dalla Pascalina alla Computazione alla Computabilità

Prof. Laura Pozzi, USI, Lugano, Switzerland

Mi presento

- Bachelor e Master in Ingegneria Informatica al Politecnico di Milano
- Dottorato al Politecnico di Milano, in cooperazione con Hewlett Packard Laboratories Bristol
  - 1996 to 2000
- Un anno in azienda nella Silicon Valley (STMicroelectronics)
- E allo stesso tempo un industrial visitor alla Università della California a Berkeley
- PostDoc all' EPFL nel Laboratorio di Architettura dei Calcolatori
  - 2001 to 2005
- Professore all'USI Lugano – Assistant poi Associate poi Ordinario

USI

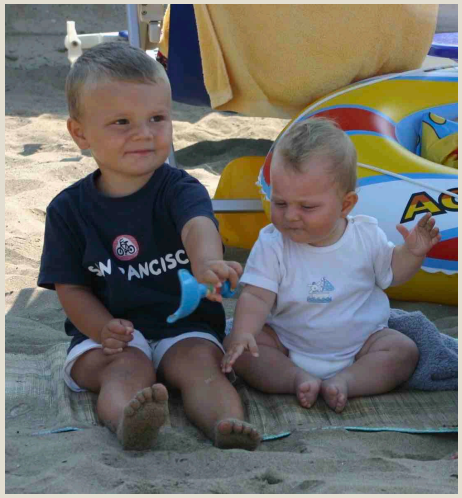
5 facoltà:

Informatica

Economia
Comunicazoni
Architettura
BioMedicina

More About me

- Durante il mio postdoc sono nati i miei due figli
- E quindi la sfida di poter gestire una carriera accademica e una famiglia allo stesso tempo
- La mia soluzione è stata quella di lavorare part time: ho lavorato al 60% per gli scorsi 20 anni



Pascal! E Computazione

Pascal inventa la Pascalina nel 1642

Si tratta di un calcolatore meccanico

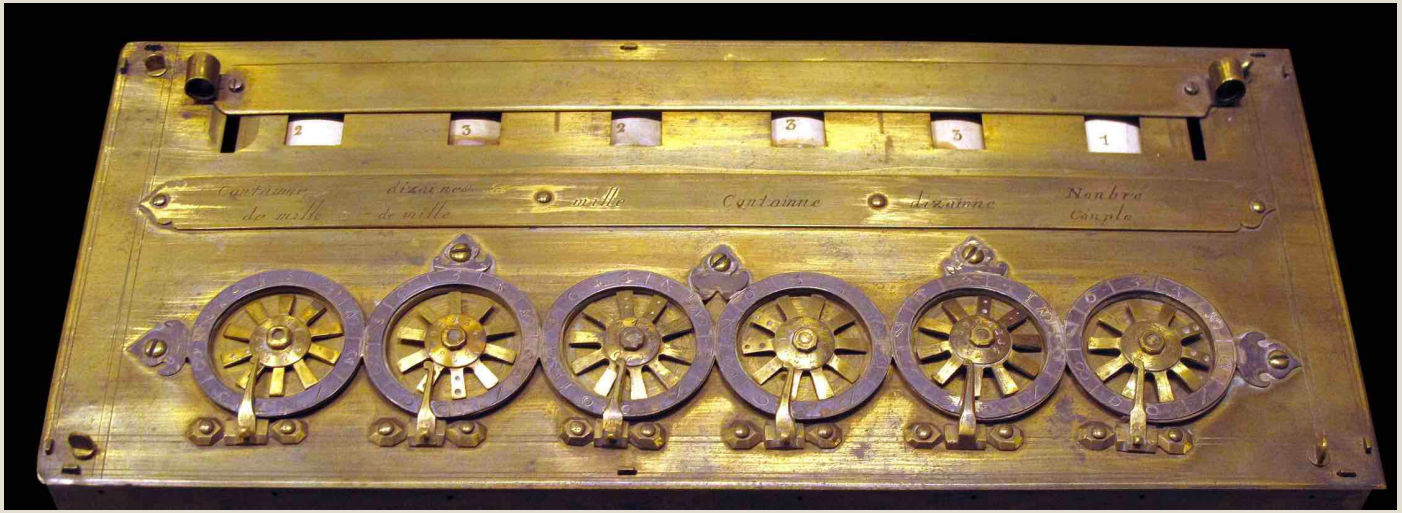


Fig. 1.

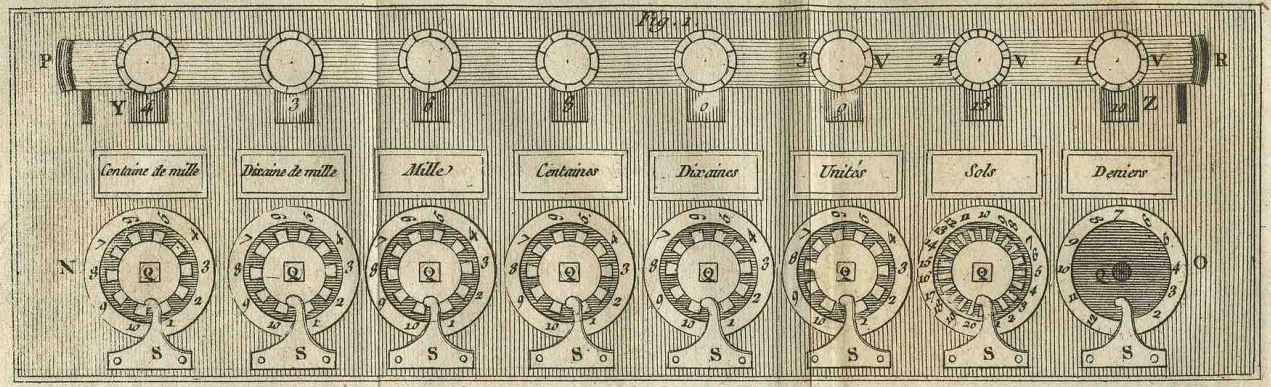
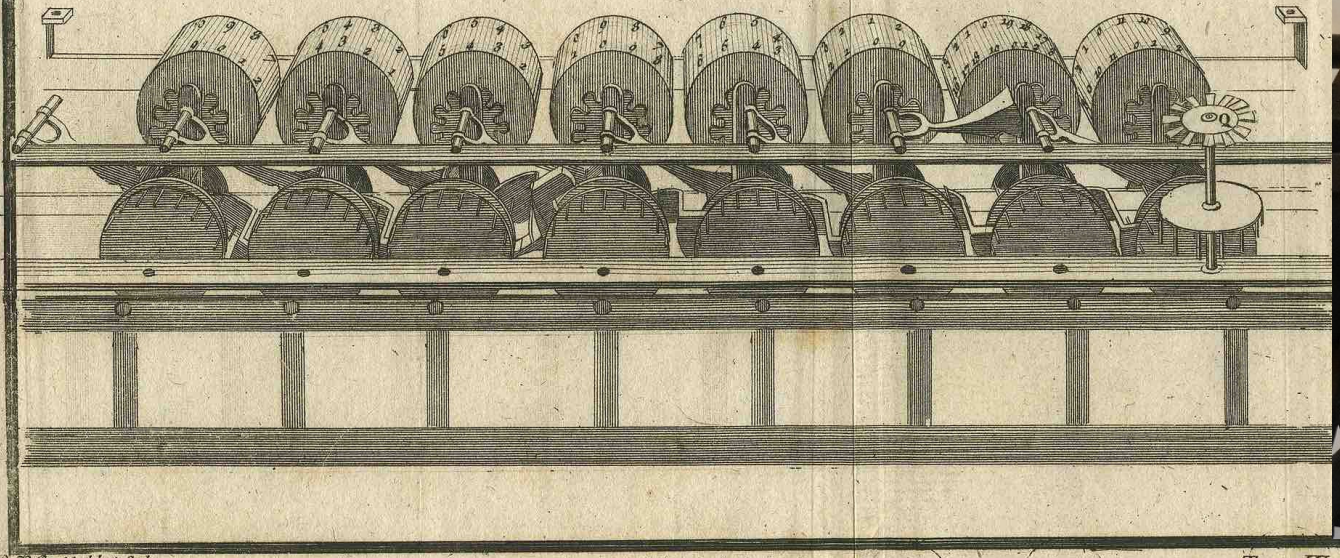
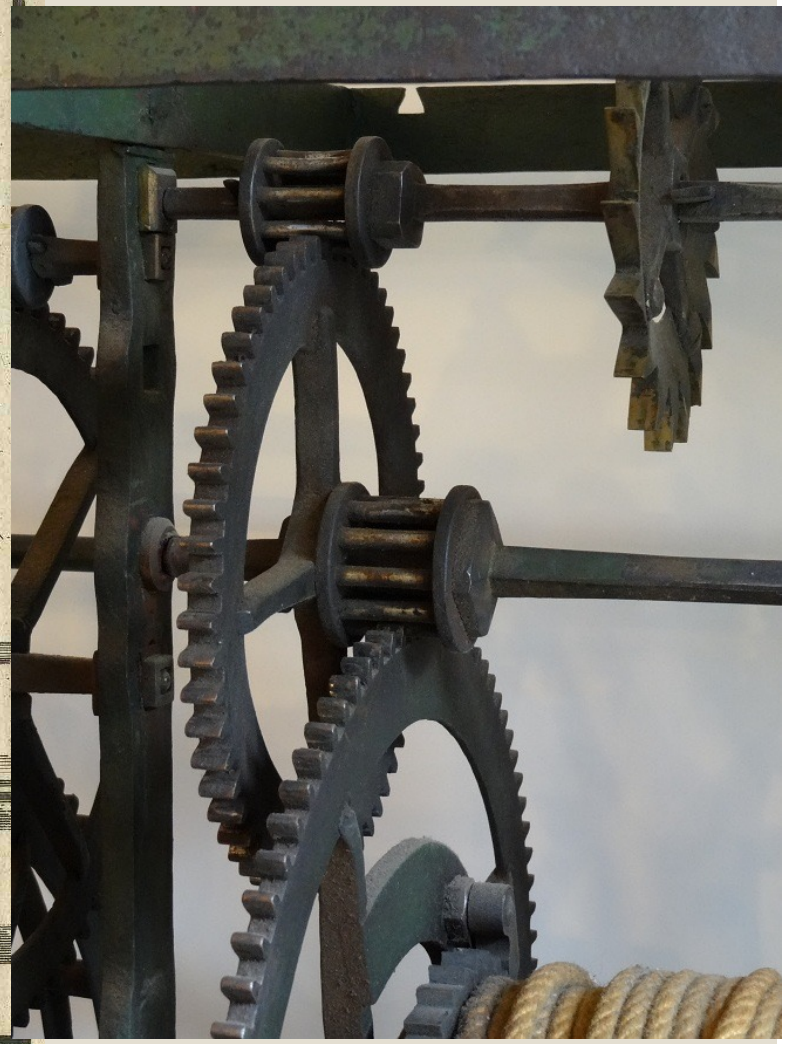

P
R
Y
Z
Centaine de mille
Dixaine de mille
Mille
Centaines
Dixaines
Unités
Sols
Deniers
N
O
S

Fig. 2.





Computazione meccanica / digitale

- I calcolatore meccanici eseguono calcoli utilizzando parti meccaniche come ingranaggi e leve
- I calcolatori digitali eseguono calcoli utilizzando componenti elettronici, come i transistor e le porte logiche
  - Tipicamente elaborando dati in forma binaria

Addizione

18 + 15 = 33

```
  1 8
+ 1 5
-----
  3 3
```

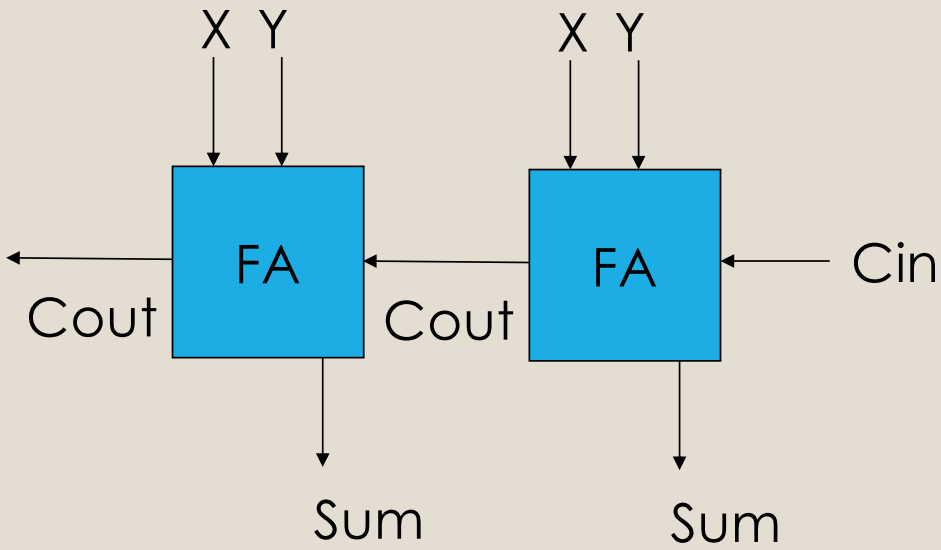
10010 + 01111 = 100001

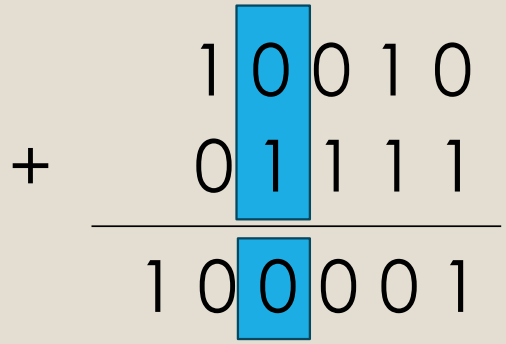
```
    1 0 0 1 0
+   0 1 1 1 1
-------------
  1 0 0 0 0 1
```

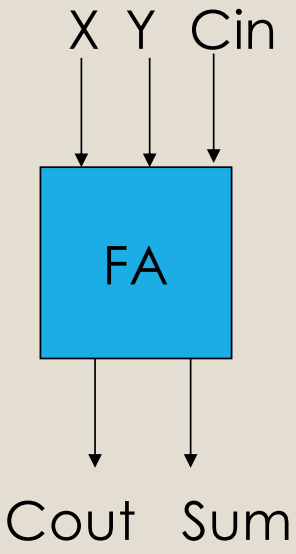
Decimale: 18 = 1x10 + 8 x1

Binario: 18 = 1x16 + 1x2

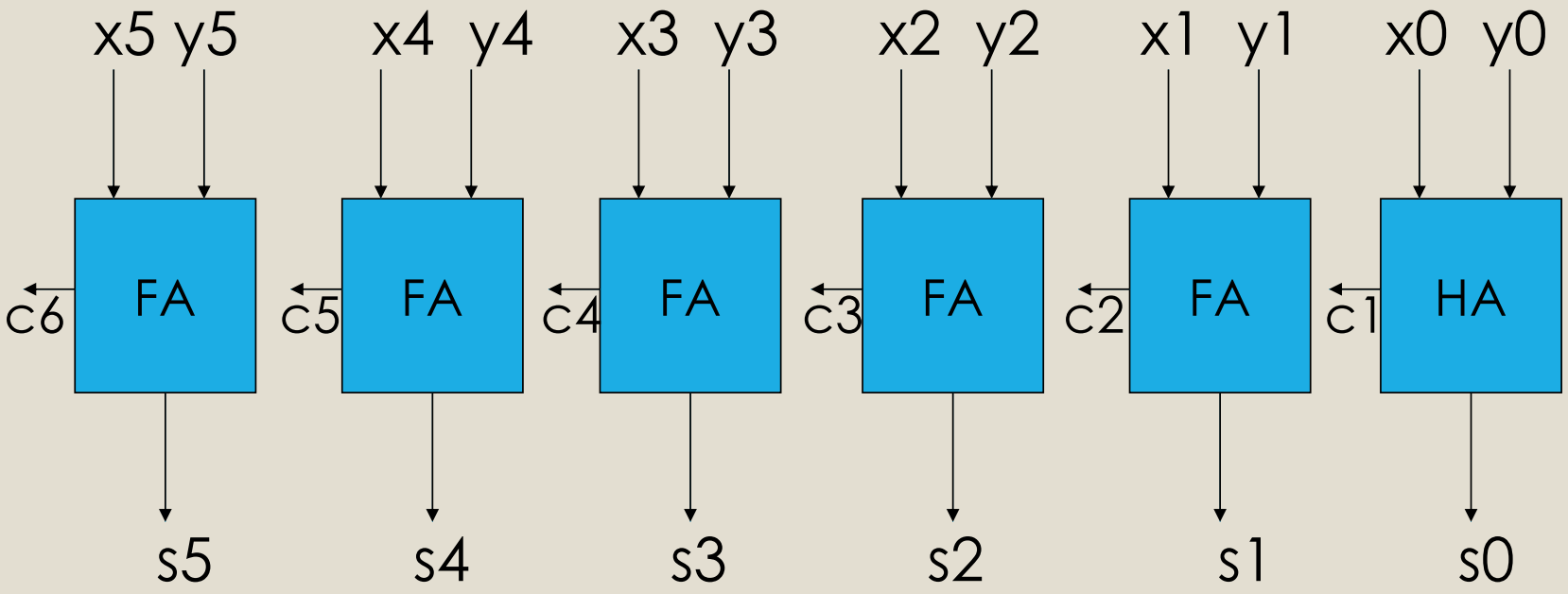
Full Adder


X Y
X Y
FA
FA
Cout
Cout
Cin
Sum
Sum


1 0 0 1 0
+ 0 1 1 1 1
1 0 0 0 0 1


X Y Cin
FA
Cout Sum

Ripple Carry Adder



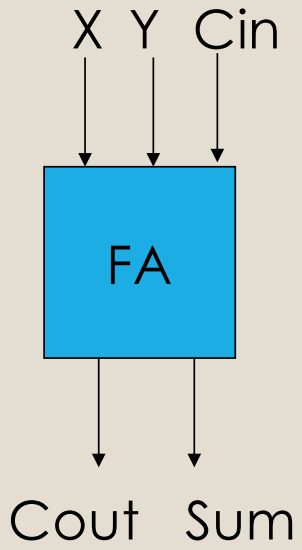
L'addizionatore più semplice

→ Il più piccolo → Il più lento

```
  1 0 0 1 0
+ 0 1 1 1 1
-----------
1 0 0 0 0 1
```

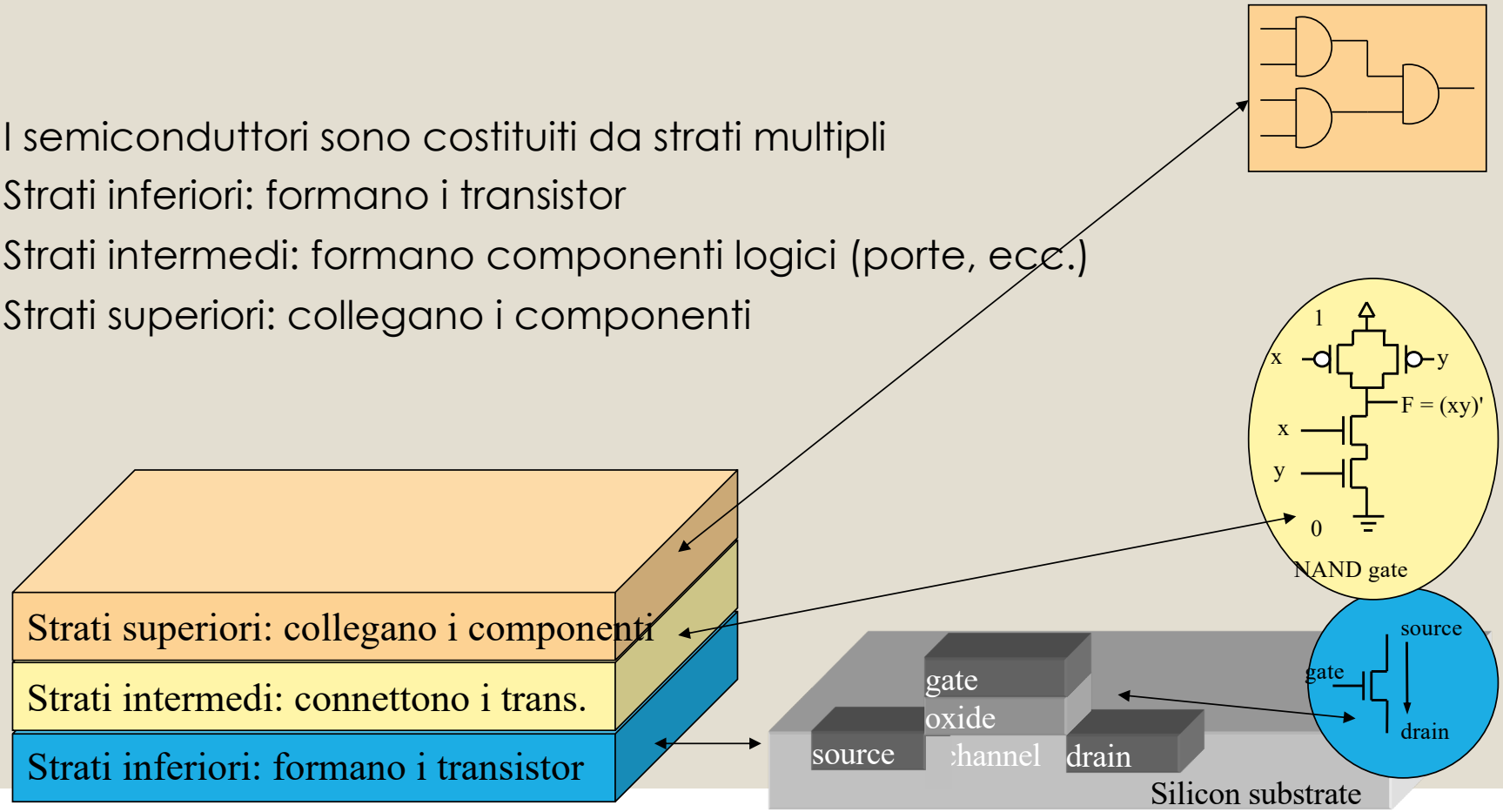
Come realizziamo un circuito?

- Dobbiamo passare dalle tavole di verità e dalle porte logiche



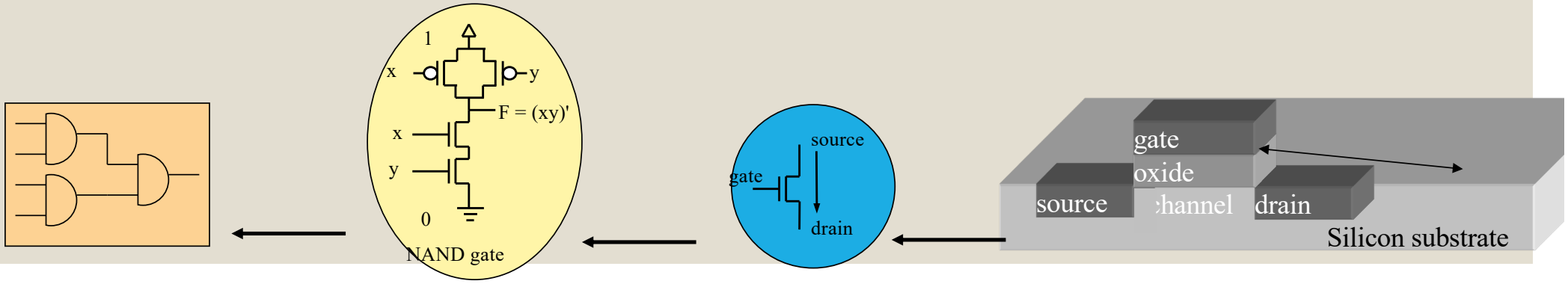
Tecnologia dei Circuiti Integrati

- I semiconduttori sono costituiti da strati multipli
- Strati inferiori: formano i transistor
- Strati intermedi: formano componenti logici (porte, ecc.)
- Strati superiori: collegano i componenti



Astrazione

- Una porta logica è costituita da transistors, che sono costituiti da substrati di silicio
- Ma noi astraiamo
- Modelliamo una porta logica attraverso la sua tabella di verità
- Senza curarci di come sia realizzata

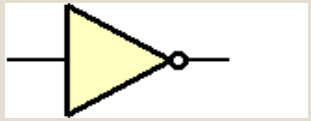


Porte Logiche

- NOT
  - Negazione F=A’ ($\overline{A}$)

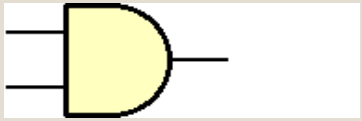
Tabella della Verità

A	F
0	1
1	0



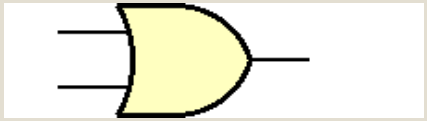
Porte Logiche

- AND
  - F=A · B
  - Zero in ingresso forza l'uscita a zero



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

- OR
  - F=A+B
  - Uno in ingresso forza l'uscita a uno



A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Circuiti

- Usiamo porte logiche(operatori) per costruire circuiti (espressioni)

- Un circuito è una implementazione di un'espressione booleana
  che a sua volta è una realizzazione di una tavola di verità
  che a sua volta è una rappresentazione di una funzionalità

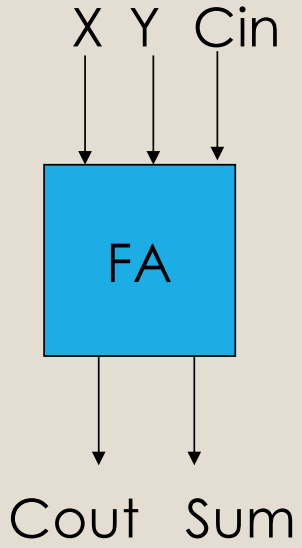
Addizione → tabella verità → espressione booleana → circuito

Torniamo al nostro addizionatore

- Avevamo detto: dobbiamo passare dalle tavole di verità e dalle porte logiche

```
  1 0 0 1 0
+ 0 1 1 1 1
-----------
1 0 0 0 0 1
```

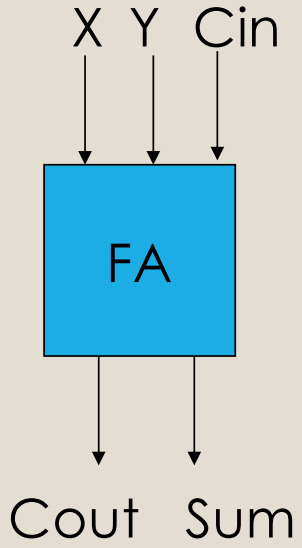
	X	Y	Cin	Cout	Sum
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	1	1



Realizziamo ora un circuito corrispondente a questa tabella di verità

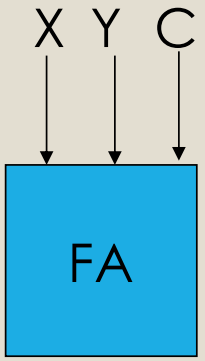
```
   1 0 0 1 0
+  0 1 1 1 1
-----------
 1 0 0 0 0 1
```

	X	Y	Cin	Cout	Sum
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	1	1



Definizioni

- letterale: singola variabile in ingresso, o complemento di una variabile in ingresso
  - Esempi: X, Y, Y', C'
- prodotto: prodotto logico (porta **and**) di 1 o più letterali
  - Esempi: X, XY, X'Y, XYC', …
- somma di prodotti : somma logica (porta **or**) di prodotti logici
  - Esempi: X **+** XY **+** X'Y **+** XYC'
- mintermine: prodotto di esattamente *n* letterali, per una funzione a *n* variabili
  - Esempi: XYC' , X'Y'C, XY'C



Corrispondenza tra mintermini e righe della tabella di verità

- Un mintermine è un prodotto che vale **1** per una sola combinazione di valori in ingresso
- Cioè per una sola riga della tabella di verità
- Per esempio: XYZ' vale 1 soltanto per la seguente combinazione:
- X=1, Y=1, Z=0 → XYZ'=1
- quindi solo nella riga 110 della tabella della verità

Row	X	Y	Z	F	**Minterm**
0	0	0	0	F(0,0,0)	X'Y'Z'
1	0	0	1	F(0,0,1)	X'Y'Z
2	0	1	0	F(0,1,0)	X'YZ'
3	0	1	1	F(0,1,1)	X'YZ
4	1	0	0	F(1,0,0)	XY'Z'
5	1	0	1	F(1,0,1)	XY'Z
6	1	1	0	F(1,1,0)	XYZ'
7	1	1	1	F(1,1,1)	XYZ

Da tabella di verità a circuito

- Come realizzare un circuito che corrisponde a una data tabella di verità

Scrivere una somma di mintermini

quali mintermini?

quelli che corrispondono alle righe della tabella

in cui l'uscita è 1

Row	X	Y	Z	F	**Minterm**
0	0	0	0	F(0,0,0)	X'Y'Z'
1	0	0	1	F(0,0,1)	X'Y'Z
2	0	1	0	F(0,1,0)	X'YZ'
3	0	1	1	F(0,1,1)	X'YZ
4	1	0	0	F(1,0,0)	XY'Z'
5	1	0	1	F(1,0,1)	XY'Z
6	1	1	0	F(1,1,0)	XYZ'
7	1	1	1	F(1,1,1)	XYZ

Da tabella di verità a circuito

- Come realizzare un circuito che corrisponde a una data tabella di verità

Row	X	Y	Z	F	Minterm
0	0	0	0	**1**	**X'Y'Z'**
1	0	0	1	0	X'Y'Z
2	0	1	0	0	X'YZ'
3	0	1	1	**1**	**X'YZ**
4	1	0	0	**1**	**XY'Z'**
5	1	0	1	0	XY'Z
6	1	1	0	**1**	**XYZ'**
7	1	1	1	**1**	**XYZ**

Scrivere una somma di mintermini

quali mintermini?

quelli che corrispondono alle righe della tabella

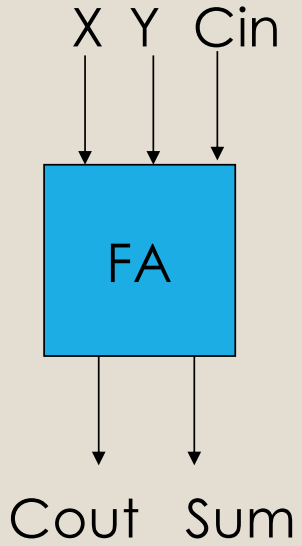
In cui l'uscita è 1

$$F(X,Y,Z) = X'Y'Z' + X'YZ + XY'Z' + XYZ' + XYZ$$

Realizziamo ora un circuito corrispondente a questa tabella di verità

```
   1 0 0 1 0
+  0 1 1 1 1
-----------
 1 0 0 0 0 1
```

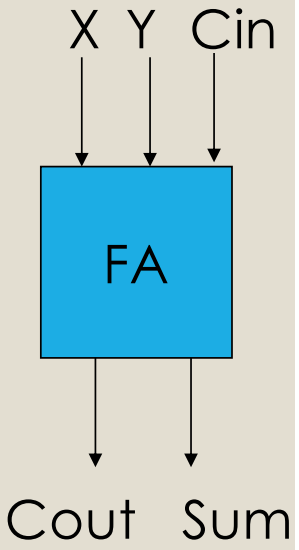
	X	Y	Cin	Cout	Sum
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	1	1



Somma di mintermini per Cout e per Sum

- Cout=
- X'YC +
- XY'C +
- XYC' +
- XYC

	X	Y	Cin	Cout	Sum
	0	0	0	0	0
	0	0	1	0	**1**
	0	1	0	0	**1**
	0	1	1	**1**	0
	1	0	0	0	**1**
	1	0	1	**1**	0
	1	1	0	**1**	0
	1	1	1	**1**	**1**



Sum = X'Y'C+X'YC'+XY'C'+XYC

```
  1 0 0 1 0
+ 0 1 1 1 1
-----------
1 0 0 0 0 1
```

Minimizzazione di Circuiti

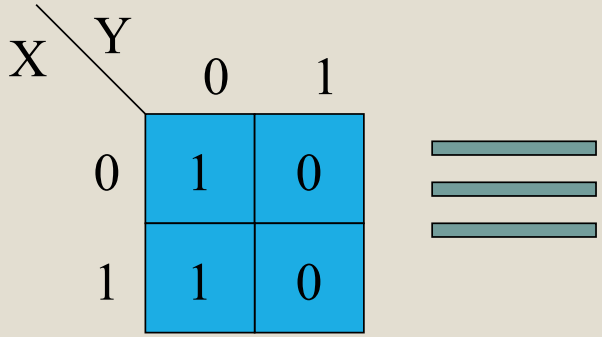
- Vogliamo ridurre il numero di porte che servono per realizzare un circuito
- Per minimizzare un'espressione, possiamo prima scriverla come Somma di Mintermini
- E poi applicare il seguente teorema

- XY + X Y' = X

generalizzato:

(un certo prodotto) Y + (un certo prodotto) Y' = un certo prodotto

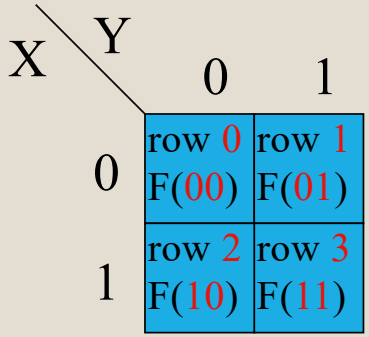
Mappe di Karnaugh



Row	X	Y	F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

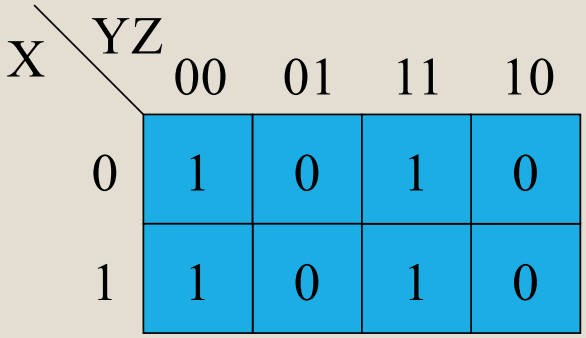
- It's another way to write a truth table
  - It has no more and no less information
- Each cell corresponds to a truth table row
  - Which means, to a minterm

Corrispondenza Cella MK e Riga TV



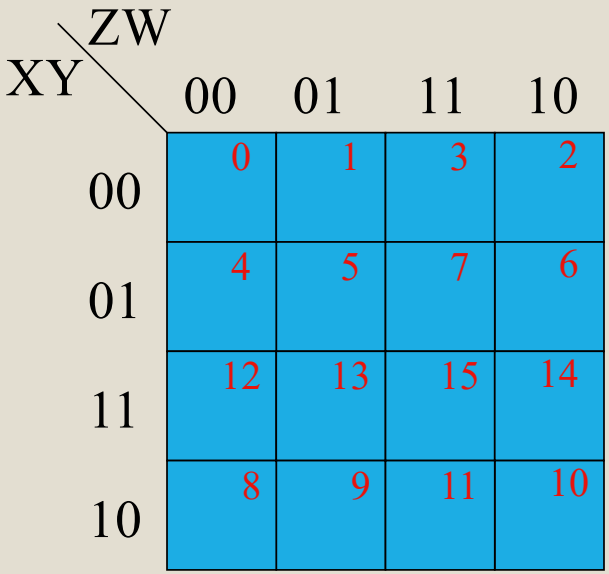
Row	X	Y	F
0	0	0	F(00)
1	0	1	F(01)
2	1	0	F(10)
3	1	1	F(11)

Mappa di Karnaugh per 3 variabili

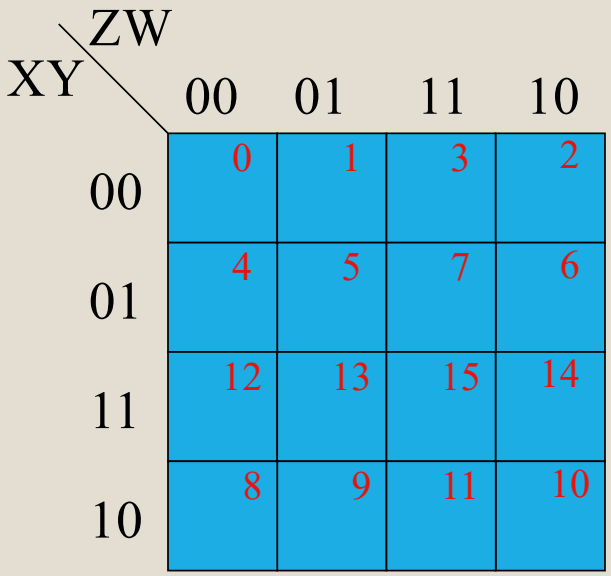


Row	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Mappa di Karnaugh per 4 variabili



Sequenza 00-01-11-10



- Ogni cella corrisponde a una combinazione in input che differisce da ogni cella adiacente in solo una variabile

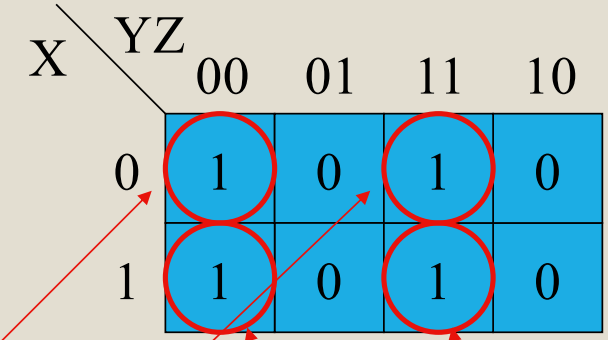
Perché vogliamo applicare:

AB + AB' = A

E vogliamo facilmente visualizzarne l'applicabilità

→ celle adiacenti

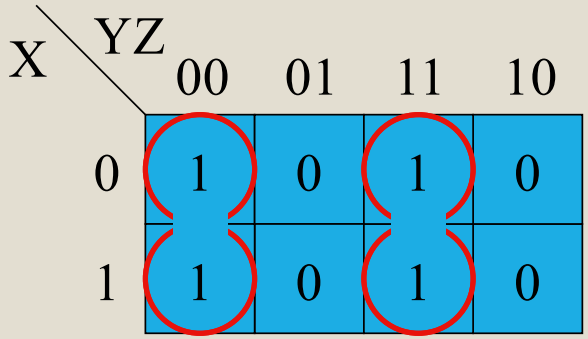
Mettiamo in pratica su un esempio



Ogni cella nella mappa
(come ogni riga nella tabella di verità)
corrisponde a un minterm

- Qual è la somma di minterms per questa funzione ?
- X'Y'Z' + X'YZ + XY'Z' + XYZ
- Possiamo semplificare?
- Sì! Usando:
  - (prodotto) L + (prodotto) L' = prodotto

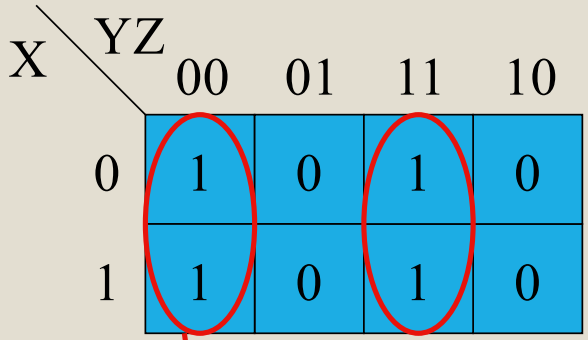
Semplifichiamo



- X'Y'Z' + X'YZ + XY'Z' + XYZ =
- = Y'Z' (X'+X) + YZ (X' + X) = Y'Z' + YZ

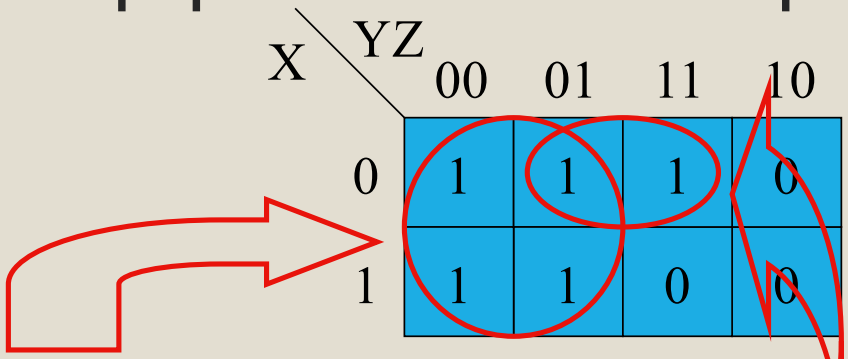
- Per espressioni lunghe: difficile individuazione
  - Le mappe di Karnaugh lo rendono semplicissimo!

Come usare le Mappe di Karnaugh per semplificare circuiti



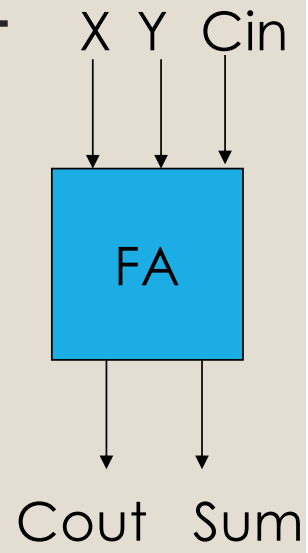
- Cerchiamo celle adiacenti di 1s
- Y'Z' YZ
- Un singolo 1 corrisponde a un minterm
- Due uni adiacenti corrispondono a un prodotto più piccolo
- 4 uni adiacenti corrispondono a un prodotto ancora più piccolo …

Corrispondenza tra gruppi di uno e prodotti



- Questo Gruppo corrisponde a quale prodotto?
- Y' perché Y è costante (valle sempre 0) in tutte le 4 celle, mentre gli altri input variano
  - X'Y'Z'+XY'Z'+X'Y'Z+XY'Z = Y'Z' + Y'Z = Y'
- E questo ?
  - X'Z
  - (perché X vale zero in tutte e due le celle, mentre Y varia)

Minimizziamo il circuito per Cout

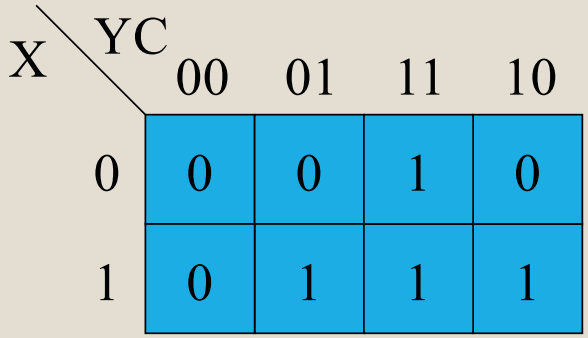


- Cout=
- X'YC +
- XY'C +
- XYC' +
- XYC

	X	Y	Cin	Cout	Sum
	0	0	0	0	0
	0	0	1	0	**1**
	0	1	0	0	**1**
	0	1	1	**1**	0
	1	0	0	0	**1**
	1	0	1	**1**	0
	1	1	0	**1**	0
	1	1	1	**1**	**1**

```
  10010
+ 01111
--------
 100001
```

Minimizziamo Cout

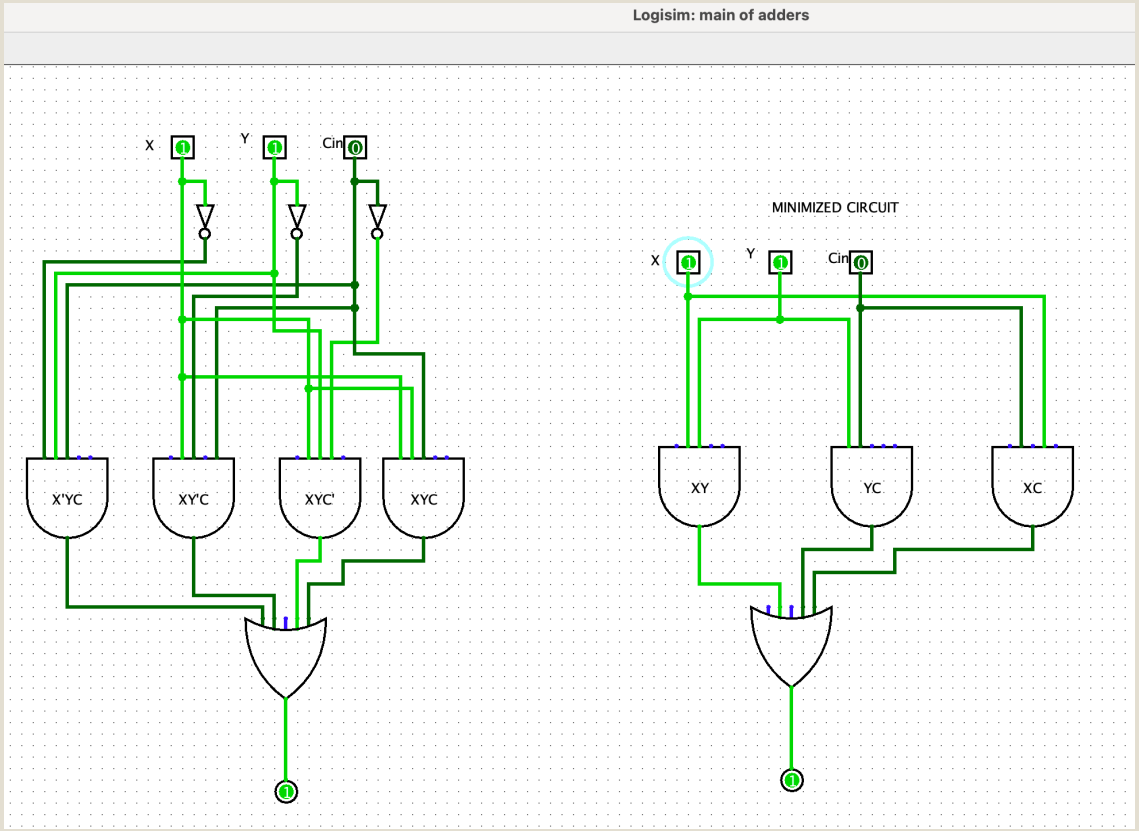


- Cout= X'YC + XY'C + XYC' + XYC

  = XY + XC + YC

Logisim

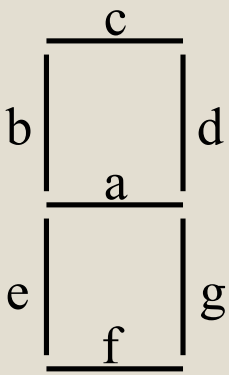
- A questo punto della presentazione, sono passata alla applicazoine Logisim, e vi ho mostrato una simulazione.
- Qui riporto solo uno screenshot

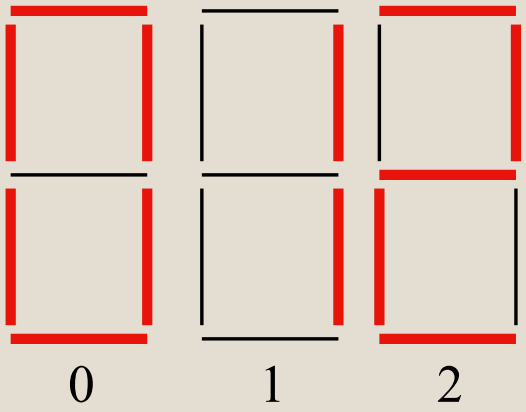


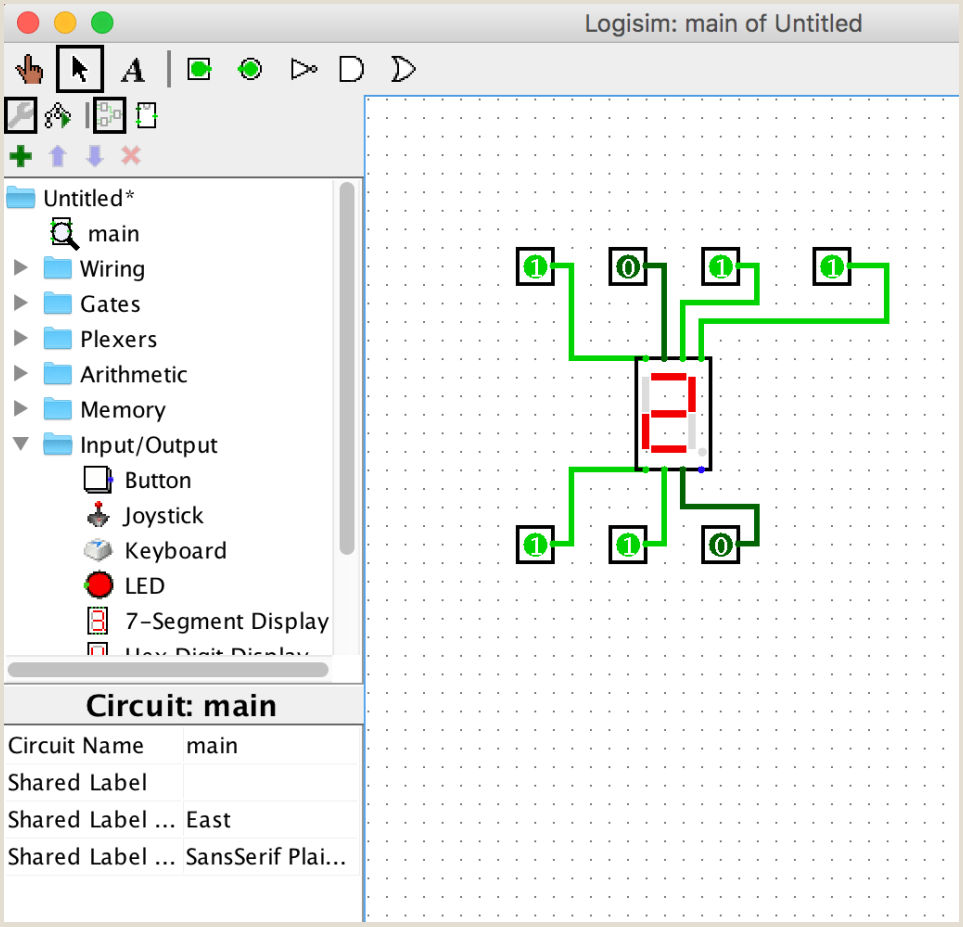
Logisim è un tool utile anche per sviluppare progetti carini per studenti, come per esempio il 7-segment display.

Per questo progetto in particolare utilizzo un programma chiamato "espresso" https://github.com/classabbyamp/espresso-logic per poter fare logic minimization in modo automatico e in parallelo su tutti i 7 segmenti.

Scrivetemi pure, nel caso siate interessati a saperne di più







...

Computazione

- Pascal e Pascalina
- Da computazione meccanica a digitale
- Astrazione
- Costruzione di circuiti a partire da tabelle di verità
- .
- Tutto può essere computato? Il computer è onnipotente?
- NO ☺
- Alcuni problemi non sono computabili
- per esempio Halting Problem
  Data una Macchina di Turing (che è un modello di computer) e un generico ingresso, non è possibile determinare se la macchina terminerà la computazione leggendo quell'ingresso
  oppure continuerà a computare per sempre senza fermarsi

Grazie !!