovi le bolle originali.

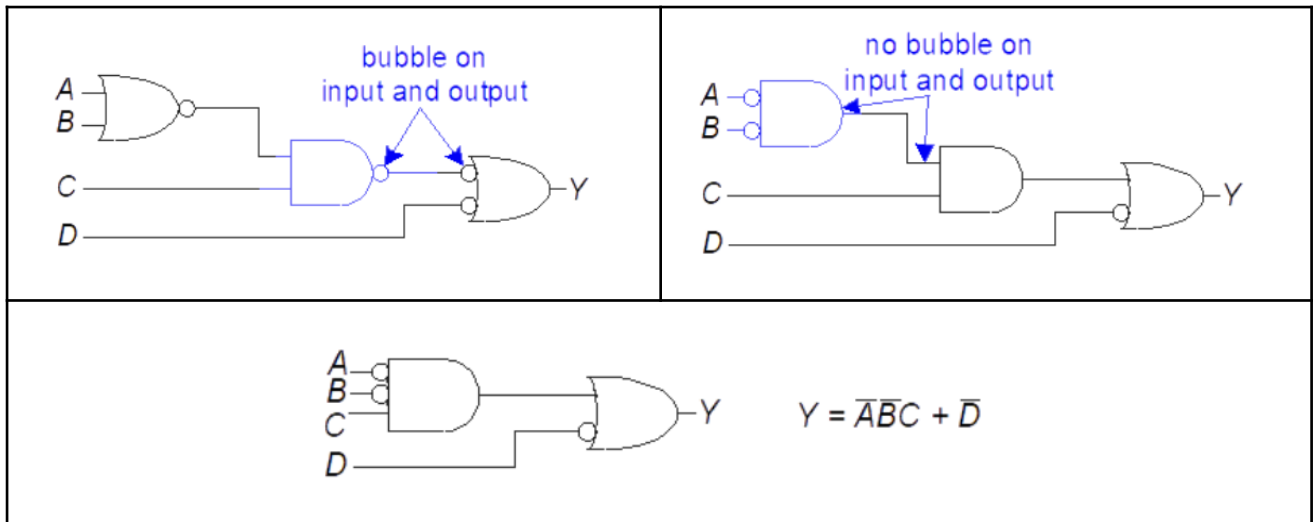
Forward	Backward
	

## Per applicarlo

- inizia dall'output, quindi lavora verso gli input;
- spingi indietro le bolle sull'output finale;
- disegna i gate in modo che le bolle si annullino.

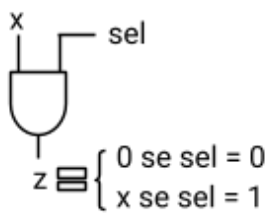
## Esempio





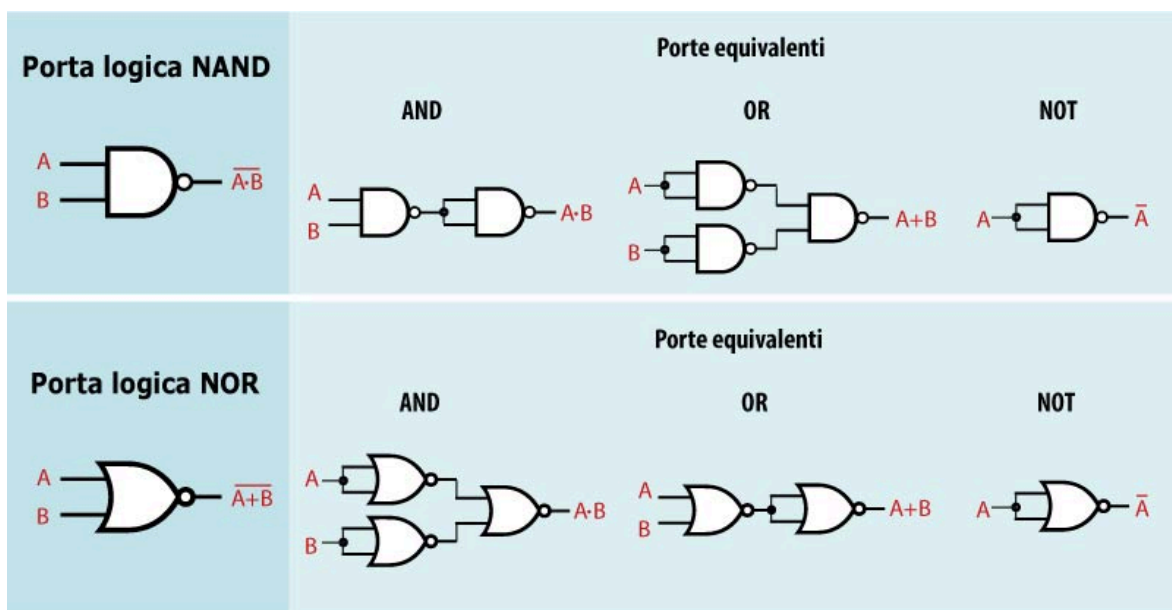
## Gating

### Circuito



Sono delle porte AND con funzione di controllo (gating) che sfruttano un segnale di selezione (sel) per scegliere che valore far entrare

## Circuiti All Nand - Nor





## Espressioni All Nand (SOP)

Prendere l'espressione forma SOP, complementare tutto due volte ed applicare De Morgan solo sugli operatori OR esterni, lasciare tutti i membri complementati senza sviluppare ulteriormente con De Morgan.

### Esempio

$$abc + \bar{a}\bar{c}d = \overline{\overline{abc}} \overline{\overline{\bar{a}\bar{c}d}} \Rightarrow \text{abbiamo un all - nand}$$

**Nota:**  $\bar{c}$  sarebbe  $\bar{c}\bar{c}$  come all - nand ma teoricamente le porte NOT sono ammesse!  
Chiedere al docente se sono ammesse o no

## Espressioni All Nor (POS)

Prendere l'espressione forma POS, complementare tutto due volte ed applicare De Morgan solo sugli operatori AND esterni, lasciare tutti i membri complementati senza sviluppare ulteriormente con De Morgan.

### Esempio

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

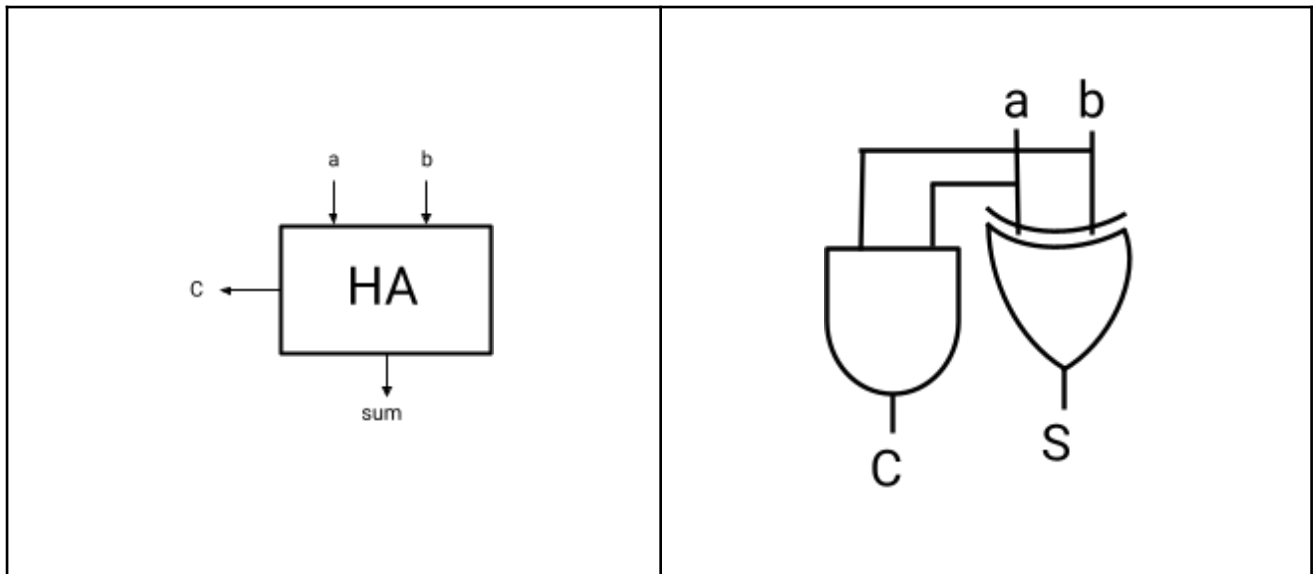
**Nota:**  $\bar{c}$  sarebbe  $\bar{c} + \bar{c}$  come all - nor ma teoricamente le porte NOT sono ammesse!  
Chiedere al docente se sono ammesse o no

---

## Progettazione Gerarchica / Modulare

### Half-Adder

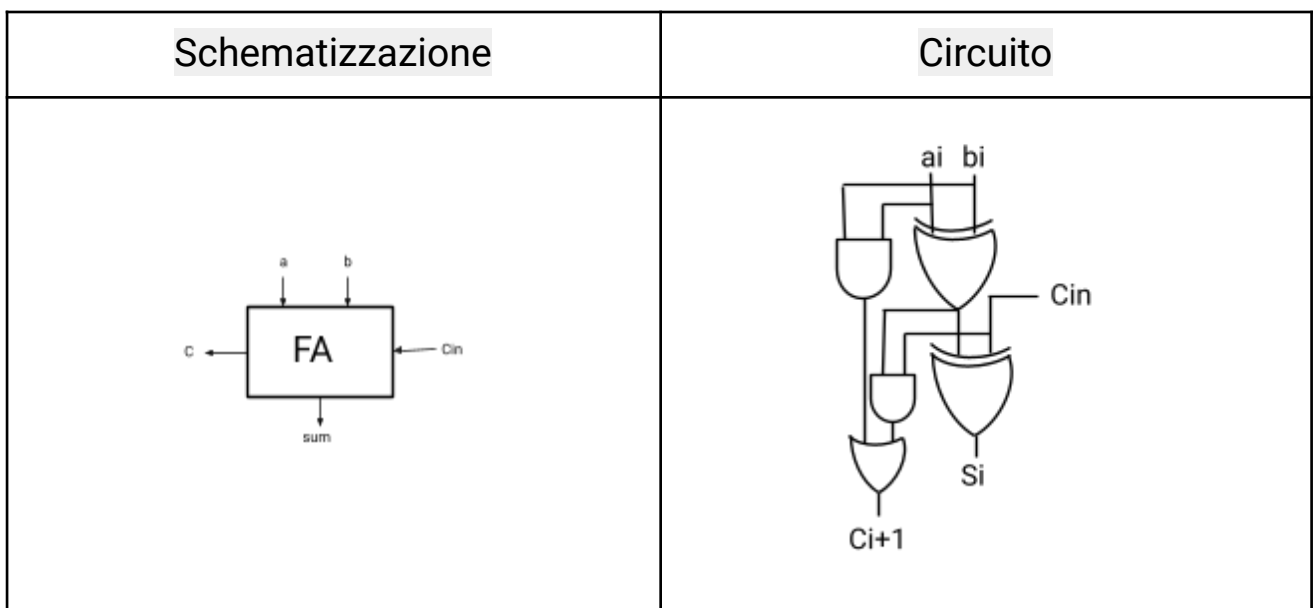
Schematizzazione	Circuito
------------------	----------



Somma due bit.

- $a, b$  sono i due bit in ingresso
- $sum$  è il valore della somma
- $C$  è il carry cioè il valore di riporto

## Full-Adder



Somma tre bit

- $a, b$  sono i due bit in ingresso
- $sum$  è il valore della somma
- $C$  è il carry cioè il valore di riporto
- $Cin$  è il carry-in cioè il valore di riporto da sommare insieme ai due bit

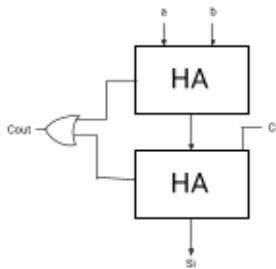
## ESPRESSIONI

$$sum = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i (a_i + b_i) c_i = a_i b_i (a_i \oplus b_i) c_i$$

Elimino il caso  $a, b = 1$  perché già previsto da  $a_i b_i$  e uso lo XOR della sum nel circuito

## FA come due HA



## Ripple-Carry-Adder

Responsabilità civile auto

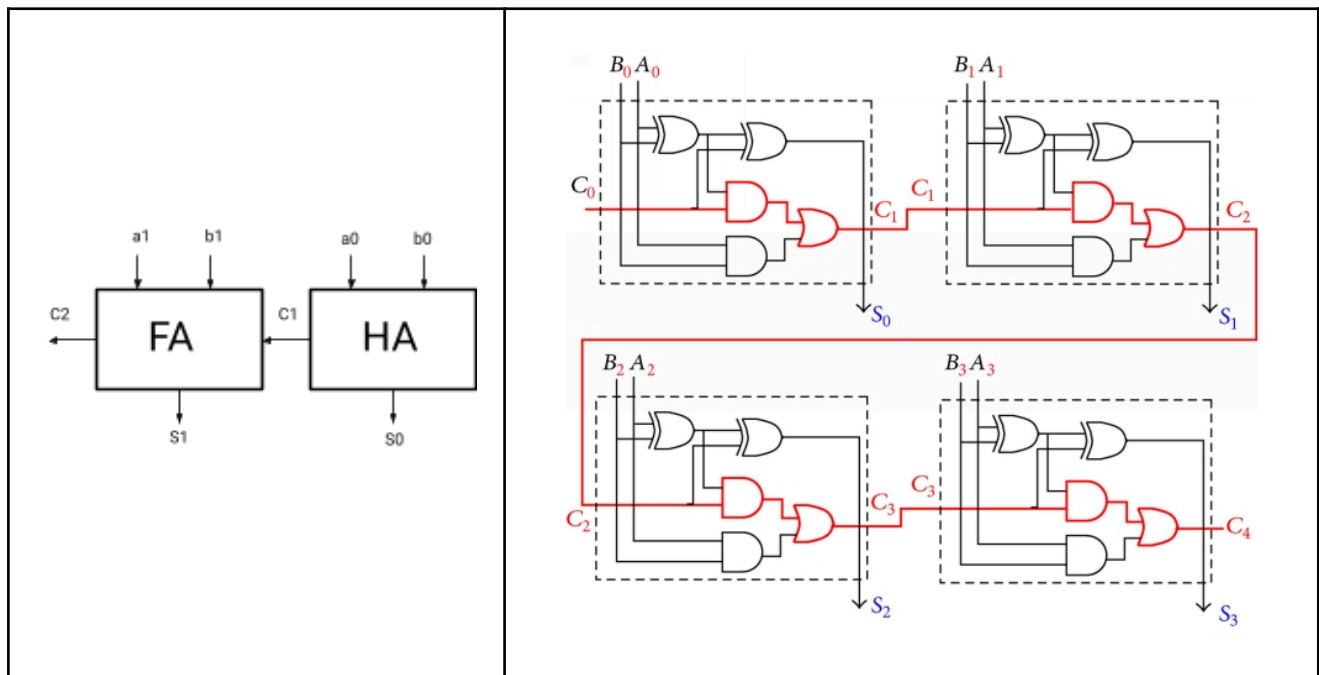
Un sommatore binario realizza la somma aritmetica fra due stringhe di  $n$  bit

$A = a_{n-1} \dots a_0$  e  $B = b_{n-1} \dots b_0$ , viste come numeri naturali.

L'Idea è effettuare la somma come siamo abituati:

- somma i bit meno significativi  $a_0$  e  $b_0$
- questo genera il bit meno significativo del risultato  $s_0$  ed un eventuale riporto  $c_1$
- procedi a sommare  $a_1, b_1$  e  $c_1$ ; questo genera  $s_1$  e  $c_2$
- ...e così via fino ai bit più significativi
- se l'ultima somma genera un riporto  $c$ , allora c'è overflow.

Schematizzazione	Circuito
------------------	----------



$A \ a_{n-1}, a_{n-2}, \dots, a_2, a_1, a_0$

$B \ b_{n-1}, b_{n-2}, \dots, b_2, b_1, b_0$

Il circuito continua con i FA fino a  $a_{n-1}$  e  $b_{n-1}$  e produce  $S_{n-1}$  e  $C_n$ .

**Nota:**  $C_n$  per i numeri binari rappresenta il bit di overflow

Il Ripple Carry Adder dà il risultato dopo  $m$  cicli quindi ha come svantaggio questa sua lentezza

Infatti solo un Full-Adder alla volta dà il risultato corretto, abbiamo quindi bisogno dell'uso di un temporizzatore

## Complemento a 2

Esempio facciamo

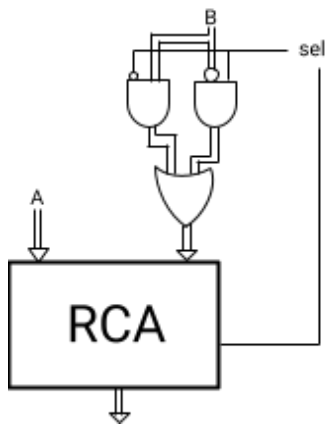
$$A - B = A + (-B) = A + (\overline{B} + 1) \Rightarrow \text{Complemento e aggiungo 1}$$

Si sostituisce il primo HA con un FA a cui entra  $C_{in} = 1$  e  $B = \overline{B}$

Usiamo quindi un segnale  $sel$  usato per scegliere tra somma e differenza

$sel = 0 \Rightarrow \text{somma}, sel = 1 \Rightarrow \text{differenza}$

## RCA con Gating



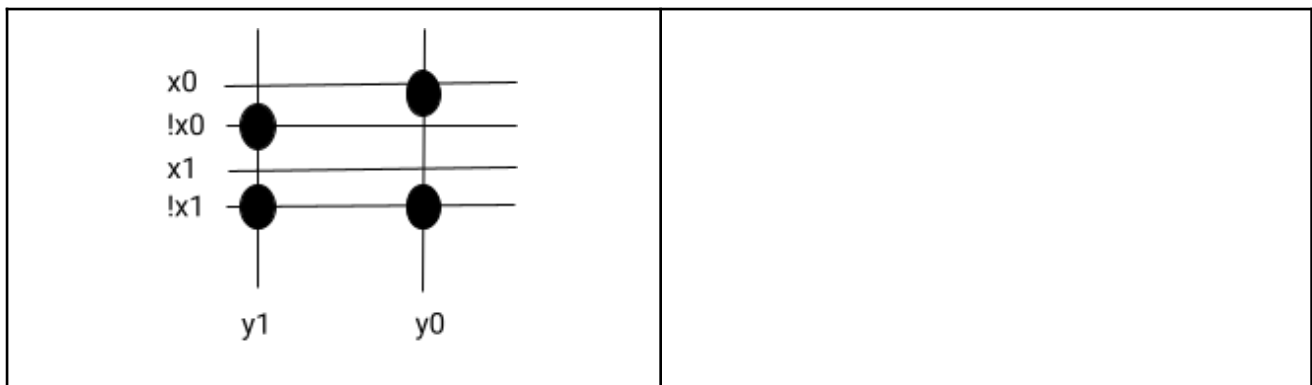
Questo è per far entrare  $B$  come numeri negativi.

**Nota:** Le frecce doppie servono per indicare dei fasci di bit, cioè che entra  $a_0, a_1, \dots, a_n$ ,  $b_0, b_1, \dots, b_n$

## Decoder

Solo un uscita vale uno, quella associata alla combinazione in input. Produce mintermini

Schematizzazione	Circuito
Come matrice di AND	Non standard
Il decoder può essere anche rappresentato come una matrice AND	Rimane una normalissima matrice di AND

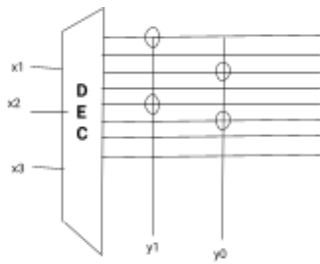


## Encoder

Solo uno degli ingressi è uguale a 1, le uscite sono la codifica dell'ingresso

Schematizzazione	Circuito
Come matrice di OR	
<p>L'encoder può essere anche rappresentato come una matrice OR</p>	

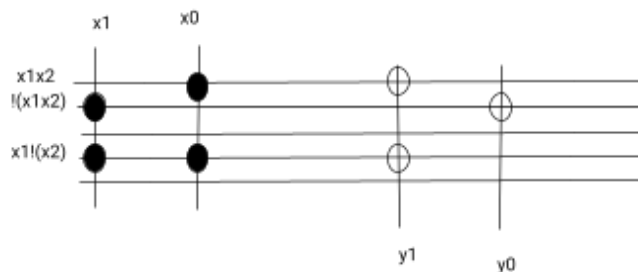
## Read Only Memory



È un decodificatore (che produce mintermini, cioè una matrice di AND) con una matrice di OR (cioè un codificatore) a cascata.

Posso realizzare funzioni in forma canonica

## Programmable Logic Array

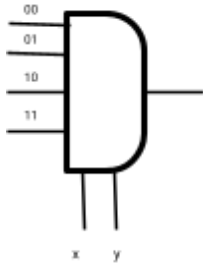
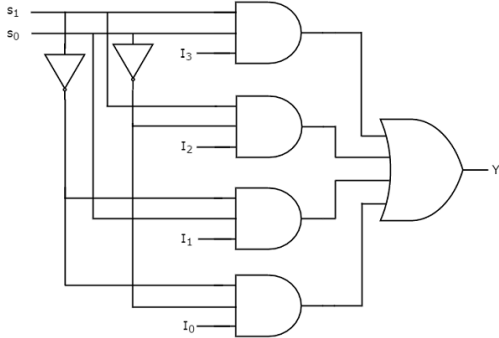


Matrice di AND che realizza i termini prodotto che compaiono nelle espressioni minimali

Matrice di OR per sommare i termini prodotto

Posso realizzare funzioni in forma minima (se voglio anche canonica ma si usa per la minima)

## Multiplexer

Schematizzazione	Circuito
<p>In figura è riportato un MUX 4-a-2.</p> 	

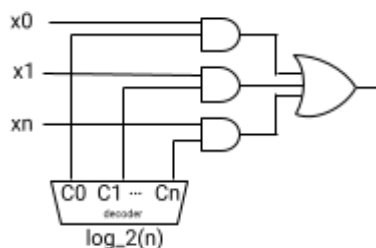
Seleziona un ingresso tra quelli possibili in base ai valori dei segnali di controllo, e trasferisce il suo valore sulla linea di uscita

Esistono vari MUX (2-a-1, 4-a-2, 8-a-4) che seguono la regola di avere  $n$  ingressi dati e  $m$  segnali di selezione

Esiste quindi il rapporto  $n = 2^m$  che va rispettato

Se all'ingresso c'è un decoder dispone di  $\log_2 n$  linee di selezione

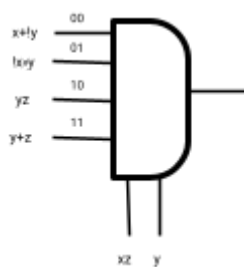
### Circuito con Decoder



### Da MUX a espressione logica

Per ricavare da un MUX l'espressione booleana associata basta creare tutte le possibili combinazioni tra entrate ed uscite del mux come prodotti e sommarli tra loro.

Per esempio

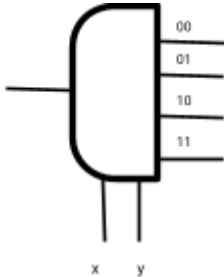
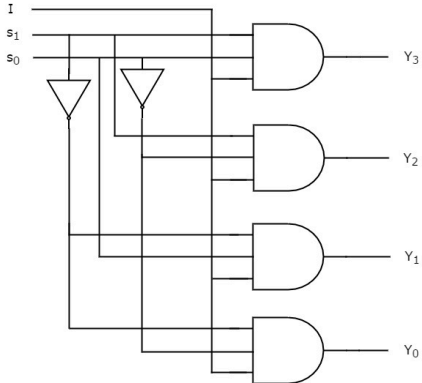


$$y = \overline{xz} \overline{y}(x + \overline{y}) + \overline{xz} y(\overline{x} \oplus y) + xzy\overline{y}(yz) + xzy(y + z)$$

Infatti creiamo le combinazioni  $00(\overline{xz} \overline{y})$ ,  $01(\overline{xz} y)$ ,  $10(xz\overline{y})$ ,  $11(xzy)$  e le mettiamo in AND con le relative uscite del MUX.



## Demultiplexer

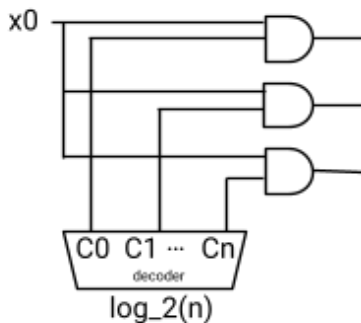
Schematizzazione	Circuito
	

È un dispositivo che accetta una singola linea di ingresso e la indirizza a una delle numerose linee di uscita digitale.

Un demux di  $2^n$  uscite dispone di  $n$  linee di selezione, che vengono utilizzate per selezionare a quale linea di uscita inviare l'ingresso.

Se all'ingresso c'è un decoder dispone di  $\log_2 n$  linee di selezione

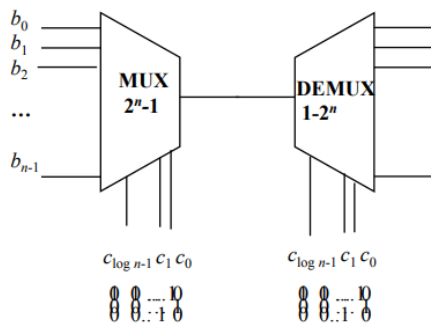
### Circuito con Decodificatore



## Mux e Demux

Usati se non posso mandare dati in parallelo, creo quindi un convertitore Parallelo - Seriale - Parallelo.

Collego quindi un Mux con un Demux che assocerà all'uscita quello che arriva in ingresso, associando primo valore alla prima uscita e così via



## Transcodificatore

### BCD a Sette Segmenti

Con 4 bit in BCD rappresentiamo i valori da 0 a 9

TABELLA

$x_3 x_2 x_1 x_0$	abcdefg
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111
1001	1111011

### 0,9 a Sette Segmenti

Posso usarlo per esempio per convertire valori da 0 a 9 in codice a 7 segmenti

usando 5 bit (il 5<sup>o</sup> è di parità)

0 fa eccezione

TABELLA

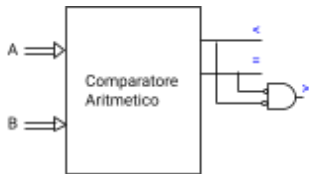
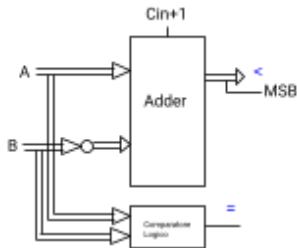
$y_4 y_3 y_2 y_1 y_0$	abcdefg
00110	1111110
10001	0110000
01001	1101101
11000	1111001
00101	0110011
10100	1011011
01100	1011111
00011	1110000
10010	1111111
01010	1111011

## Comparatore Aritmetico

Serve a capire se  $A > B$  quindi se  $A - B > 0$

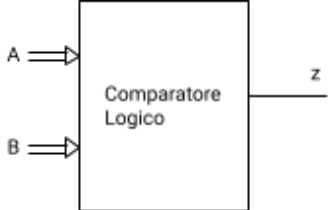
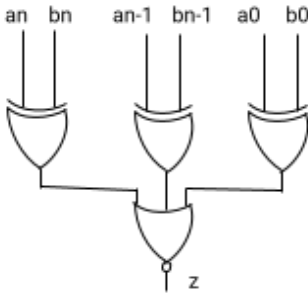
Chiaramente assumiamo di usare  $Ca_2$  e quindi andremo a valutare il primo bit

Per capire il segno del risultato mi basterà un addizionatore

Schematizzazione	Circuito
	

## Comparatore Logico

Serve a vedere se due sequenze di bit sono uguali

Schematizzazione	Circuito
	

$A \ a_n, a_{n-1}, a_1, a_0$

$B \ b_n, b_{n-1}, b_1, b_0$

Voglio comparare tutte le coppie (bit nella stessa posizione) di  $A$ ,  $B$ , e voglio come risultato ( $z$ ):

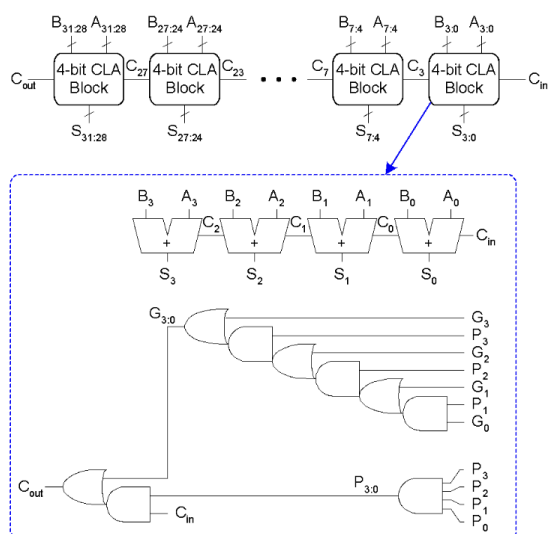
$z = 1$  se  $A = B$

$z = 0$  altrimenti

**Nota:** Potremmo usare anche degli xnor ed un and al posto del nor

## Carry-Lookahead adder

Calcola  $C_{out}$  per un blocco di k-bit usando due segnali intermedi: propagate e generate.



La colonna  $i$  produce un riporto generando un carry out o propagando un carry nel carry out.

### Segnali all'interno dei blocchi di k-bit

- **generate:** La colonna  $i$  genererà un risultato se  $A_i$  e  $B_i$  sono entrambi 1.
  - $G_i = A_i B_i$
- **propagate:** La colonna  $i$  propagherà un riporto al riporto se  $A_i$  o  $B_i$  è 1.
  - $P_i = A_i + B_i$
- **carry-out:** Il carry-out della colonna  $i$  ( $C_i$ ) è:
  - $C_i = A_i B_i + (A_i + B_i)C_{i-1} = G_i + P_i C_{i-1}$

### Segnali tra i blocchi di k-bit

$j \rightarrow$  blocco più significativo,  $i \rightarrow$  blocco meno significativo

- **propagate:**  $P_{i:j} = P_i P_{i-1} P_{i-2} \dots P_j$
- **generate:**  $G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} G_j))$ 
  - il segnale di propagate o viene generato o propagato dal blocco precedente, all'inizio deve essere necessariamente generato dal primo blocco;
- **carry-out:**  $C_i = G_{i:j} + P_{i:j} C_{i-1}$

### Carry-Lookahead Adder Delay

Il percorso critico di un carry-lookahead adder è quello che va dal  $C_{in}$  al  $C_{out}$  (anche se dal circuito potrebbe sembrare che sia quello del segnale di generate) perchè quest'ultimo segnale si deve trasmettere attraverso tutti i blocchi, a differenza del generate che viene calcolato in parallelo tra tutti i blocchi. Quindi il percorso critico è ridotto a  $t_{and-or}$

Delay per un N-bit CLA con blocchi di k-bit:

$$t_{CLA} = t_{pg} + t_{pg-clock} + (N/k - 1)t_{and-or} + kt_{FA}$$

- $t_{pg}$ : ritardo per calcolare tutti i  $G_i$  e  $P_i$
- $t_{pg-clock}$ : ritardo per calcolare tutti i  $G_{i:j}$  e  $P_{i:j}$

Un sommatore carry-lookahead di N bit è generalmente molto più veloce di un sommatore ripple-carry per **N > 16**

## Prefix-Adder

Calcola il carry-in ( $C_{i-1}$ ) per ogni colonna, quindi calcola la somma:

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$

Calcola G e P per blocchi da 1, 2, 4, 8 bit, ecc. Fino a quando tutto  $G_i$  (carry in) è noto.

Quindi utilizza  $\log_2 N$  fasi.

I carry vengono generati in una colonna o propagati da una colonna precedente.

## Equazioni

Si considera una colonna -1 che contiene il  $C_{in}$  come segnale di generate, quindi:

$$G_{-1} = C_{in} \quad , \quad P_{-1} = 0$$

il carry-in della colonna i è uguale al carry-out dalla colonna i-1:

$$C_{i-1} = G_{i-1:-1}$$

Quindi l'equazione della somma diventa:

$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

Adesso dovrà calcolare i segnali G e P per i blocchi di bit che vanno da i a j (2,4,8 bit):

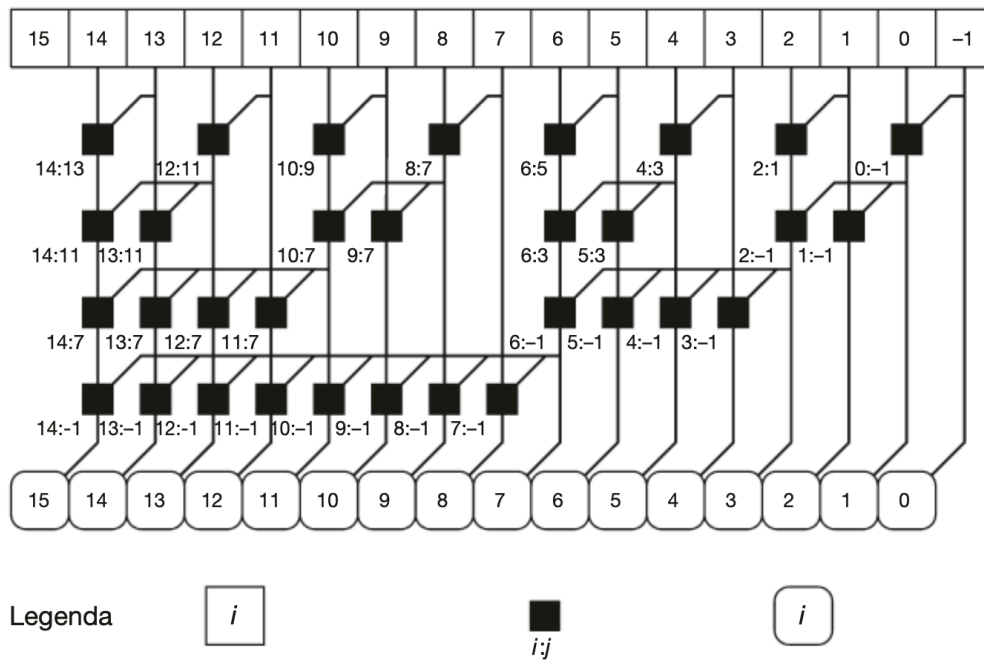
**Segnale di Generate:**  $G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$

- il blocco i:j genererà un carry se:
  - la parte superiore (i:k) genera un riporto;
  - la parte superiore (i:k) propaga un riporto generato nella parte inferiore (k-1:j);

**Segnale di Propagate:**  $P_{i:j} = P_{i:k} P_{k-1:j}$

- il blocco i:j propagherà un riporto se sia la parte superiore che quella inferiore propagano il riporto

## Schema



Segnali di P e G partendo dagli input	Segnali di P e G partendo da blocchi di $i:k$ e arrivando a $i:j$	Formula di $S_i$
<p><math>A_i B_i</math></p> <p><math>P_{i:i} \quad G_{i:i}</math></p>	<p><math>P_{i:k} P_{k-1:j} G_{i:k} G_{k-1:j}</math></p> <p><math>P_{i:j} \quad G_{i:j}</math></p>	<p><math>G_{i-1:-1} A_i B_i</math></p> <p><math>S_i</math></p>

## Delay

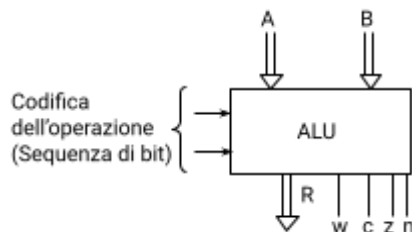
$$t_{PA} = t_{pg} + \log_2 N(t_{pg-prefix}) + t_{XOR}$$

- $t_{pg}$ : delay per produrre  $P_i, G_i$  (AND or OR gate)
- $t_{pg-prefix}$ : delay della cella con prefisso nero (AND-OR gate)

## ALU (Semplice/Banale)

Modulo che esegue operazioni logiche (AND, OR, ecc.) e aritmetiche (+, -, \*, ecc.) tra due o più bit

### Schematizzazione



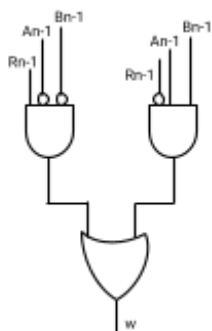
Bit del condition code:

- $w = \text{overflow}$
- $c = \text{riporto in uscita}$
- $z = \text{risultato è } 0$
- $n = \text{risultato è negativo}$

### Bit del condition code

$w \Rightarrow$  Confronto i bit più significativi di  $R$ ,  $A$ ,  $B$

Se i due operandi sono positivi e il risultato è negativo o se i due operandi sono negativi e il risultato è positivo, allora  $w = 1$



$c \Rightarrow$  Serve per sapere il riporto in caso dovessi spezzare le operazioni in più parti

Se ad esempio il mio adder è a 32 bit e devo fare una somma a 64 bit, ne elaboro 32 alla volta, e al secondo blocco passo il valori di  $c$

$z \Rightarrow$  Un comparatore logico compara il risultato con una stringa di 0

$n \Rightarrow$  Vedo il bit più significativo di  $R$  (che è in  $Ca_2$  quindi vedo se è  $= 1$ )

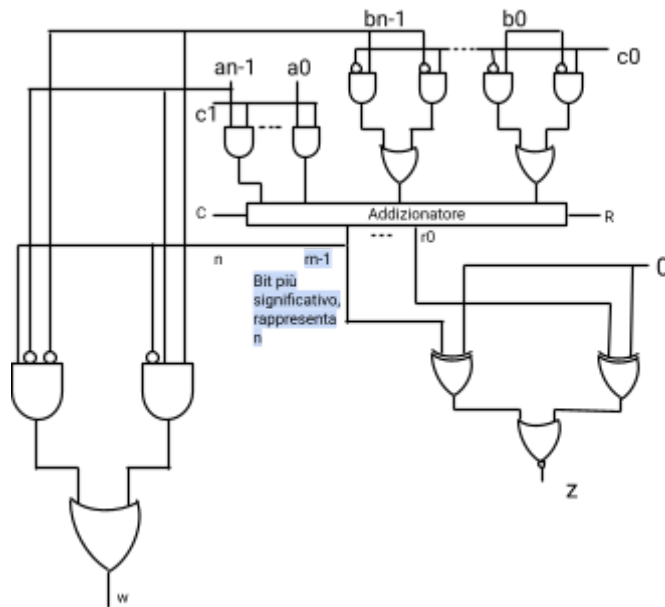


## Circuito

$c_0 = 0$  pone l'operando a zero

$c_1 = 1$  complementazione logica di  $b$

$r = 1$  incrementa di 1



## Circuiti sequenziali

Sono come i circuiti combinatori ma contengono elementi di memoria e viene introdotto il concetto di tempo, per capire i valori da mantenere

Sono quindi composti da:

- Elementi combinatori come moduli standard e porte logiche
- Un insieme di FF che mantengono i valori

Le funzioni fondamentali di una rete sequenziale sono:

- memorizzazione di valori booleani (stato)
- modifica dei valori memorizzati in base ai segnali in ingresso (transizioni di stato)

I FF sono le reti sequenziali più semplici, e già esibiscono le funzioni fondamentali

Per descrivere il comportamento dei FF abbiamo usato delle Tavole di Verità "estese", in cui abbiamo introdotto il fattore tempo ( $y$  e  $Y$ )

$s$	$r$	$y$	$Y$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

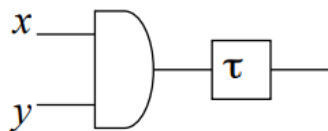
Per ogni riga della tabella ho una freccia dallo stato associato al valore di  $y$  a quello di  $Y$  etichettato con la sequenza di valori in ingresso

Finora abbiamo solo considerato circuiti aciclici. Questo perché abbiamo implicitamente assunto che le porte siano ideali, nel senso che hanno un tempo di attraversamento nullo.

Quindi non avrebbe senso:



In realtà, le porte hanno un tempo di attraversamento, tipicamente modellato avendo una porta ideale (ad attraversam. nullo) e in serie un ritardo  $\tau$ :



Questo introduce il fattore tempo nei circuiti, che pertanto verranno chiamati reti sequenziali.

## Analisi

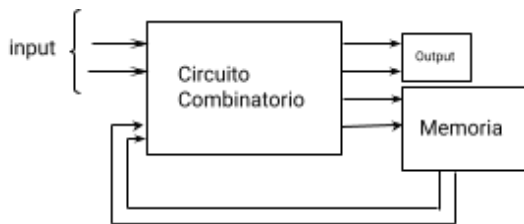
- Espressioni booleane delle equazioni di eccitazione (ingressi FF) e delle uscite
- Tavole degli stati futuri
  - Ingressi
  - Valori delle funzioni di eccitazione
  - Output
  - Stati attuali
  - Stati futuri
- Diagramma della nostra rete sequenziale (Automa a stati finiti) o FSM

## Memorizzazione e feedback

Sono delle reti in cui sono presenti elementi di memorizzazione

Per la memoria si usano:

- Porte logiche già definite
- Linee di feedback, cioè l'uscita di una porta alimenta una o più porte che hanno prodotto quell'uscita



## Latch SR

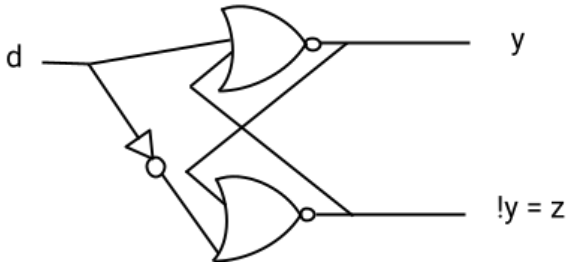
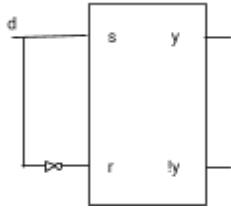
È un circuito di base per la memorizzazione di un bit

Circuito	Schematizzazione
Funzionalità	Stato del latch $[0, 1]$  $y(t) = \text{Stato al tempo } t$ $y(t + 1) = \text{Stato al tempo } t + 1 = Y(\text{stato } t)$
<ul style="list-style-type: none"> <li>- Memorizzazione <math>s = 0, r = 0</math></li> <li>- Set <math>s = 1, r = 0</math></li> <li>- Reset <math>s = 0, r = 1</math></li> </ul>	

## Latch D

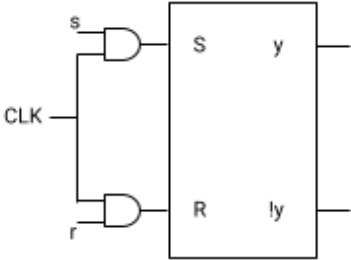
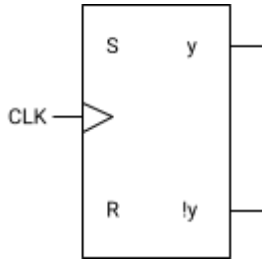
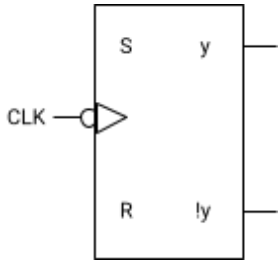
È il sottoinsieme  $[0, 1][1, 0]$  del latch  $sr$

Il segnale in output è uguale al segnale in input

Circuito	Schematizzazione
	

## Latch sincrono

Chiamato anche Gate Latch rende il cambiamento non immediato usando il segnale di *clock*

Circuito	Schematizzazione
	 <p>Questo è sensibile al fronte d'onda</p>  <p>Questo è sensibile al fronte di discesa</p>

## Preset e Reset

I Flip-Flop hanno degli ingressi asincroni chiamati

- Pre-Set
- Clear

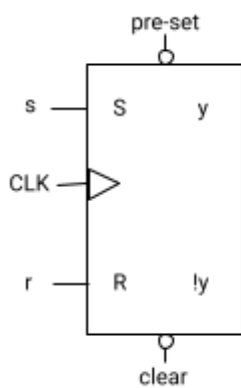
Questi segnali sono asincroni perché non filtrati dal clock e quindi agiscono immediatamente sullo stato del FF

Sono segnali che arriva ai FF negati

Funzionamento:

- PRESET = CLEAR = 0: normale funzionamento del FF
- PRESET = 1, CLEAR = 0: set immediato del FF
- PRESET = 0, CLEAR = 1: reset immediato del FF
- PRESET = CLEAR = 1: non usata

## Schematizzazione



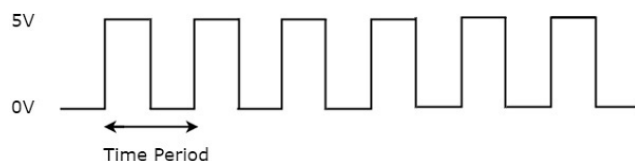
## Enable

Segnale di controllo utilizzato per vedere tutte le operazioni dei FF sul registro

## Clock

Il *clock* è un segnale usato come segnale di controllo, è un segnale periodico ed ha un fronte d'onda ed uno di discesa

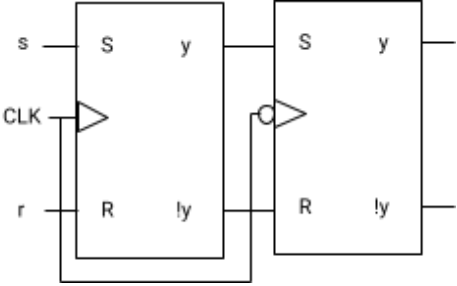
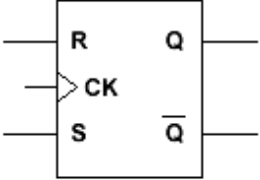
## Segnale



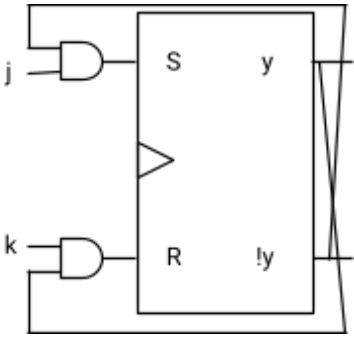
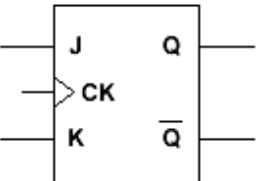
## Flip Flop SR

Chiamato anche Latch Master-Slave.

Trasferisco il valore prodotto sul primo latch al secondo

Circuito	Schematizzazione																				
																					
Tavola degli stati futuri	Tavola inversa																				
<table border="1"> <thead> <tr> <th>SR</th><th>Y</th></tr> </thead> <tbody> <tr> <td>00</td><td>y</td></tr> <tr> <td>01</td><td>0</td></tr> <tr> <td>10</td><td>1</td></tr> <tr> <td>11</td><td>-</td></tr> </tbody> </table>	SR	Y	00	y	01	0	10	1	11	-	<table border="1"> <thead> <tr> <th>yY</th><th>SR</th></tr> </thead> <tbody> <tr> <td>00</td><td>0δ</td></tr> <tr> <td>01</td><td>10</td></tr> <tr> <td>10</td><td>01</td></tr> <tr> <td>11</td><td>δ0</td></tr> </tbody> </table>	yY	SR	00	0δ	01	10	10	01	11	δ0
SR	Y																				
00	y																				
01	0																				
10	1																				
11	-																				
yY	SR																				
00	0δ																				
01	10																				
10	01																				
11	δ0																				

## Flip Flop JK

Circuito	Schematizzazione
	
Tavola degli stati futuri	Tavola inversa

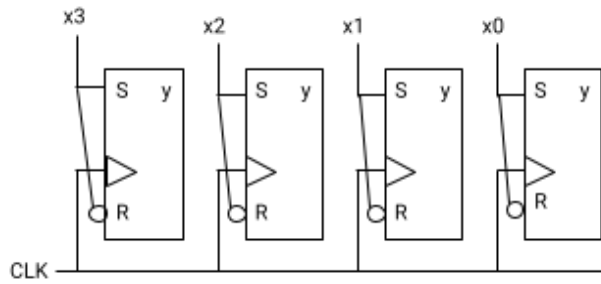
<table> <tr> <th>JK</th><th>Y</th></tr> <tr> <td>00</td><td>y</td></tr> <tr> <td>01</td><td>0</td></tr> <tr> <td>10</td><td>1</td></tr> <tr> <td>11</td><td><math>\overline{y}</math></td></tr> </table>	JK	Y	00	y	01	0	10	1	11	$\overline{y}$	<table> <tr> <th>yY</th><th>JK</th></tr> <tr> <td>00</td><td>0δ</td></tr> <tr> <td>01</td><td>1δ</td></tr> <tr> <td>10</td><td>δ1</td></tr> <tr> <td>11</td><td>δ0</td></tr> </table>	yY	JK	00	0δ	01	1δ	10	δ1	11	δ0
JK	Y																				
00	y																				
01	0																				
10	1																				
11	$\overline{y}$																				
yY	JK																				
00	0δ																				
01	1δ																				
10	δ1																				
11	δ0																				

# Flip Flop D

Circuito	Schematizzazione																
Tavola degli stati futuri	Tavola inversa																
<table> <tr> <th>D</th><th>Y</th></tr> <tr> <td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td></tr> </table>	D	Y	0	0	1	1	<table> <tr> <th>yY</th><th>D</th></tr> <tr> <td>00</td><td>0</td></tr> <tr> <td>01</td><td>1</td></tr> <tr> <td>10</td><td>0</td></tr> <tr> <td>11</td><td>1</td></tr> </table>	yY	D	00	0	01	1	10	0	11	1
D	Y																
0	0																
1	1																
yY	D																
00	0																
01	1																
10	0																
11	1																

# Come Latch SR

I FF D possono essere realizzati a partire dal Latch SR



Poiché qui ci limitiamo ai casi

SR	Y
00	y
01	0
10	1
11	-

e dunque abbiamo un FF D, infatti se  $x_n$  vale 0 uscirà 0, altrimenti 1

## Flip Flop T

Circuito	Schematizzazione														
Tavola degli stati futuri	Tavola inversa														
<table> <tr> <th>T</th><th>Y</th></tr> <tr> <td>0</td><td>y</td></tr> <tr> <td>1</td><td><math>\bar{y}</math></td></tr> </table>	T	Y	0	y	1	$\bar{y}$	<table> <tr> <th>yY</th><th>T</th></tr> <tr> <td>00</td><td>0</td></tr> <tr> <td>01</td><td>1</td></tr> <tr> <td>10</td><td>1</td></tr> </table>	yY	T	00	0	01	1	10	1
T	Y														
0	y														
1	$\bar{y}$														
yY	T														
00	0														
01	1														
10	1														



	<table border="1"> <tr> <td>11</td><td>0</td></tr> </table>	11	0
11	0		

## Counters

Sono dei registri sincroni usati per contare il numero di occorrenze di un determinato evento, sempre modulo un certo numero naturale.

In generale un contatore modulo  $2^n$  è composto da  $n$  FF

Tipicamente, gli eventi che può contare sono i colpi di clock o le occorrenze di alcuni valori (o sequenze) di input.

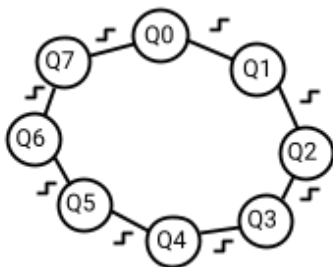
Si distinguono in:

- sincroni (tutti i FF del contatore hanno lo stesso clock)
- asincroni (nello stesso contatore i vari FF hanno clock diversi)

Gli stati vengono letti da destra a sinistra (dal più significativo al meno)

### Contatore modulo 8

Conta da 0 a 7 ed ha 3 *bit*, caso particolare contatori  $2^n$



In questo diagramma di Moore il segno di transizione rappresenta il fronte d'onda del clock.

Il numero dello stato invece il valore dell'uscita.

Questo contatore dunque aumenta di 1 ad ogni segnale di clock.

Tavola degli stati futuri

$y_2$	$y_1$	$y_0$	$Y_2$	$Y_1$	$Y_0$	$J_2$	$K_2$	$J_1$	$K_1$	$J_0$	$K_0$
0	0	0	0	0	1	0	$\delta$	0	$\delta$	1	$\delta$
0	0	1	0	1	0	0	$\delta$	1	$\delta$	$\delta$	1

0	1	0	0	1	1	0	δ	δ	0	1	δ
0	1	1	1	0	0	1	δ	δ	1	δ	1
1	0	0	1	0	1	δ	0	0	δ	1	δ
1	0	1	1	1	0	δ	0	1	δ	δ	1
1	1	0	1	1	1	δ	0	δ	0	1	δ
1	1	1	0	0	0	δ	1	δ	1	δ	1

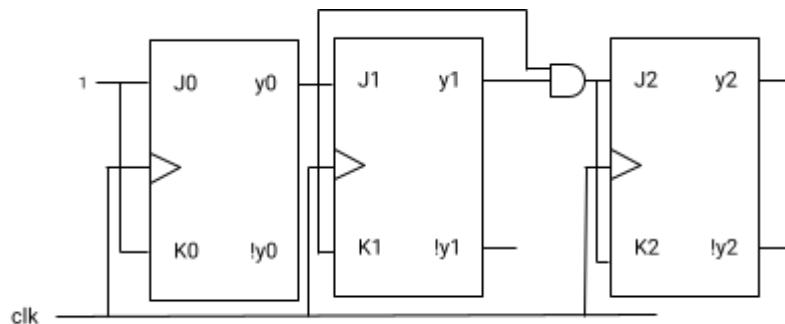
## ESPRESSIONI

$$J_0 = K_0 = 1$$

$$J_1 = K_1 = y_0$$

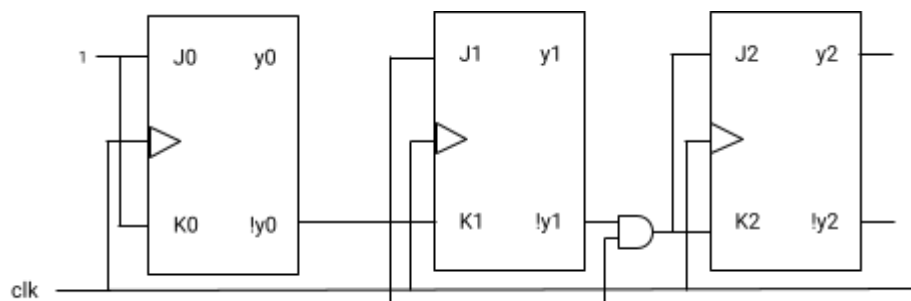
$$J_2 = K_2 = y_1 y_0$$

## CIRCUITO



## Contatore alla rovescia (Down Counter)

- Prendiamo il valore dello stato complementato
- Lo mandiamo a entrambi gli ingressi del ff successivo
- Poi procediamo normalmente ma mandando il valore in AND con lo stato negato del successivo

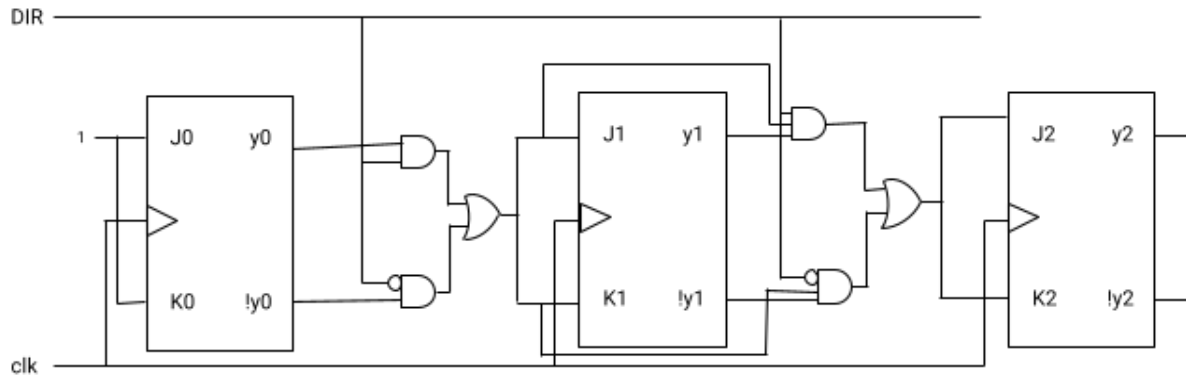


## Contatore Bidirezionale (Up Down Counter)

Dunque con un segnale di controllo possiamo decidere se creare un Contatore in avanti o indietro.

Quindi avrò un contatore e un segnale *dir*:

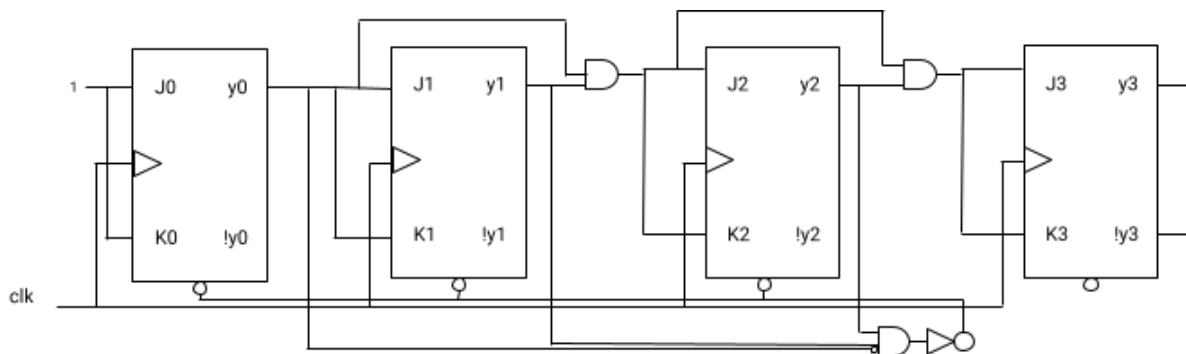
- $dir = 1 \Rightarrow \text{conto avanti}$
- $dir = 0 \Rightarrow \text{conto indietro}$



## Contatore con modulo $m \neq 2^n$

Devo realizzare un progetto ad HOC, realizzo quindi dei circuiti sequenziali ad hoc e dei circuiti combinatori ad hoc per manipolare ingressi e uscite

Ad esempio riportiamo un contatore modulo 6



Questo contatore conta da 000 a 101, appena riconosce la sequenza 110 attiva il segnale asincrono CLEAR che in maniera quasi immediata (c'è un piccolissimo ritardo in cui si legge la sequenza 110) resetta i FF

## Contatore Pre-Selezione

Il preset è un segnale usato per impostare i valori dei FF in modo asincrono

## Registri

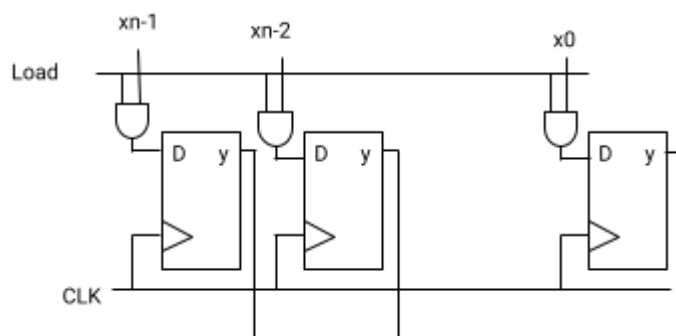
Usati per memorizzare informazioni e sono formati da un insieme di  $n$  FF, quindi possono memorizzare ad esempio parole lunghe  $n$

Eseguono operazioni di

- Memorizzazione, cioè mantengono l'informazione
- Caricamento con un valore o un'informazione
  - Parallelo
  - Seriale

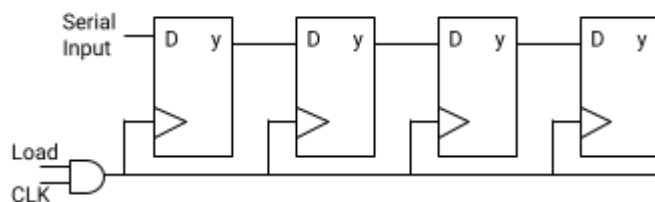
## Caricamento Parallelo

Abbiamo dei FF sincronizzati da un segnale di clock, il caricamento è eseguito tramite un segnale di LOAD che arriva a delle porte AND



## Caricamento Seriale

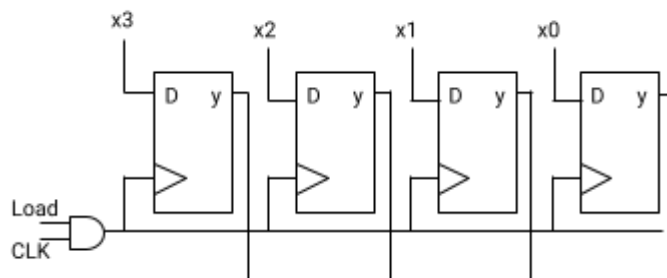
Il segnale LOAD qui è unito al clock



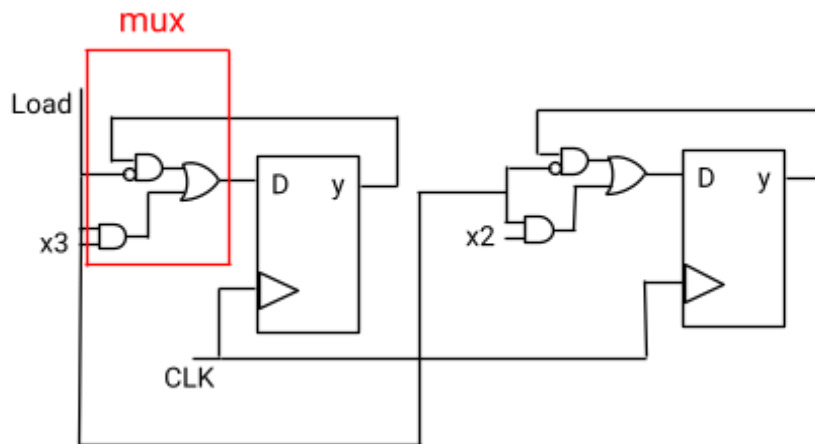
## Caricamento Parallelo (FF D)

Riceve  $n$  linee dati, i dati vengono memorizzati quando il segnale

$load = 1, clock = 1$

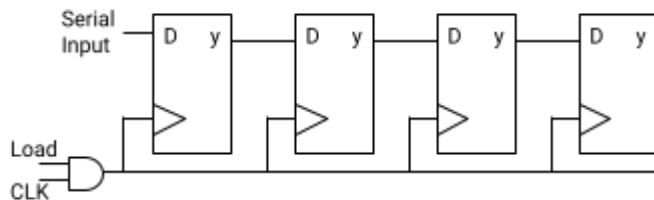


CARICAMENTO / MEMORIZZAZIONE



Con il segnale LOAD scelgo se mantenere o caricare le informazioni

### Caricamento Seriale (FF D)



Se voglio caricare la sequenza 1011:

$input \Rightarrow | * | * | * | * | \Rightarrow output$

$t_0 \Rightarrow | 1 | * | * | * |$

$t_1 \Rightarrow | 1 | 1 | * | * |$

$t_2 \Rightarrow | 0 | 1 | 1 | * |$

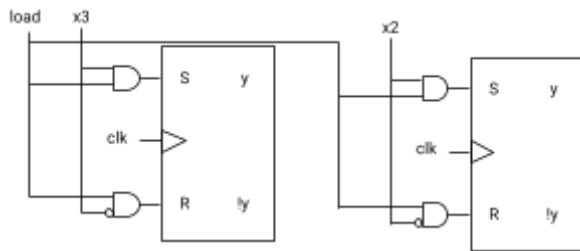
$t_3 \Rightarrow | 1 | 0 | 1 | 1 |$

Dopo 4 colpi di clock fermo il caricamento

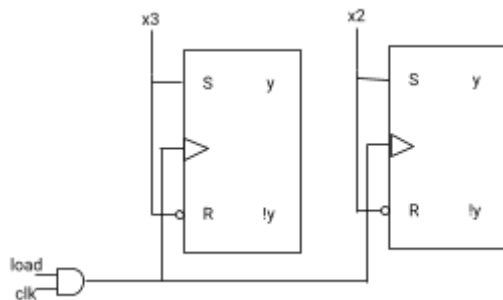
### Caricamento Parallelo (FF SR)

LOAD

$load = 1$  caricamento,  $load = 0$  memorizzazione



Oppure posso unire il segnale di load con il clock in una AND



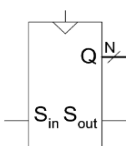
## Output o Scaricamento

- Scaricamento parallelo
- Scaricamento seriale

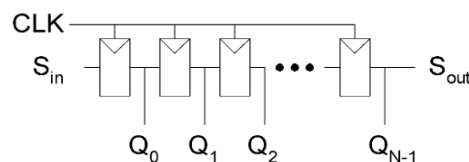
## Registri a scorrimento

- Ad ogni colpo di clock:
  - sposta (shift) un nuovo bit dentro;
  - sposta (shift) un bit fuori;
- Usato per la conversione *serial-to-parallel*: converte gli input seriali ( $S_{in}$ ) in output paralleli ( $Q_{0:N-1}$ );

**Symbol:**



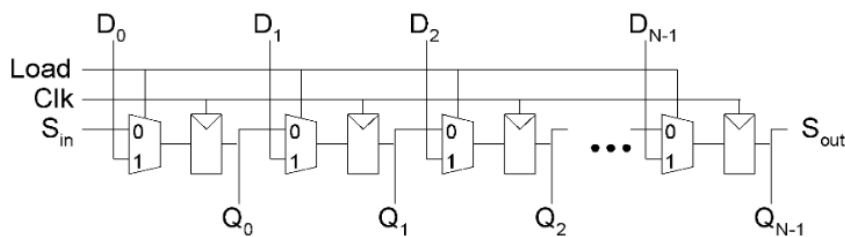
**Implementation:**



## Registri a scorrimento con caricamento parallelo

- quando  $load = 1$  agisce come un normale registratore a N-bit ( $Q_n$  dipende da  $D_n$ );

- quando  $load = 0$  agisce come uno Shift Register ( $Q_n$  dipende da  $S_{in}$ );
- adesso si può comportare sia come un convertitore serial-to-parallel che come un parallel-to-serial



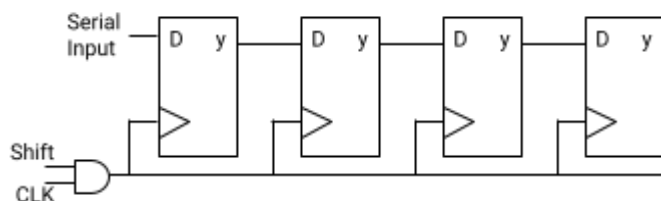
Gli shifter registers possono essere utilizzati per realizzare un trasmettitore seriale che crea un segnale di dati seriali da trasmettere (attraverso un cavo solitamente) ad un ricevitore che riconvertirà i dati.

- Gli N bit di dato  $D_{in}$  sono inviati allo shifter register  $SR_1$ ;
- I segnali di clock e  $S_{out}$  sono inviati al ricevitore  $SR_2$ ;
- $SR_2$  viene utilizzato per ricostruire il dato  $D_{out}$ ;

## Caricamento / Scaricamento

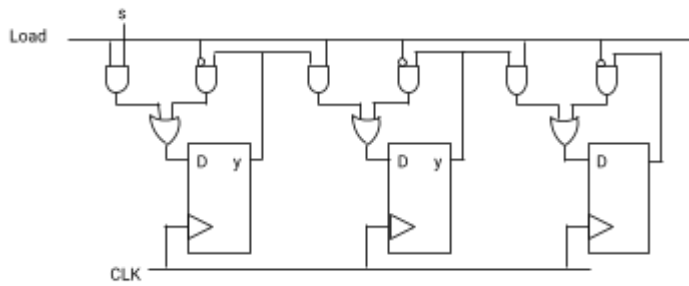
**Caricamento:** Riceve in input una linea di dati, i cui valori ad ogni fronte d'onda discendente di clock vengono memorizzati progressivamente, purché il segnale  $shift = 1$

**Scaricamento:** Sull'unica linea di uscita vengono riproposti in sequenza gli n valori caricati dall'input, uno per ogni fronte d'onda discendente del clock in cui  $shift = 1$

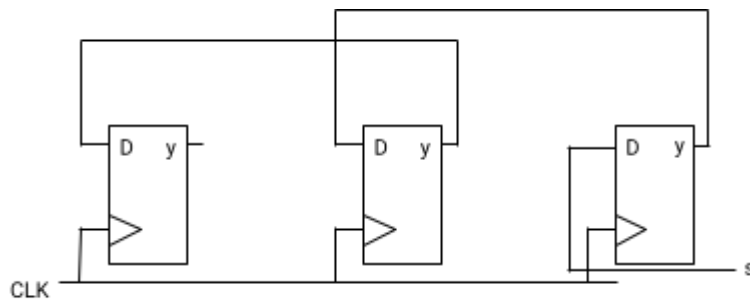


## Registro a caricamento con shift a destra

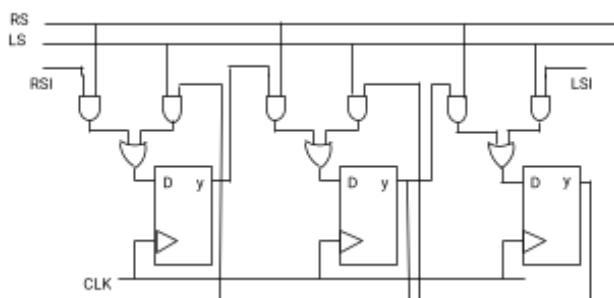
Il caricamento è eseguito con un singolo input seriale e ad ogni ciclo di clock viene caricato un valore e con un segnale di controllo eseguiamo o meno lo scorrimento  
 Se  $Load = 1 \Rightarrow$  Eseguo lo shift,  $Load = 0 \Rightarrow$  Memorizzo



## Registro scorrimento a sinistra



## Registro scorrimento bidirezionale



$RS = 1 \Rightarrow \text{Shift a destra}$

$LS = 1 \Rightarrow \text{Shift a sinistra}$

**Nota:** o  $RS = 1$  oppure  $LS = 1$

## Registro universale

È il registro che mi permette di eseguire tutte le operazioni (shift right, shift left, load, mem)

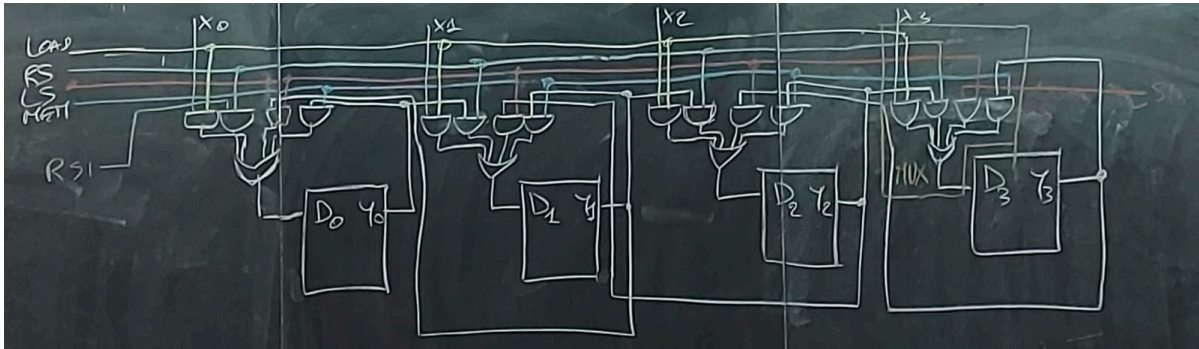
Quindi avrò bisogno dei segnali:

- $LS \Rightarrow \text{Scorro a sinistra}$
- $RS \Rightarrow \text{Scorro a destra}$
- $mem \Rightarrow \text{Memorizza}$



- *load*  $\Rightarrow$  Caricamento in parallelo

## Disegno OBJ



## Diagramma temporale

Vanno rappresentate le seguenti informazioni:

- Clock
- Input
- Stati (i bit degli stati,  $y_0, y_1, y_n$ )
- Uscite

## FSM

### Definizione

Una macchina a stati finiti è un modello di calcolo basato su un'ipotetica macchina composta da uno o più stati. Solo un singolo stato di questa macchina può essere attivo contemporaneamente. Significa che la macchina deve passare da uno stato all'altro per eseguire azioni diverse.

Un Sistema di Transizioni Etichettate (LTS, dall'inglese Labelled Transition System) è una quadrupla  $(Q, \Sigma, q_0, \delta)$  dove:

- $Q$  è l'insieme degli stati
- $\Sigma$  è l'alfabeto (di input)
- $q_0 \in Q$  è lo stato iniziale
- $\delta: Q \times \Sigma \rightarrow Q$  è la funzione di transizione.

Se alla definizione di LTS aggiungo anche un insieme  $F \subset Q$  di stati finali (o di accettazione) e impongo il vincolo che tutti gli insiemi considerati siano finiti, ottengo un automa a stati finiti, che è quindi una quintupla  $A = (Q, \Sigma, q_0, \delta, F)$ .

Se invece aggiungo alla definizione di LTS un alfabeto di output  $\Delta$ , una funzione di output  $\lambda$  e impongo il vincolo che tutti gli insiemi considerati siano finiti, ottengo un automa con output, che è quindi una sestupla  $M = (Q, \Sigma, \Delta, q_0, \delta, \lambda)$ .

Quindi in fine avrò:

$\Sigma \Rightarrow$  alfabeto di supporto, insieme finito di simboli

$Q \Rightarrow$  insieme finito di stati

$\delta \Rightarrow$  Funzione di transizione

$Q \times \Sigma$  cioè il prodotto cartesiano con descrizione di passaggio da uno stato all'altro

$U \Rightarrow$  Insieme finito di simboli di uscita

Moore: Uscite associate agli stati

Mealy: Uscite associate sugli archi

Cioè:

$\lambda_{moore}: Q \rightarrow U \Rightarrow$  In uno stato leggo l'uscita

$\lambda_{mealy}: Q \times \Sigma \rightarrow U \Rightarrow$  Tabella associa le uscite

## Procedura di progettazione di una FSM

Puoi trovare un esempio grafico qui  $\rightarrow$  [Esempio progettazione di una FSM](#)

1. Identificare input e output
2. Schizzo del diagramma di transizione dello stato
3. Scrivere la tabella delle transizioni di stato
4. Selezionare le codifiche degli stati
5. Per la macchina Moore:
  - a. Riscrivere la tabella delle transizioni di stato con le codifiche degli stati
  - b. Scrivere la tabella degli output
6. Per una macchina Mealy:
  - a. Riscrivere la tabella di transizione degli stati combinati e degli output con le codifiche degli stati
7. Scrivere equazioni booleane per lo stato successivo e la logica di uscita (puoi aiutarti con le mappe k)
8. Disegnare lo schema del circuito

## Derivare una FSM da uno schema

Puoi trovare un esempio grafico qui  $\rightarrow$  [Esempio da schema a FSM](#)

La derivazione del diagramma di transizione di stato da uno schema segue quasi il processo inverso rispetto alla progettazione FSM:

1. Esaminare il circuito, dichiarando ingressi, uscite e bit di stato.
2. Scrivere le equazioni degli stati successivi e degli output.
3. Creare le tabelle degli stati successivi e degli output.
4. Ridurre la tabella degli stati successivi per eliminare gli stati irraggiungibili.
5. Assegnare un nome (codifica) a ciascuna combinazione di bit di stato valida.
6. Riscrivere le tabelle degli stati successivi e degli output con i nomi di stato.
7. Disegnare il diagramma delle transizioni di stato. [Nota bene: all'esame dovrai disegnare al massimo 7 stati, in quanto noi trattiamo mappe k con massimo 3 bit]
8. Dichiarare a parole cosa fa la FSM.

## Codifica One-Hot o codifica a singolo 1

Viene utilizzato un bit di stato per ognuno degli stati. Viene chiamata in inglese codifica one hot perché in ogni momento uno solo dei bit è "caldo", cioè VERO.

### Esempio

Una FSM con tre stati con codifica a singolo 1 avrà come codifiche di stato 001, 010 e

100. Ogni bit di stato viene immagazzinato in un flip-flop, quindi una codifica a singolo 1 necessita di più flip-flop rispetto a una codifica binaria

## Equivalenza tra automi

Due automi (dello stesso tipo) sono equivalenti se, per ogni possibile sequenza di input, generano entrambi la stessa sequenza di output.

## Equivalenza tra stati

Due stati si dicono equivalenti quando a fronte degli stessi simboli di ingresso transitano nello stesso stato successivo e producono la stessa uscita.

## Semplificazione di FSM (Tabella Triangolare)

1. Confronto ogni stato con tutti gli altri (cioè i seguenti nella tabella dell'automa)
2. Se le uscite dei due stati **non** sono equivalenti metto una "X" nella casella corrispondente
3. Se le uscite sono uguali confronto per ogni input gli stati futuri

- a. Se gli stati futuri sono uguali, lascio la casella vuota ad indicare che i due stati sono uguali
  - b. Se gli stati futuri sono diversi segno la coppia/e degli stati futuri nella cella
4. Controllo le coppie salvate nelle caselle, se sono tutte equivalenti, gli stati sono uguali e metto un "O" altrimenti metto una "X" (nella tabella di esempio invece della "X" il testo delle coppie è in rosso per motivi di formattazione)

### Esempio

	a	b	$\Rightarrow$		a	b
$S_1$	$S_2 / 0$	$S_6 / 0$		$S_1$	$S_2 / 0$	$S_6 / 0$
$S_2$	$S_7 / 0$	$S_3 / 1$		$S_2$	$S_7 / 0$	$S_3 / 1$
$S_3$	$S_6 / 0$	$S_3 / 1$		$S_3$	$S_6 / 0$	$S_3 / 1$
$S_4$	$S_3 / 1$	$S_7 / 0$		$S_{4-6}$	$S_3 / 1$	$S_7 / 0$
$S_5$	$S_2 / 0$	$S_7 / 0$		$S_5$	$S_2 / 0$	$S_7 / 0$
$S_6$	$S_3 / 1$	$S_7 / 0$		$S_7$	$S_7 / 0$	$S_5 / 0$
$S_7$	$S_7 / 0$	$S_5 / 0$				

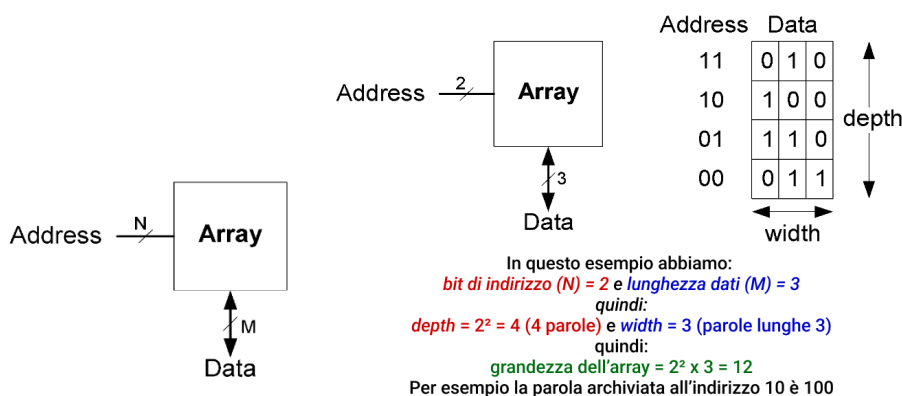
$S_2$	X					
$S_3$	X	$S_6, S_7$				
$S_4$	X	X	X			
$S_5$	$S_6, S_7$	X	X	X		
$S_6$	X	X	X		X	
$S_7$	$S_2, S_7$ $S_5, S_6$	X	X	X	$S_2, S_7$ $S_5, S_6$	X

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
--	-------	-------	-------	-------	-------	-------

## Memorie

### Memory arrays

- Archiviacono in modo efficiente grandi quantità di dati;
- Valore dei dati a M-bit letto/scritto su ciascun indirizzo univoco a N-bit;
- 3 tipi comuni:
  - Dynamic random access memory (DRAM)
  - Static random access memory (SRAM)
  - Read only memory (ROM)
- array bidimensionale di celle di bit;
- ogni cella di bit archivia un solo bit;
- N bit di indirizzo e dati di M bits:
  - $2^n$  righe e M colonne;
  - Depth: numero di righe (numero di parole);
  - Width: numero di colonne (dimensione della parola)
  - Grandezza dell'array:  $depth \times width = 2^n \times M$ ;

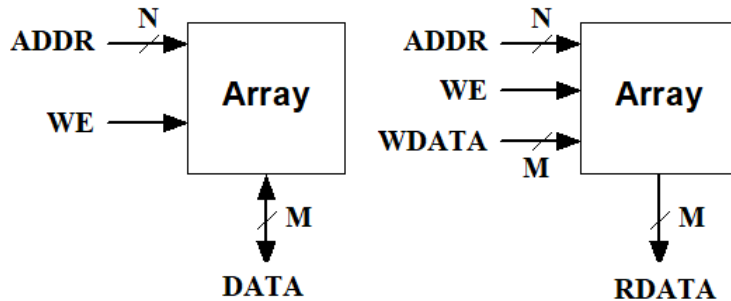


### RAM Array

Un Array della RAM ha le seguenti porte:

- N bit di indirizzo (ADDR)
- i dati (intesi come parole di M bit) possono avere:
  - una porta bidirezionale con M bit (DATA) che permette sia di scrivere che leggere le parole;

- una porta read\_data (RDATA) con M bits in lettura e una porta write\_data con M bits in scrittura (WDATA);
- un segnale di Enable per la scrittura (WE, Write Enable).

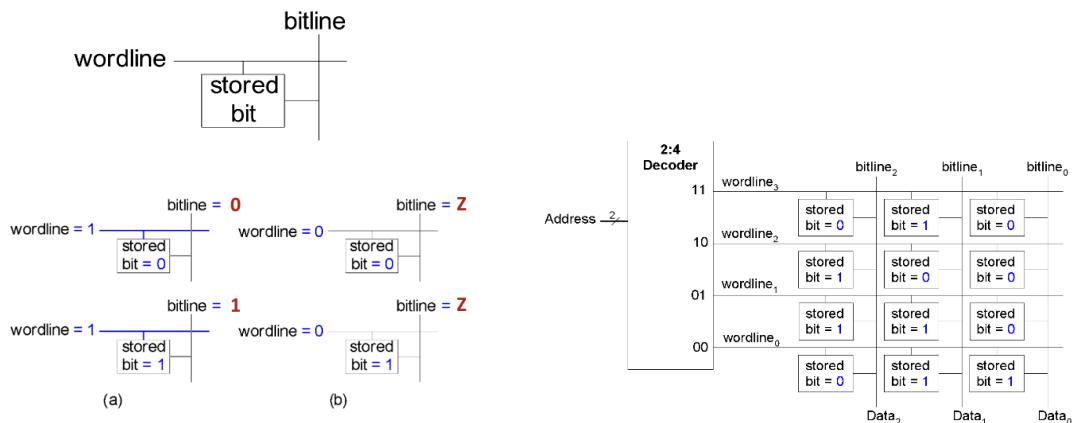


## Memory Array bit cells

la wordline è responsabile per selezionare una riga specifica di celle di memoria nell'array, mentre la bitline è responsabile per leggere o scrivere dati nelle colonne specifiche di quelle celle di memoria selezionate dalla wordline.

La wordline:

- si comporta come un enable;
- singola riga nell'array di memoria letta/scritta
- corrisponde ad un indirizzo univoco;
- solo una wordline HIGH alla volta;



## RAM

- è volatile: perde i dati quando viene spenta;
- letta e scritta velocemente;

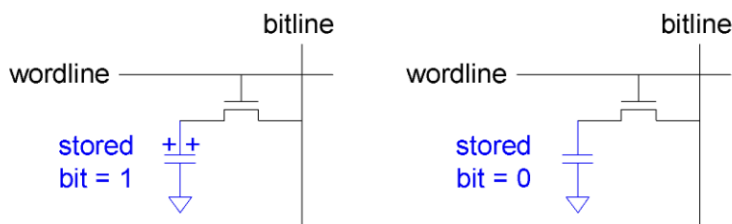
Esistono due tipi di RAM che differiscono in come archiviano i dati:

- DRAM (Dynamic random access memory) che usa dei *capacitor*;

- SRAM (Static random access memory) che usa *cross-coupled inverters* (inverter ad accoppiamento incrociato).

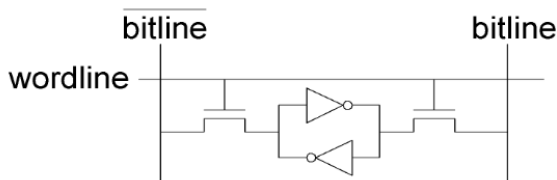
## DRAM

- i bits dei dati sono archiviati su capacitor;
- è dinamica perché il valore deve essere refreshato (riscritto) periodicamente e dopo la lettura
  - perdite di carica del capacitor degradano il valore;
  - la lettura distrugge il valore archiviato.



## SRAM

Non ha bisogno di essere refreshata come la DRAM, però la sua struttura la rende costosa e più grande.



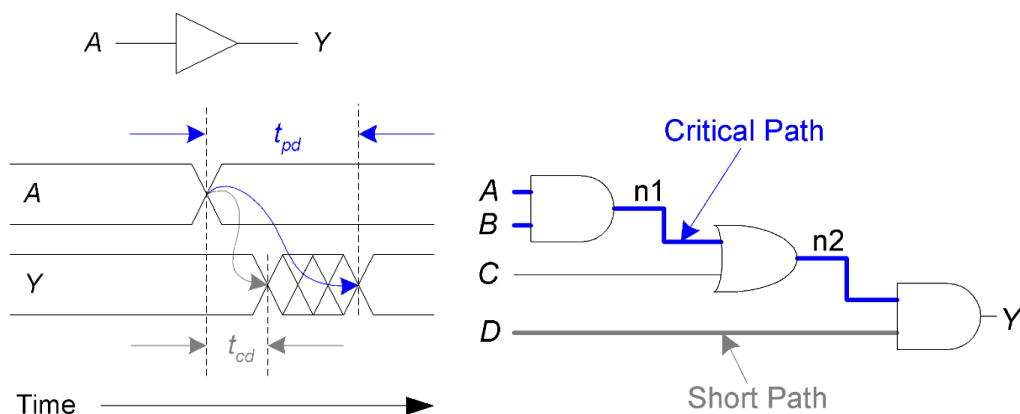
## ROM

- è NON volatile: mantiene i dati quando spenta;
- letta velocemente ma la scrittura è impossibile o molto lenta;

## Timing

## Logica combinatoria

Delay	/	tempo tra il cambio dell'input e il cambio dell'output
Propagation delay	$t_{pd}$	ritardo massimo dall'ingresso all'uscita (percorso critico)
Contamination delay	$t_{cd}$	ritardo minimo dall'ingresso all'uscita (percorso corto)



## Logica sequenziale

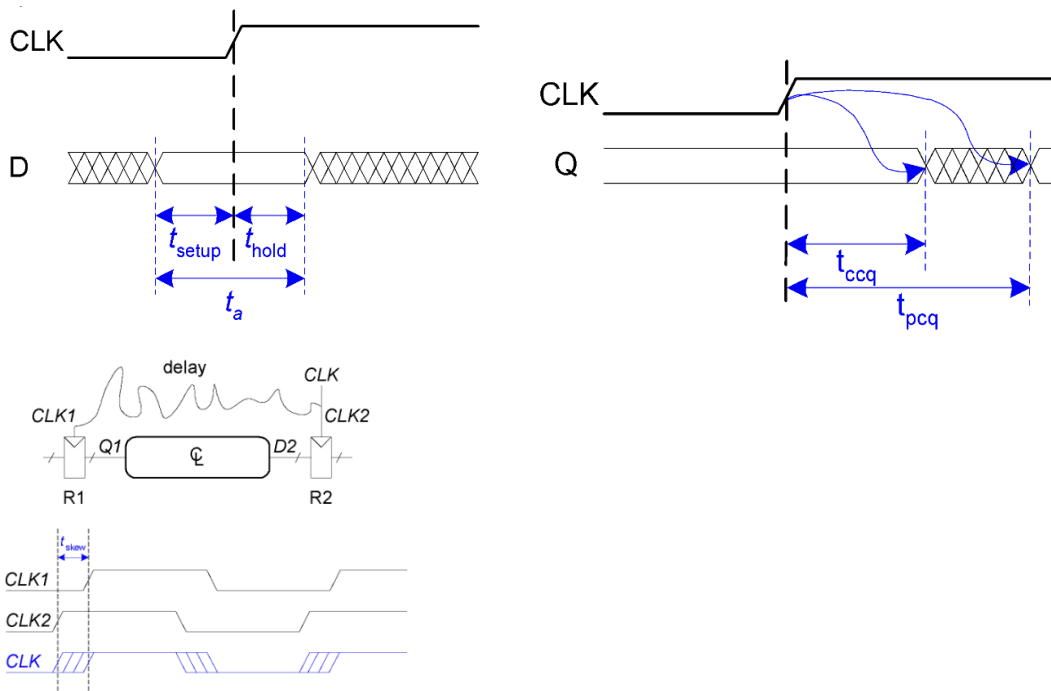
I flip-flop campionano D sul fronte di clock. Quindi D deve essere stabile:

- quando campionato
- attorno al fronte di clock

In caso contrario, può verificarsi instabilità

Definizioni		
Setup time	$t_{setup}$	il tempo prima del fronte di clock in cui i dati devono essere stabili (ovvero non cambino)
Hold time	$t_{hold}$	il tempo successivo al fronte di clock nel quale i dati devono essere stabili
Aperture time	$t_a$	Il tempo totale vicino al fronte di clock nel quale i dati devono essere stabili ( $t_a = t_{setup} + t_{hold}$ )
Propagation delay	$t_{pcq}$	tempo dopo il fronte del clock in cui è garantito che l'uscita Q sia stabile (ovvero, smetta di cambiare)
Contamination delay	$t_{ccq}$	tempo dopo il fronte di clock in cui Q potrebbe essere instabile (ovvero, iniziare a cambiare)
Clock Skew	$t_{skew}$	la differenza tra due colpi di clock.

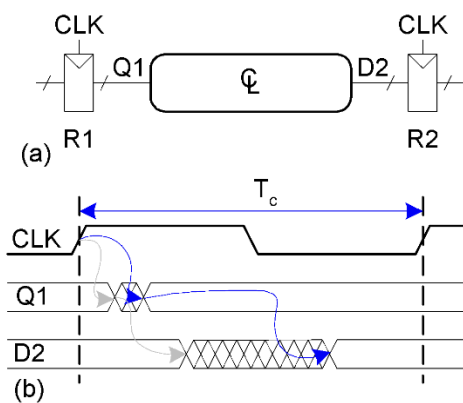




## Dynamic Discipline

É un insieme di regole che i circuiti devono seguire (riguardo il timing) per evitare risultati inaspettati o instabili.

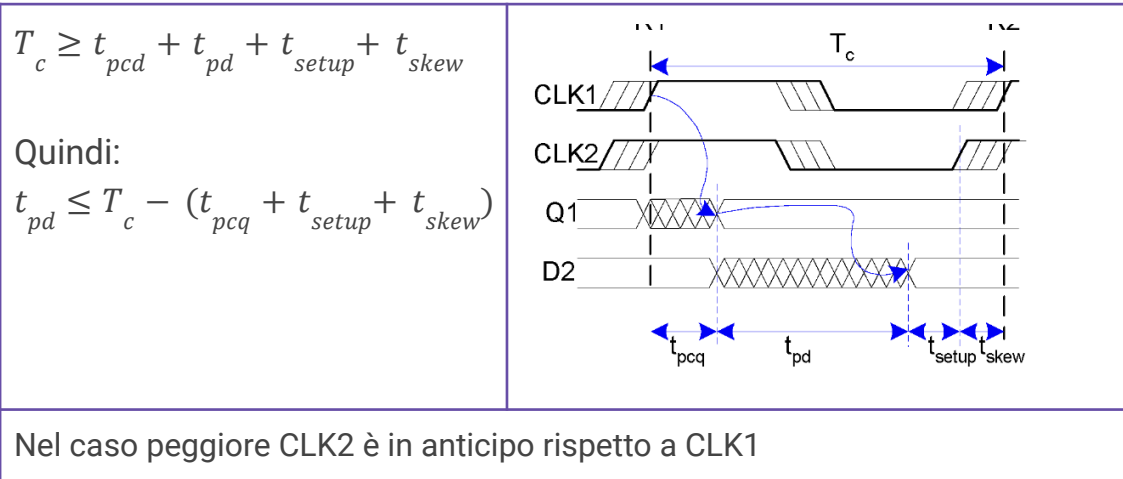
Il ritardo tra i registri ha un delay minimo e massimo, che dipende dai delays degli elementi del circuito.



$T_c \rightarrow$  tempo del clock

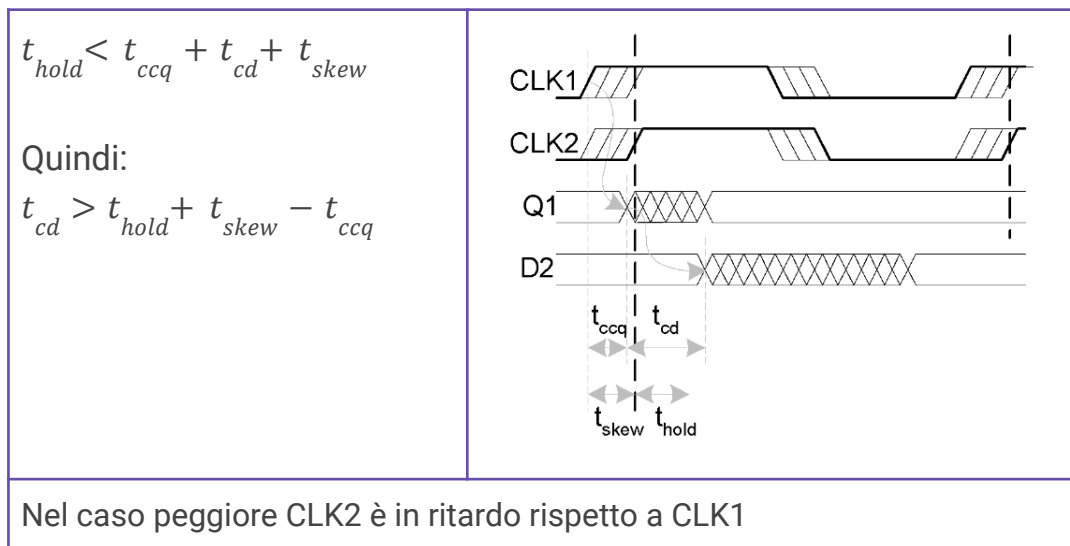
## Vincolo Setup Time

Dipende dal delay massimo dal registro R1 attraverso la logica combinatoria a R2. L'ingresso nel registro R2 deve essere stabile almeno  $t_{setup}$  prima del fronte di clock



## Vincolo Hold Time

Dipende dal delay minimo dal registro R1 attraverso la logica combinatoria a R2. L'ingresso nel registro R2 deve essere stabile almeno  $t_{hold}$  dopo il fronte di clock



Esegui sempre l'analisi del caso peggiore per garantire che la disciplina dinamica non venga violata.

# Systemverilog

## HDL

Hardware Description Language (HDL) è un linguaggio utilizzato per descrivere la struttura, il comportamento e i tempi dei circuiti elettronici e, più comunemente, dei circuiti logici digitali.

- specifica solo la funzione logica;
- Lo strumento di progettazione assistita da computer (CAD) produce o sintetizza le porte ottimizzate.

Esistono due HDL principali:

- **SystemVerilog:**
  - sviluppato nel 1984 da Gateway Design Automation;
  - Standard IEEE (1364) nel 1995;
  - Esteso nel 2005 (IEEE STD 1800-2009);
- **VHDL 2008:**
  - Sviluppato nel 1981 dal Dipartimento della Difesa americano;
  - Standard IEEE (1076) nel 1987;
  - Aggiornato nel 2008 (IEEE STD 1076-2008);

## Applicazioni

Un HDL ha diverse applicazioni, tra cui:

- **Simulazione**
  - gli ingressi vengono applicati al circuito;
  - La correttezza delle uscite viene controllata;
  - milioni di dollari risparmiati eseguendo il debug nella simulazione invece che nell'hardware;
- **Sintesi**
  - Trasforma il codice HDL in una netlist che descrive l'hardware (cioè un elenco di porte e i cavi che le collegano).

Quando si usa un HDL, bisogna pensare all'hardware che l'HDL dovrebbe produrre.

## Modulo Systemverilog



Ci sono due tipi di moduli:

- **Comportamentale:** descrive cosa fa un modulo;
- **Strutturale:** descrive come è costruito partendo da moduli più semplici;

## Sintassi

- distinzione tra maiuscole e minuscole:
  - Esempio: reset e Reset non sono lo stesso segnale.
- Nessun nome che inizia con numeri:
  - Esempio: 2mux è un nome non valido
- Gli spazi bianchi sono stati ignorati
- Commenti:
  - // commento su una sola riga
  - /\* multilinea  
commento \*/

## Precedenza operatori

~	negazione	più alta
*, /, %	moltiplicazione, divisione, modulo	
+, -	somma, sottrazione	
<<, >>	scorrimento	
<<<, >>>	scorrimento aritmetico	
<, <=, >, >=	confronto	
==, !=	uguale, diverso	
&, ~&	AND, NAND	
^, ~^	XOR, XNOR	
, ~	OR, NOR	più bassa
?:	operatore ternario	

## Numeri

Formato: N'Bvalore

- N = numero di bit

- B = base

N'B è opzionale ma consigliato (di default è decimale)

## Tipo di dati logic

Il tipo di dati logic può assumere valori 0, 1, z e x.

- x indica un livello logico non valido. Se un bus viene portato simultaneamente a 0 e 1 da due buffer tristate abilitati (o altre porte), il risultato è x, che indica conflitto.
- Se tutti i buffer tristate che guidano un bus sono contemporaneamente OFF, il bus fluttua, indicato da z.
- All'inizio della simulazione, i nodi di stato come le uscite del flip-flop vengono inizializzati su uno stato sconosciuto (x in SystemVerilog)

## Strutture

### Always

#### Struttura generale:

```
always @(sensitivity list)
    statement;
```

Ogni volta che si verifica l'evento nella sensitivity list, l'istruzione viene eseguita.

Le dichiarazioni che devono essere contenute all'interno delle dichiarazioni *always*:

- if/else;
- case, casez;
- for.

Se viene utilizzato un *always* generico, la sensitivity list DEVE includere tutti i segnali utilizzati nell'*always*.

tutte le uscite dovrebbero avere valori predefiniti o devono essere assegnate per ogni combinazione di ingressi. In caso contrario, verrà generato un latch per mantenere il valore corrente.

### Generate

Il costrutto loop *generate* fornisce un metodo semplice e conciso per creare più istanze di istanze di moduli, assegnare istruzioni e così via. Considerala come una macchina per la clonazione.

Il costrutto *generate* condizionale ti consente di modificare la struttura del tuo progetto in base ai valori dei parametri.