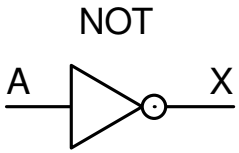
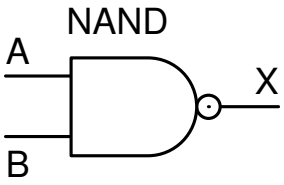
Porte Logiche e Circuiti Combinatori



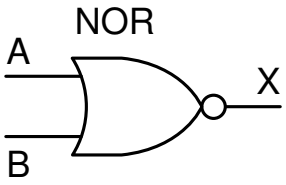
A	X
0	1
1	0

(a)



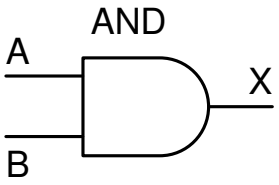
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)



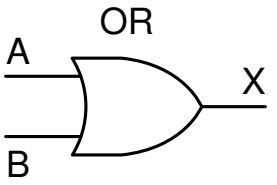
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)



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

(d)



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

(e)

Prof. Ivan Lanese

Algebra di Boole

- Abbiamo visto che gli elaboratori digitali usano valori binari {0,1}

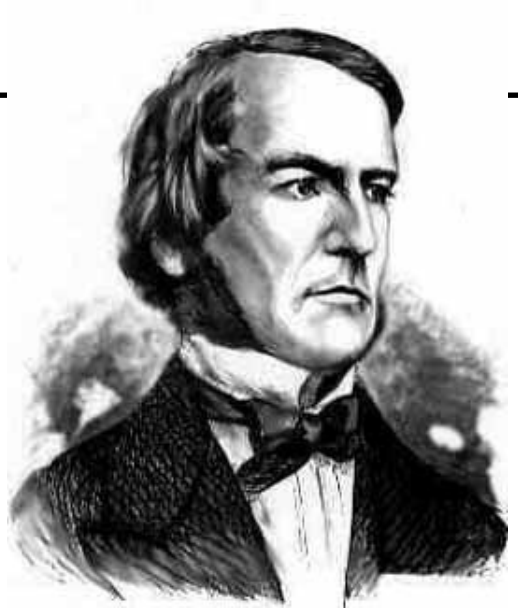
- L'algebra di Boole definisce una aritmetica per tali due valori

Gli operatori di tale algebra sono:

- Addizione (+ oppure OR):
  0+0=0; 0+1=1; 1+0=1; 1+1=1

- Moltiplicazione (• oppure AND):
  0•0=0; 0•1=0; 1•0=0; 1•1=1

- Negazione (barra sopra operando oppure NOT):
  $\overline{0}=1$; $\overline{1}=0$



George Boole, 1815-1864

("*A Calculus of Logic*")

Espressioni booleane e proprietà dell'algebra di Boole

- Un'espressione booleana si costruisce usando:
  - le costanti 0 e 1 (identificano i valori di verità "falso" e "vero")
  - gli operatori booleani
  - le variabili (A,B,C,x,y,...)
- Usiamo equivalenze fra espressioni booleane per descrivere le proprietà dell'algebra di Boole
- We also have not idempotence: $\overline{\overline{A}} = A$

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Funzioni booleane e tabelle di verità

- Una funzione booleana associa a delle variabili booleane in input un valore logico in output

- Un modo naturale per descrivere le funzioni booleane sono le tabelle di verità:

  - per ogni combinazione di possibili valori per le variabili in input, indicano il corrispondente valore di output

  - nell'esempio a fianco, F(A,B,C) è una funzione booleana a tre variabili A, B, C

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

Mintermini

- Un letterale è
  - una variabile (ad esempio A)
  - oppure una variabile negata (ad esempio $\overline{A}$)
- Un mintermine su n variabili è l'AND fra n letterali corrispondenti alle n variabili
- Ogni combinazione delle variabili di una funzione booleana ha un corrispondente mintermine (vero per quella specifica combinazione)

A	B	C	F	
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	0	$\overline{A}\,\overline{B}\,C$
0	1	0	1	$\overline{A}\,B\,\overline{C}$
0	1	1	0	$\overline{A}\,B\,C$
1	0	0	0	$A\,\overline{B}\,\overline{C}$
1	0	1	0	$A\,\overline{B}\,C$
1	1	0	1	$A\,B\,\overline{C}$
1	1	1	1	$A\,B\,C$

Espressioni booleane per descrivere funzioni

- Ogni funzione booleana può essere definita tramite un'espressione booleana basata sui soli operatori di base AND, OR, NOT

- In particolare esiste una forma "canonica":

  - considerare i mintermini corrispondenti alle combinazioni dell'input per cui la funzione è vera

A	B	C	F	
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	0	$\overline{A}\,\overline{B}\,C$
0	1	0	1	$\overline{A}\,B\,\overline{C}$
0	1	1	0	$\overline{A}\,B\,C$
1	0	0	0	$A\,\overline{B}\,\overline{C}$
1	0	1	0	$A\,\overline{B}\,C$
1	1	0	1	$A\,B\,\overline{C}$
1	1	1	1	$A\,B\,C$

  - fare l'OR di questi mintermini: $\overline{A}B\overline{C} + AB\overline{C} + ABC$

Esempio di rappresentazione canonica

Storiella: Ogni potenziale sospettato è contraddistinto dalla possibilità di avere oppure no un alibi (*a*), un motivo per commettere l'omicidio (*m*), ed una relazione con l'arma del delitto (*w*). La polizia decide di considerare come sospettati coloro che soddisfano l'espressione NOT(*a*) AND (*m* OR *w*)

Tabella di verità della funzione "sospettato" $s(a,m,w) = \overline{a} \cdot (m + w)$

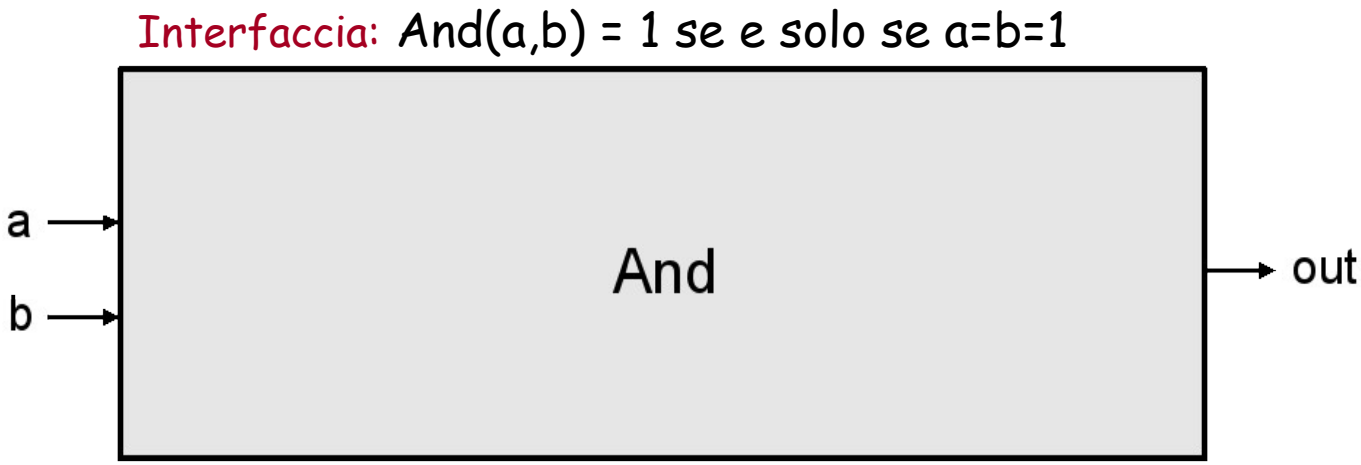
a	m	w	*minterm*	suspect(a,m,w)= not(a) and (m or w)
0	0	0	$m_0 = \overline{a}\,\overline{m}\,\overline{w}$	0
0	0	1	$m_1 = \overline{a}\,\overline{m}\,w$	1
0	1	0	$m_2 = \overline{a}\,m\,\overline{w}$	1
0	1	1	$m_3 = \overline{a}\,m\,w$	1
1	0	0	$m_4 = a\,\overline{m}\,\overline{w}$	0
1	0	1	$m_5 = a\,\overline{m}\,w$	0
1	1	0	$m_6 = a\,m\,\overline{w}$	0
1	1	1	$m_7 = a\,m\,w$	0

Forma canonica: $s(a,m,w) = \overline{a}\,\overline{m}\,w + \overline{a}\,m\,\overline{w} + \overline{a}\,m\,w$

Come implementare funzioni booleane

- Impareremo a costruire circuiti combinatori, cioè "scatole" che implementano funzioni booleane

- Ad esempio potremmo pensare ad un circuito che implementa la funzione: And(a,b) = a AND b

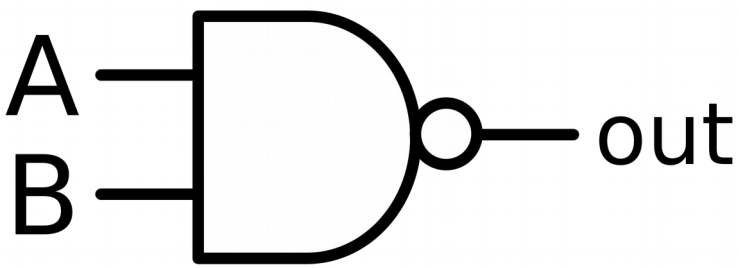


- Per costruire questi circuiti useremo dei "mattoncini" di base (dette porte logiche)

  - In particolare useremo la porta NAND che implementa la seguente funzione: Nand(a,b) = NOT(AND(a,b))

La porta logica NAND

- La porta logica NAND ha la seguente rappresentazione grafica e la seguente tabella di verità

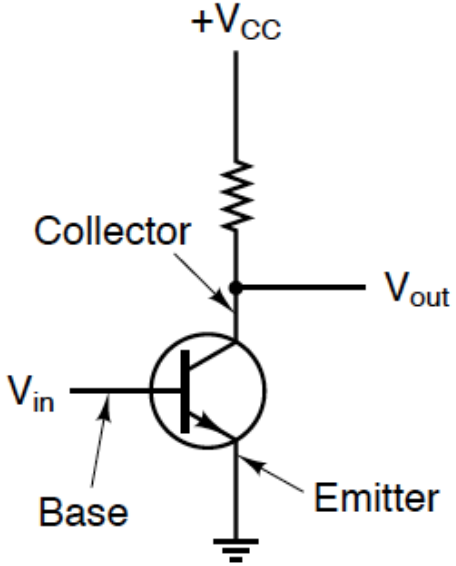


A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

- Si usa questa porta per due principali motivi:
  - Facile da realizzare fisicamente tramite transistor
  - Porta universale (si implementano AND, OR e NOT usando solo porte NAND)

Transistor

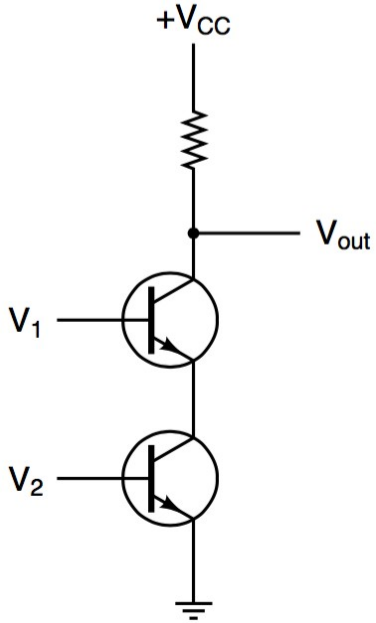
- Un transistor è un dispositivo a 3 connessioni: collettore, base, emettitore
  - se non c'è tensione sulla base, si comporta come una resistenza infinita fra collettore e emettitore
  - in presenza di tensione sulla base, si comporta da conduttore ideale tra collettore ed emettitore



- Il sistema in figura implementa un NOT
  - se non diamo tensione a V_{in}, la tensione V_{cc} viene trasmessa a V_{out}
  - se invece diamo tensione a V_{in}, non ci sarà tensione su V_{out} in quanto la tensione V_{cc} scarica a terra passando da collettore a emettitore

NAND: realizzazione tramite transistor

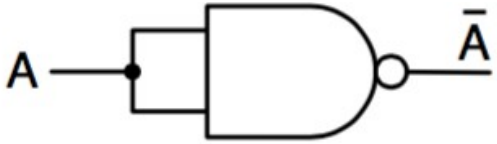
- Assumendo che la presenza di tensione rappresenti la costante booleana 1, mentre la sua assenza la costante booleana 0, il seguente dispositivo implementa la porta logica NAND

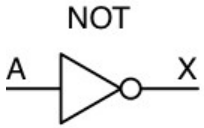


  - La tensione V_{cc} scarica a terra solo se entrambi gli input V_1 e V_2 sono sottoposti a tensione (in altri termini, l'output è uguale a 1 per ogni combinazione di input, ad esclusione del caso in cui entrambi gli input siano uguali a 1)

NAND: universalità

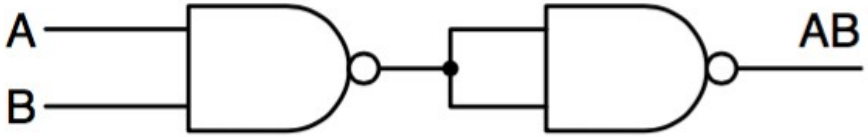
- Implementazione porta logica NOT:

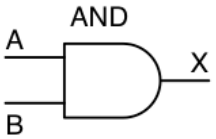




A	X
0	1
1	0

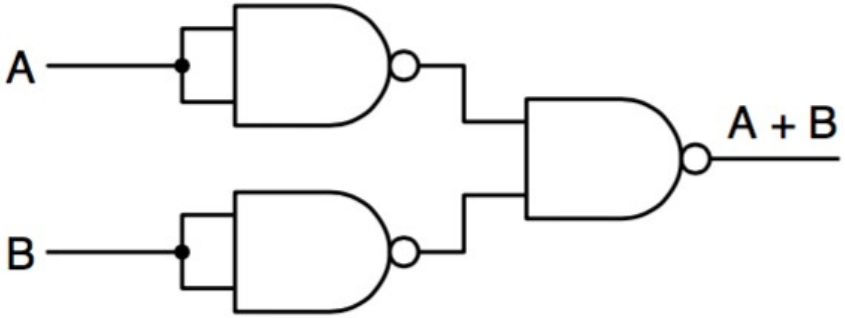
- Implementazione porta logica AND:

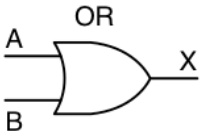




A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

- Implementazione porta logica OR:

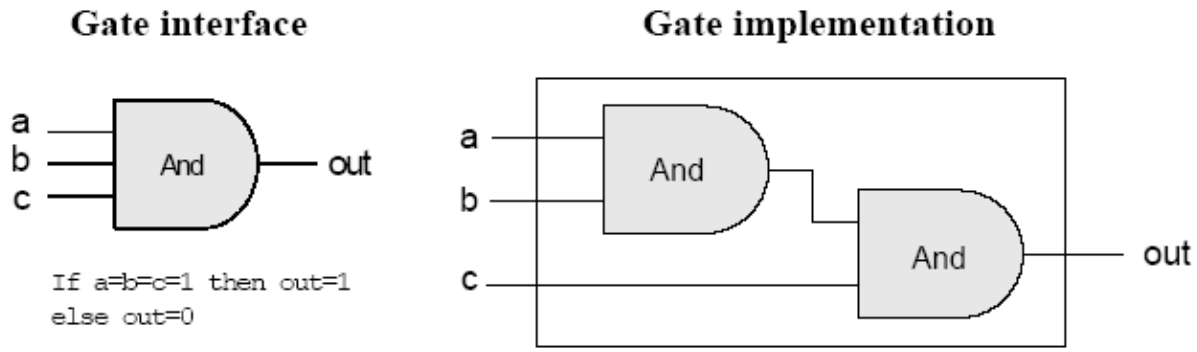




A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Implementazione altre funzioni booleane

- Per realizzare nuovi circuiti di solito si descrive prima l'interfaccia, ed in seguito si fornisce un'implementazione

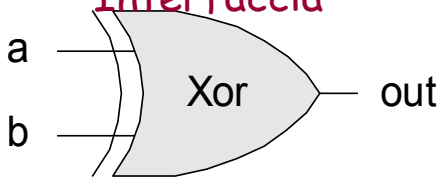


- Osservazione:
  - Possono esistere più implementazioni per la medesima interfaccia

Porta logica Xor

- La porta logica Xor ha la seguente interfaccia:

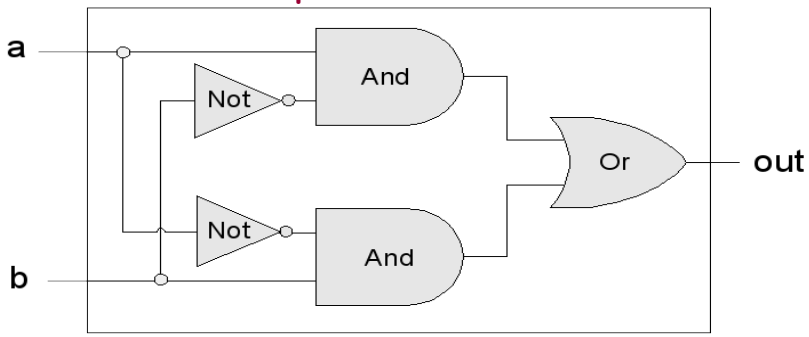
Interfaccia



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

- Eccone due implementazioni:

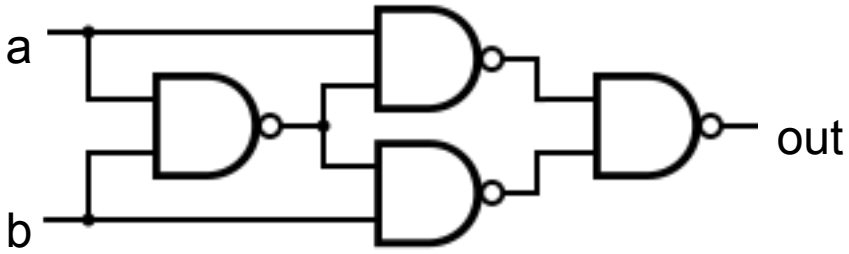
Implementazione



Xor(a,b) =
Or(And(a,Not(b)),And(Not(a),b))

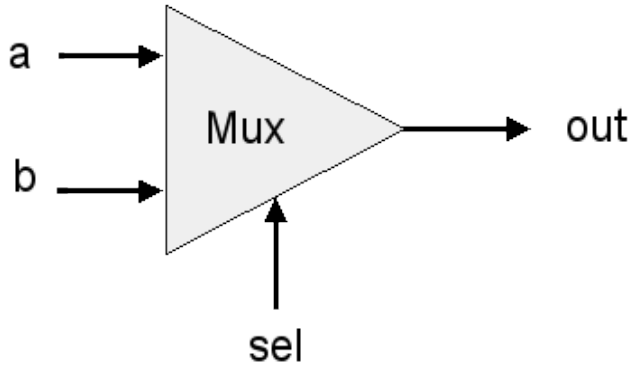
(forma canonica)

Implementazione



Un esercizio: multiplexer

a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



sel	out
0	a
1	b

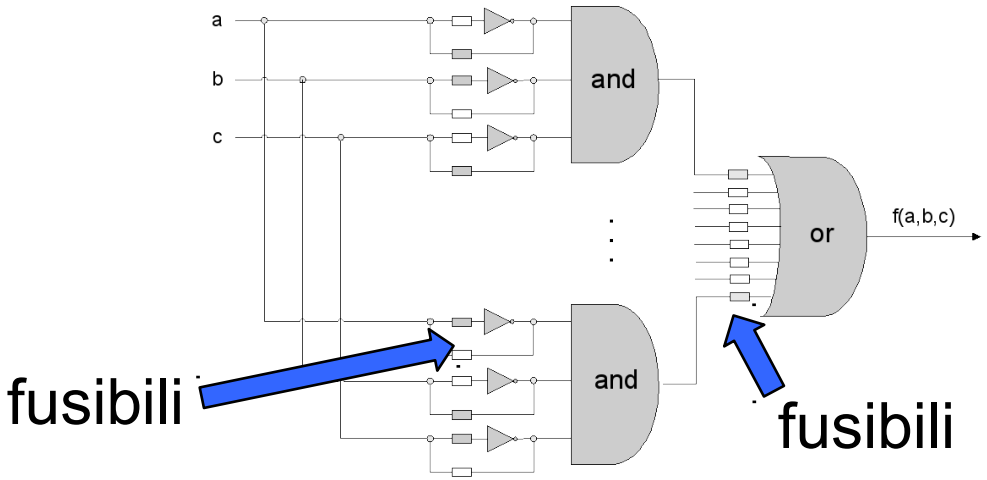
Progettare un'implementazione del multiplexer descritto sopra:
si consiglia l'uso delle porte logiche Not, And, Or

Array logico programmabile (PLA)

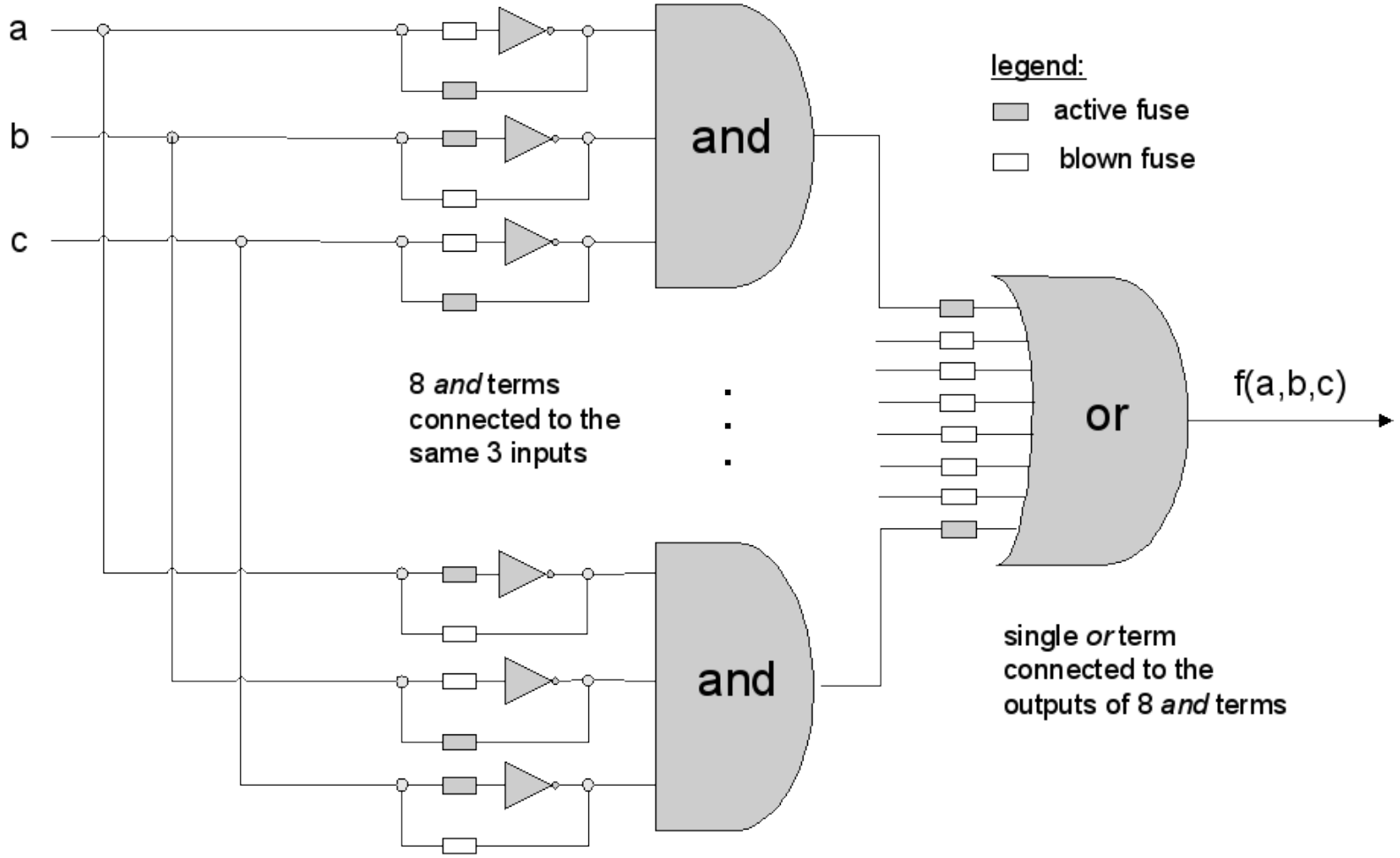
- Tutte le funzioni booleane hanno una forma canonica (OR fra i mintermini per cui l'output è vero)

- Array logici programmabili: circuiti universali pre-costruiti che data una quantità di input (3 nella figura) prevedono porte AND per tutti i possibili mintermini, con uscite che entrano in un OR

- E' possibile programmare qualsiasi funzione, semplicemente fondendo dei fusibili che interrompono il collegamento fra gli AND dei mintermini che non interessano, e l'OR finale



Funzione realizzata tramite PLA

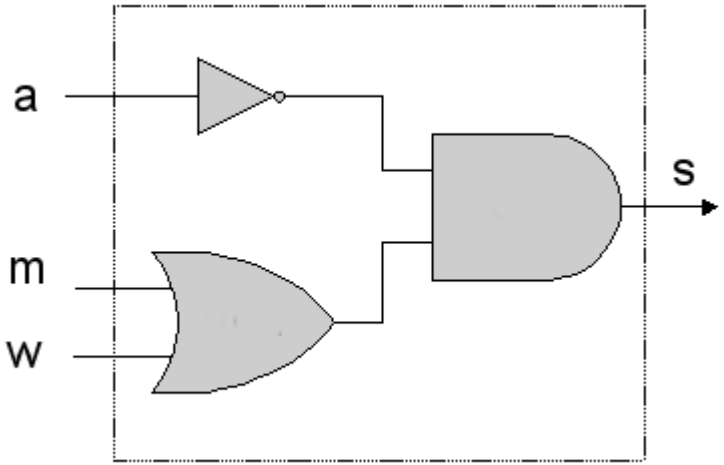


implementation of $f(a,b,c) = a\,\overline{b}\,c + \overline{a}\,b\,\overline{c}$

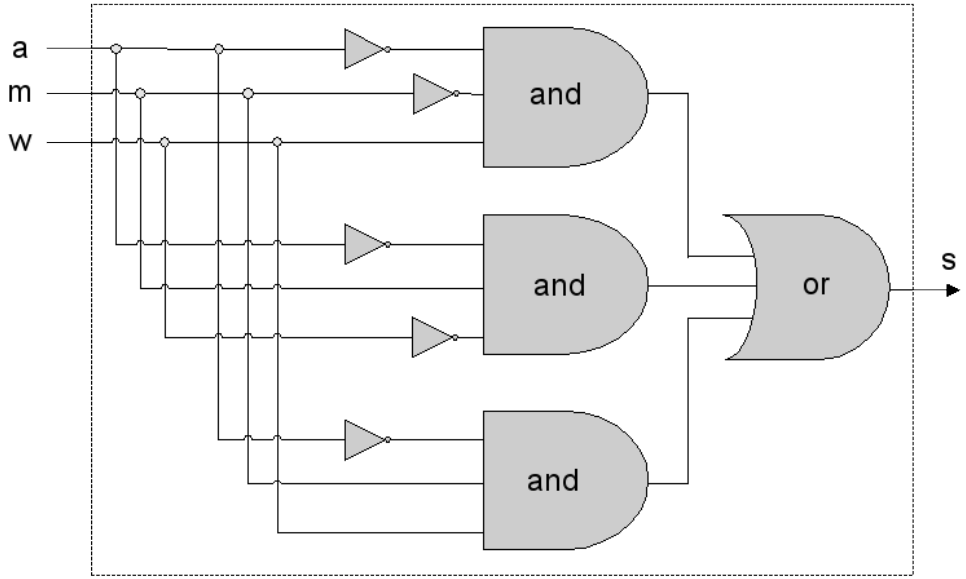
(the on/off states of the fuses determine which gates participate in the computation)

Esempio della funzione “sospettato”

$$s(a,m,w) = \overline{a} \cdot (m + w)$$



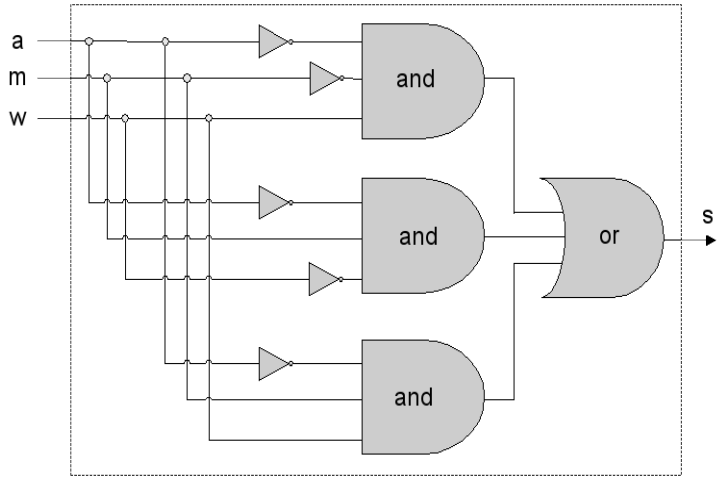
$$s(a,m,w) = \overline{a}\,\overline{m}\,w + \overline{a}\,m\,\overline{w} + \overline{a}\,m\,w$$



Ancora sulla funzione “sospettato”

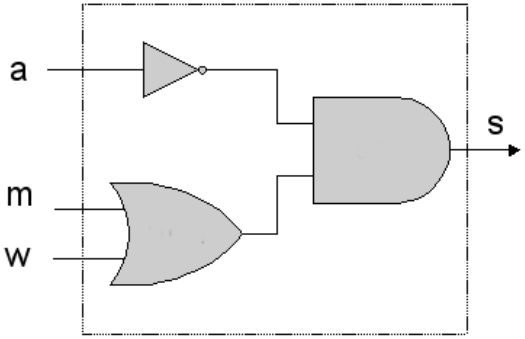
- La forma canonica è utile se si vuole usare un PLA…

$$s(a,m,w) = \bar{a}\,\bar{m}\,w + \bar{a}\,m\,\bar{w} + \bar{a}\,m\,w$$



- …ma la seguente forma equivalente usa meno porte

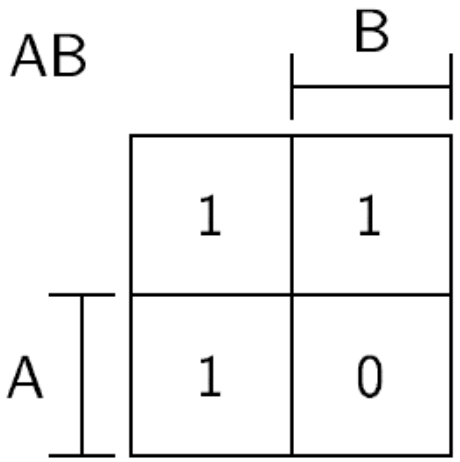
$$s(a,m,w) = \bar{a} \cdot (m + w)$$



Mappe di Karnaugh

- Le mappe di Karnaugh sono un modo per rappresentare funzioni booleane (simili alle tabelle di verità)..
  - ..ma tale rappresentazione permette di costruire un circuito combinatorio "minimale"
- Mappa di Karnaugh:
  - tabella bidimensionale con una cella per ogni possibile mintermine
  - due celle adiacenti differiscono per un solo letterale (codice Grey)
  - nella cella si mette 1 se la funzione booleana vale 1, si mette 0 altrimenti

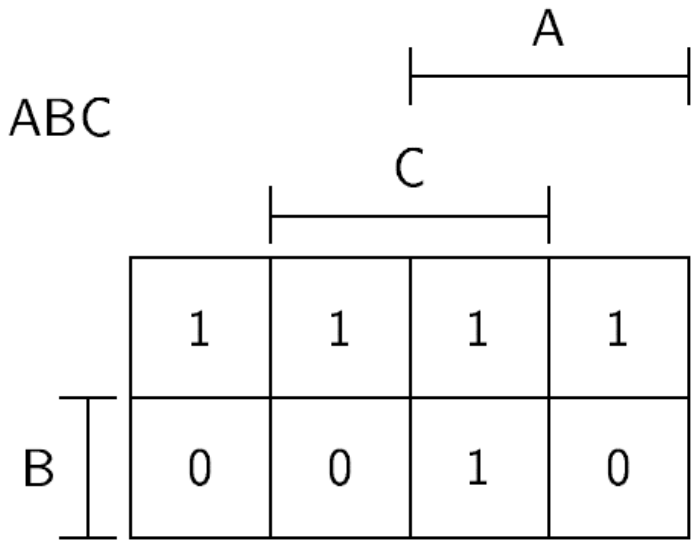
Esempio di Mappa di Karnaugh per funzione a 2 variabili



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

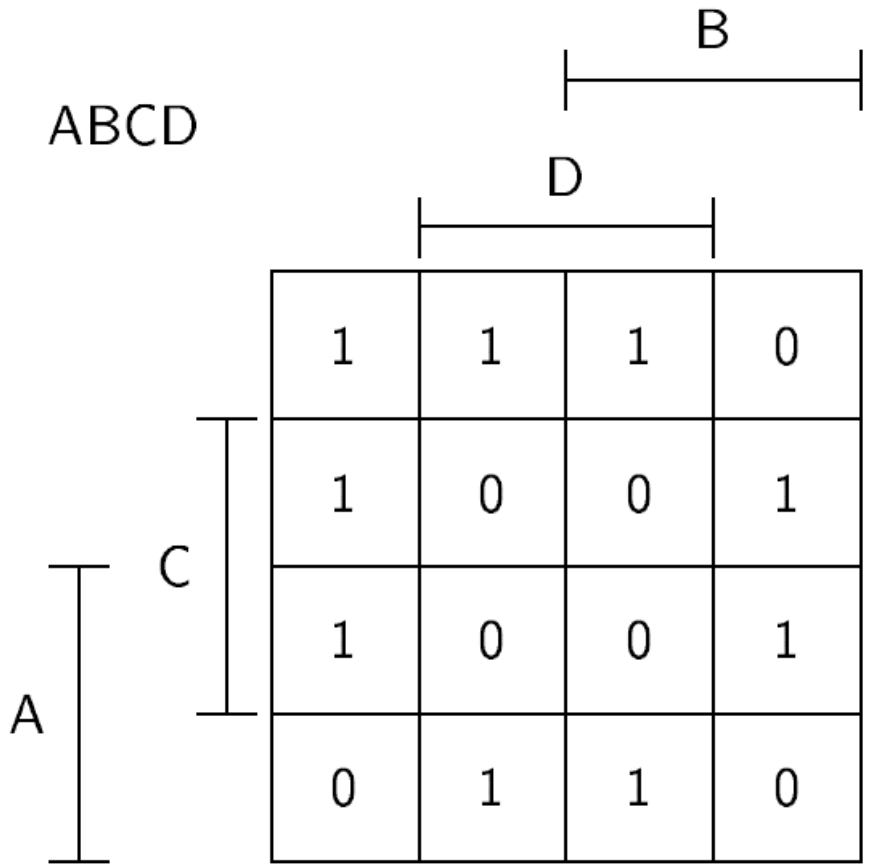
- Le indicazioni A (rispettivamente B) a fianco della mappa indicano righe (rispettivamente colonne) contenenti i mintermini con il letterale A (rispettivamente B)

Esempio di Mappa di Karnaugh per funzione a 3 variabili



A	*B*	*C*	*F*
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Esempio di Mappa di Karnaugh per funzione a 4 variabili



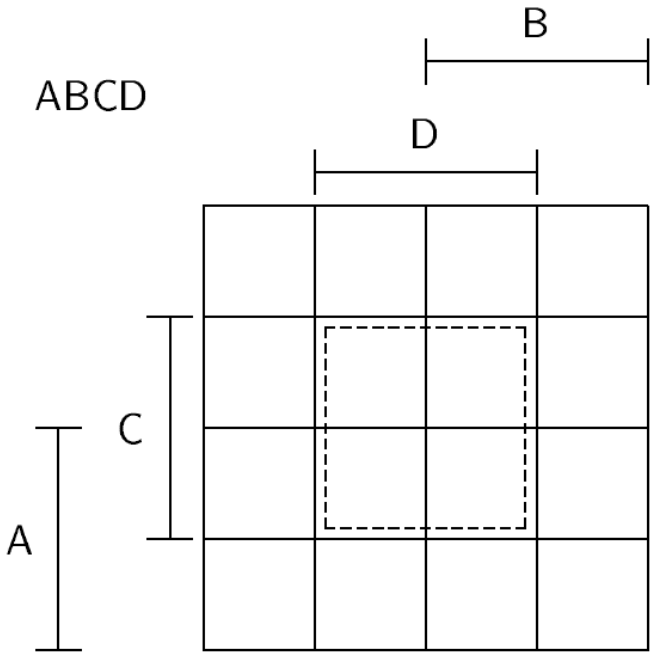
Raggruppamenti in Mappe di Karnaugh

- Esistono blocchi di celle, detti raggruppamenti, corrispondenti a tutti i mintermini che contengono alcuni letterali

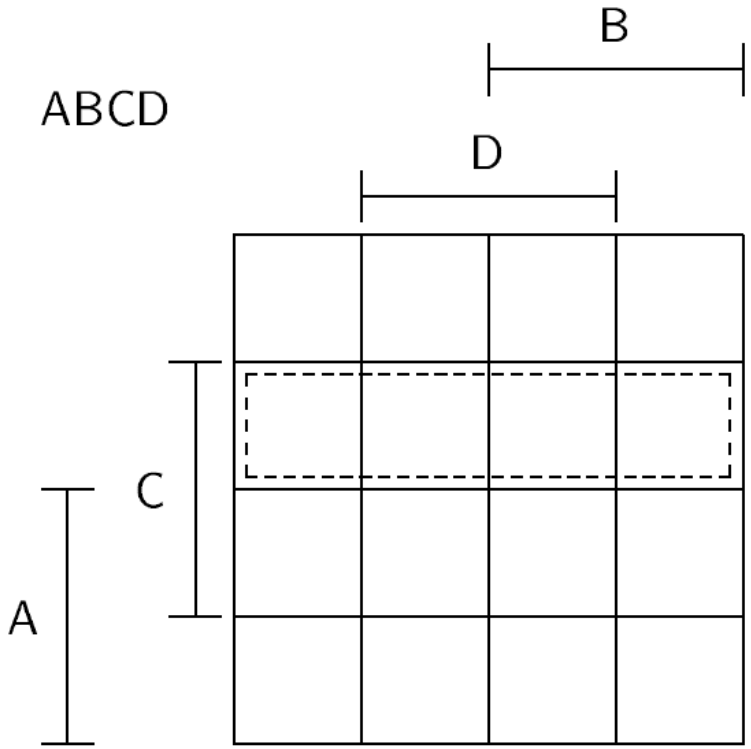
- Esempio: il raggruppamento riportato sotto coincide con i quattro mintermini in cui appaiono i letterali C e D

  - Tale raggruppamento viene identificato con il prodotto CD

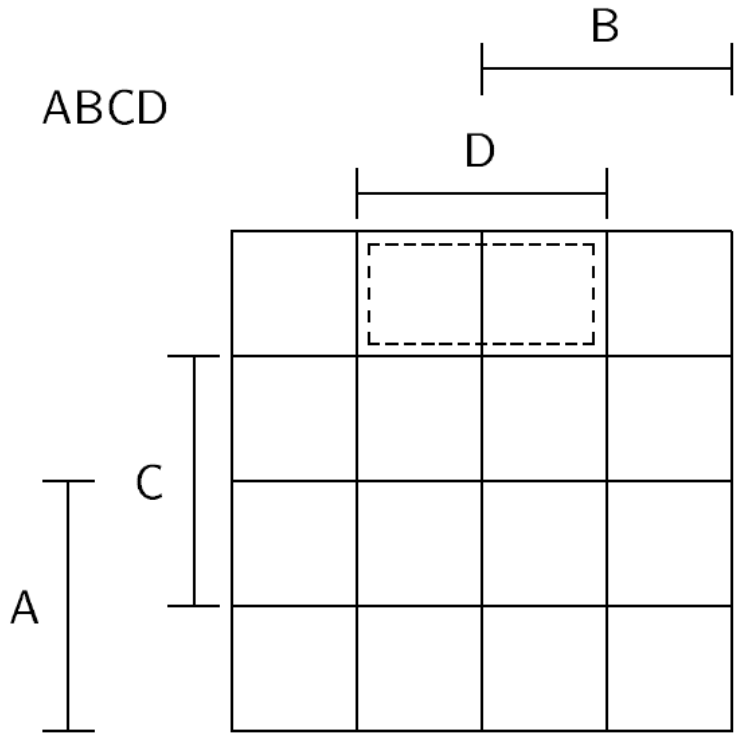
  - CD è infatti l'espressione che definisce la funzione corrispondente alla mappa qui a fianco assumendo un 1 nelle celle del raggruppamento, e uno 0 in tutte le altre celle



Altri esempi di raggruppamenti

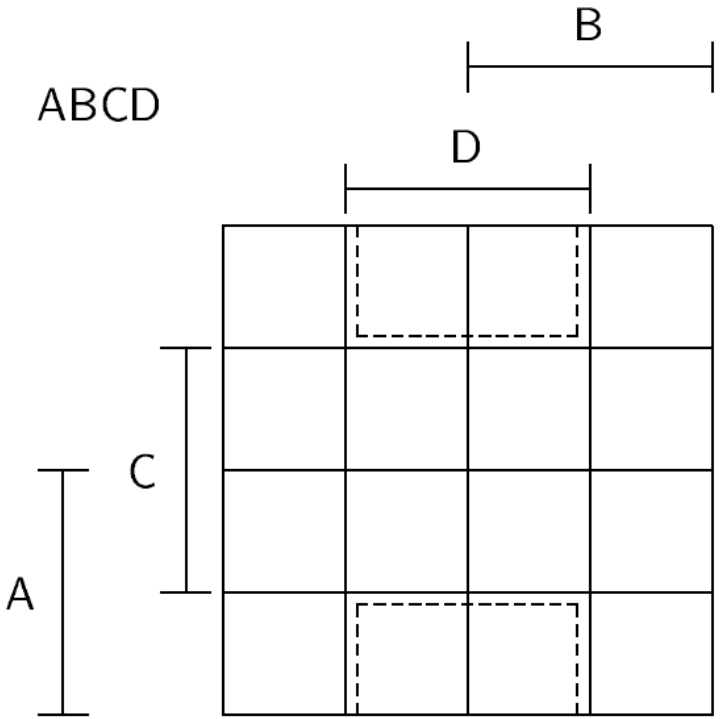


Prodotto Corrispondente: $\overline{A}C$

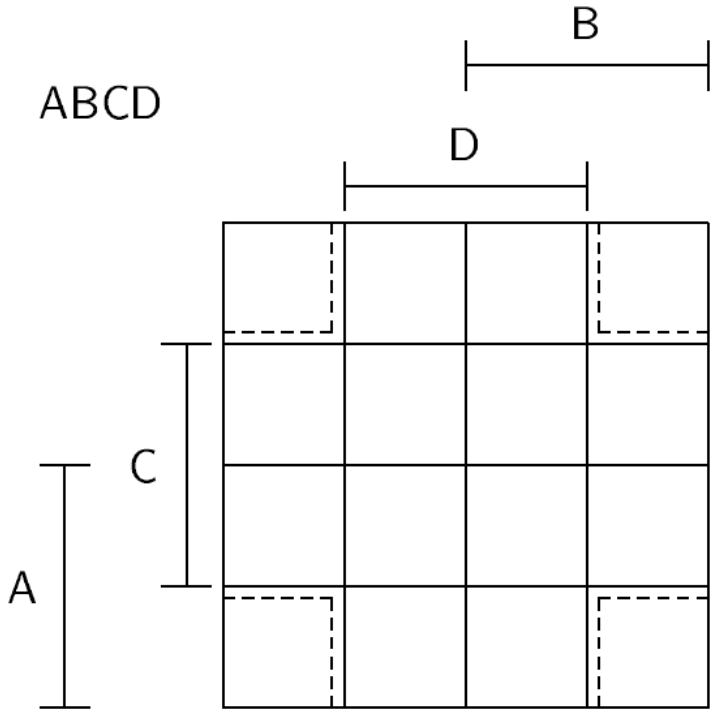


Prodotto Corrispondente: $\overline{A}\,\overline{C}D$

Altri esempi di raggruppamenti (continua)



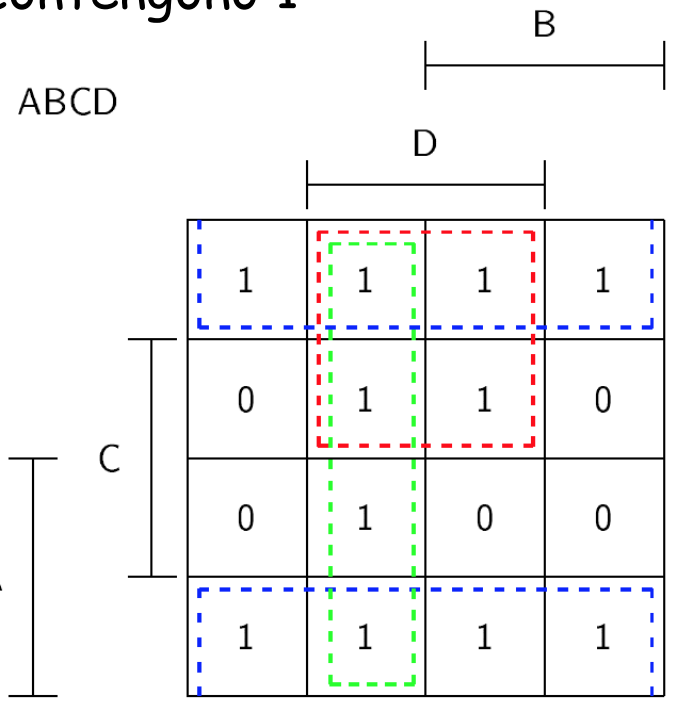
Prodotto Corrispondente: $\overline{C}D$



Prodotto Corrispondente: $\overline{C}\,\overline{D}$

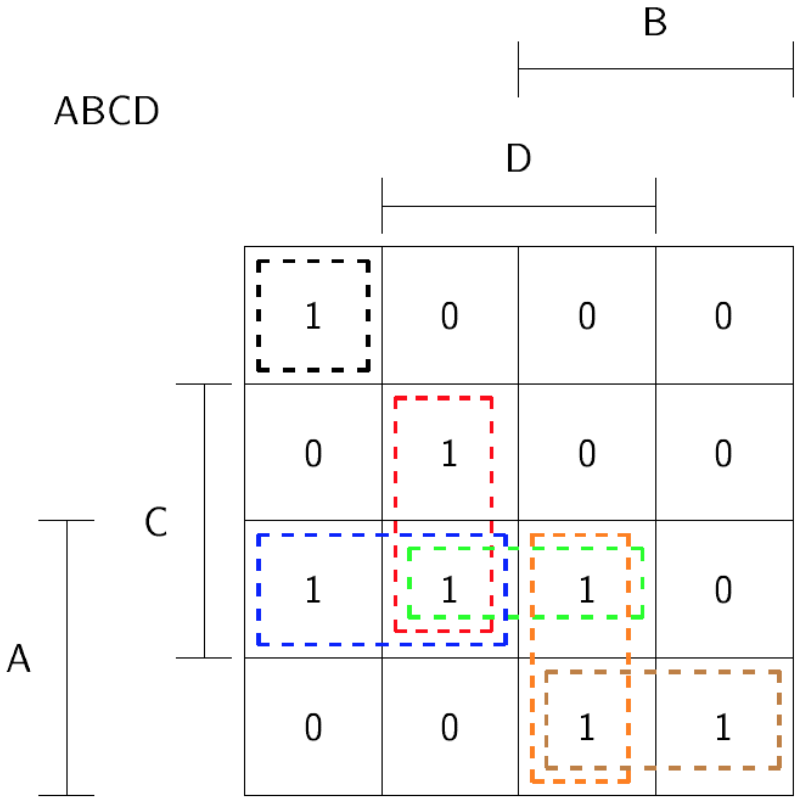
Funzione booleana per mappe di Karnaugh

- E' possibile esprimere una funzione descritta da una mappa di Karnaugh nel modo seguente:
  - identificare una copertura, cioè dei raggruppamenti tali che:
    - tutte le celle dei raggruppamenti contengono 1
    - le celle al di fuori di tali raggruppamenti contengono 0
  - considerare l'OR fra i prodotti corrispondenti a tali raggruppamenti (come nell'esempio qui a fianco)



Somma di Prodotti: $\overline{C} + \overline{B}D + \overline{A}D$

Altro esempio



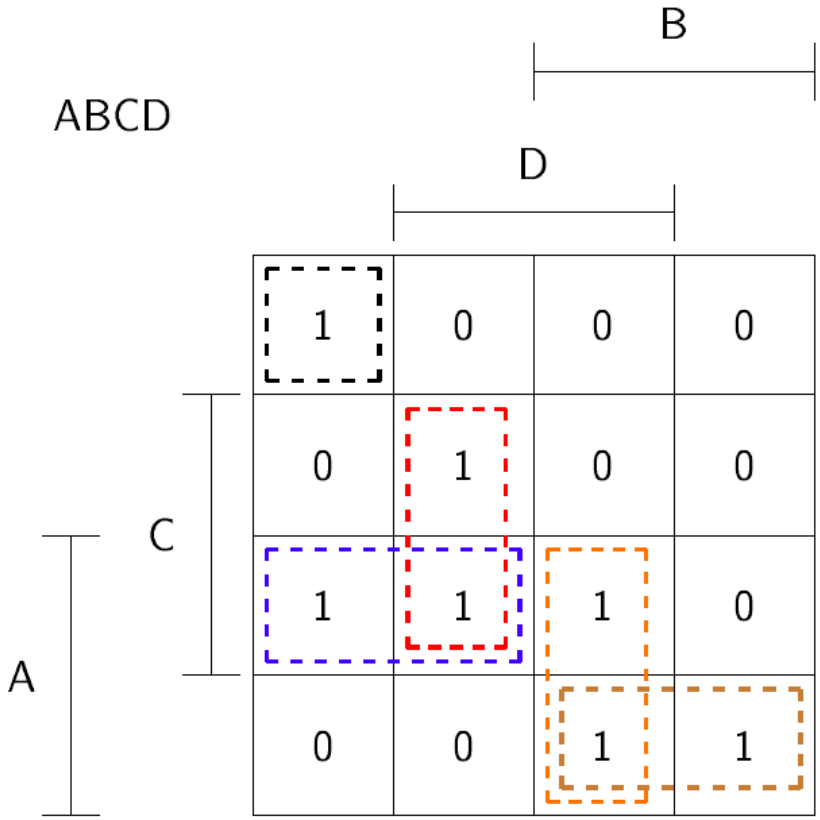
- La figura riporta una copertura, ma tale copertura non è minimale
  - le celle del raggruppamento verde sono già incluse in altri raggruppamenti (stessa cosa per il raggruppamento arancione)

Copertura minimale

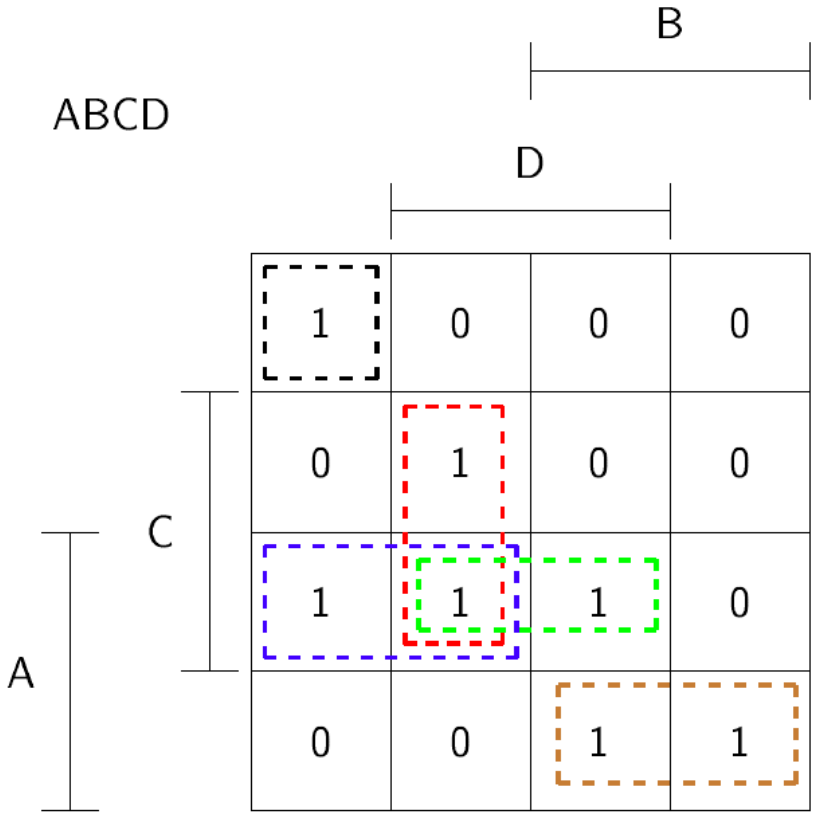
- Per essere minimale una copertura deve essere composta da:
  - raggruppamenti che non sono contenuti in potenziali raggruppamenti più grandi
  - raggruppamenti che contengono almeno una cella che non appare anche in altri raggruppamenti della copertura
- L'espressione booleana corrispondente ad una copertura minimale risulta essere un'espressione del tipo somma di prodotti di letterali (in altri termini OR fra AND di letterali) con un numero minimale di addendi

Copertura minimale (esempio)

- Le forme minimali per l'ultimo esempio sono due:



$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C + \overline{B}CD + AB\overline{C} + ABD$$



$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C + \overline{B}CD + AB\overline{C} + ACD$$

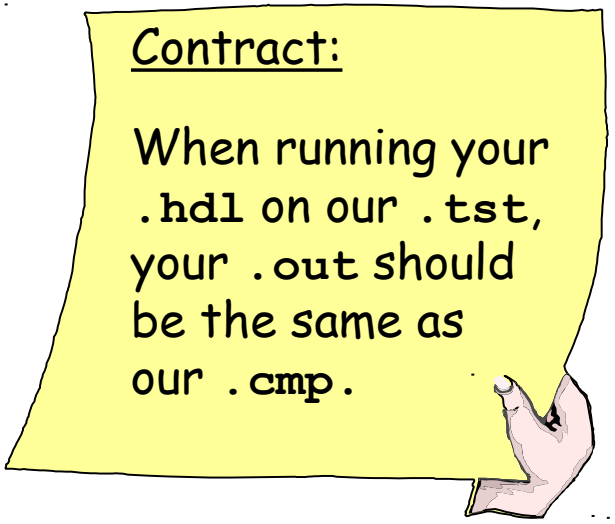
HDL e Hardware Simulator

- Sperimenteremo la progettazione di circuiti combinatori seguendo l'approccio del progetto: www.nand2tetris.org
  - I circuiti vengono definiti tramite il linguaggio HDL (Hardware Description Language)
  - Il comportamento dei circuiti descritti in HDL vengono simulati tramite un programma chiamato HardwareSimulator
- Nelle prossime slide vedremo come usare il linguaggio HDL ed il programma HardwareSimulator

Costruzione di una porta logica **And**

And.cmp

a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



And.hdl

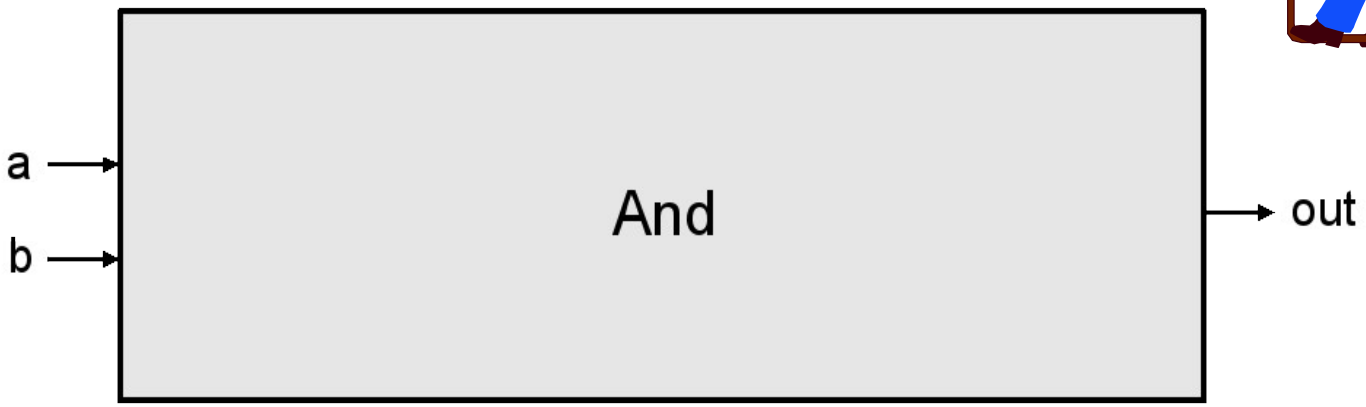
```
CHIP And
{    IN  a, b;
     OUT out;
     // implementation missing
}
```

And.tst

```
load And.hdl,
output-file And.out,
compare-to And.cmp,
output-list a b out;
set a 0,set b 0,eval,output;
set a 0,set b 1,eval,output;
set a 1,set b 0,eval,output;
set a 1, set b 1, eval, output;
```

Costruzione di una porta logica **And**

Interfaccia: And(a,b) = 1 se e solo se a=b=1



And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
     // implementation missing
}
```

Costruzione di una porta logica **And**

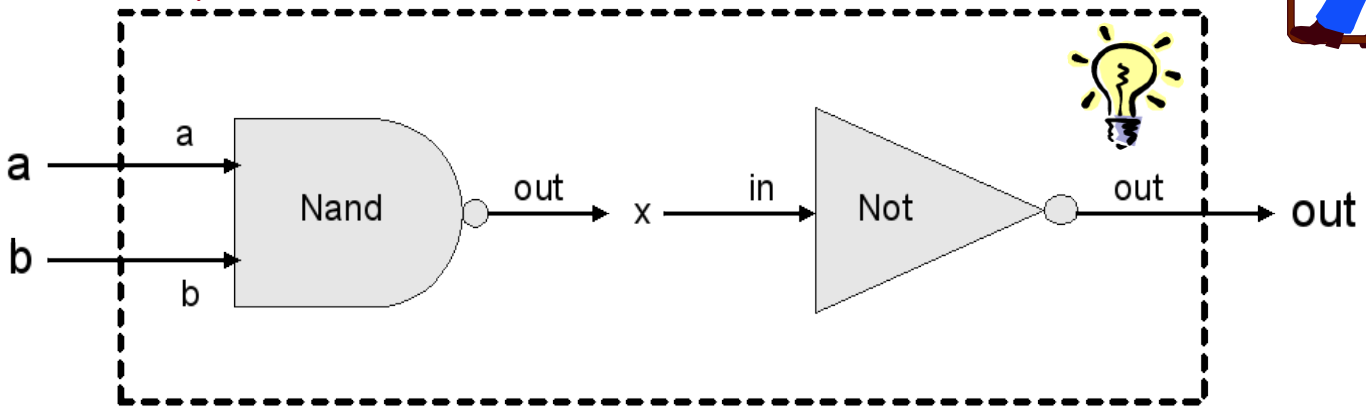
Implementazione: And(a,b) = Not(Nand(a,b))

And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
     // implementation missing
}
```

Costruzione di una porta logica **And**

Implementazione: And(a,b) = Not(Nand(a,b))

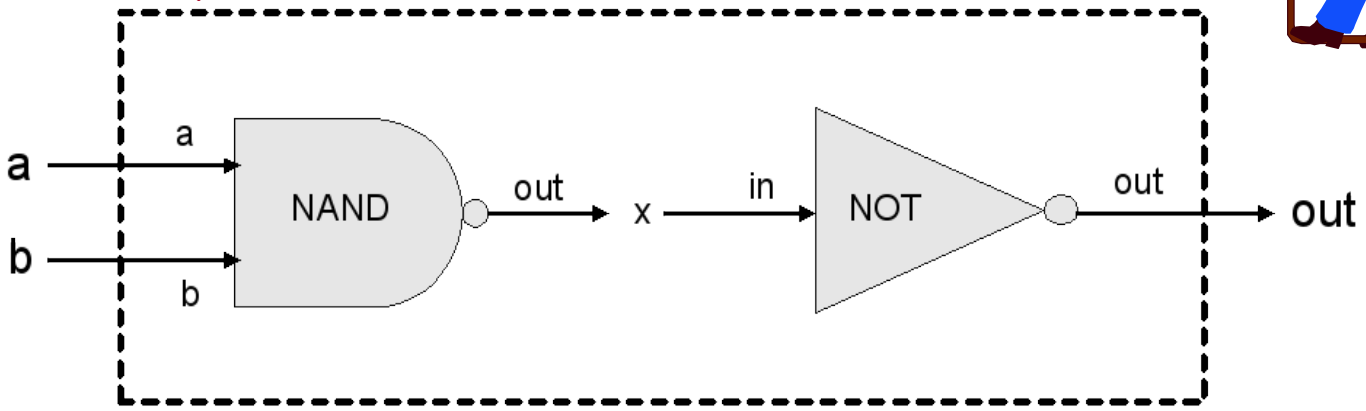


And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
     // implementation missing
}
```

Costruzione di una porta logica **And**

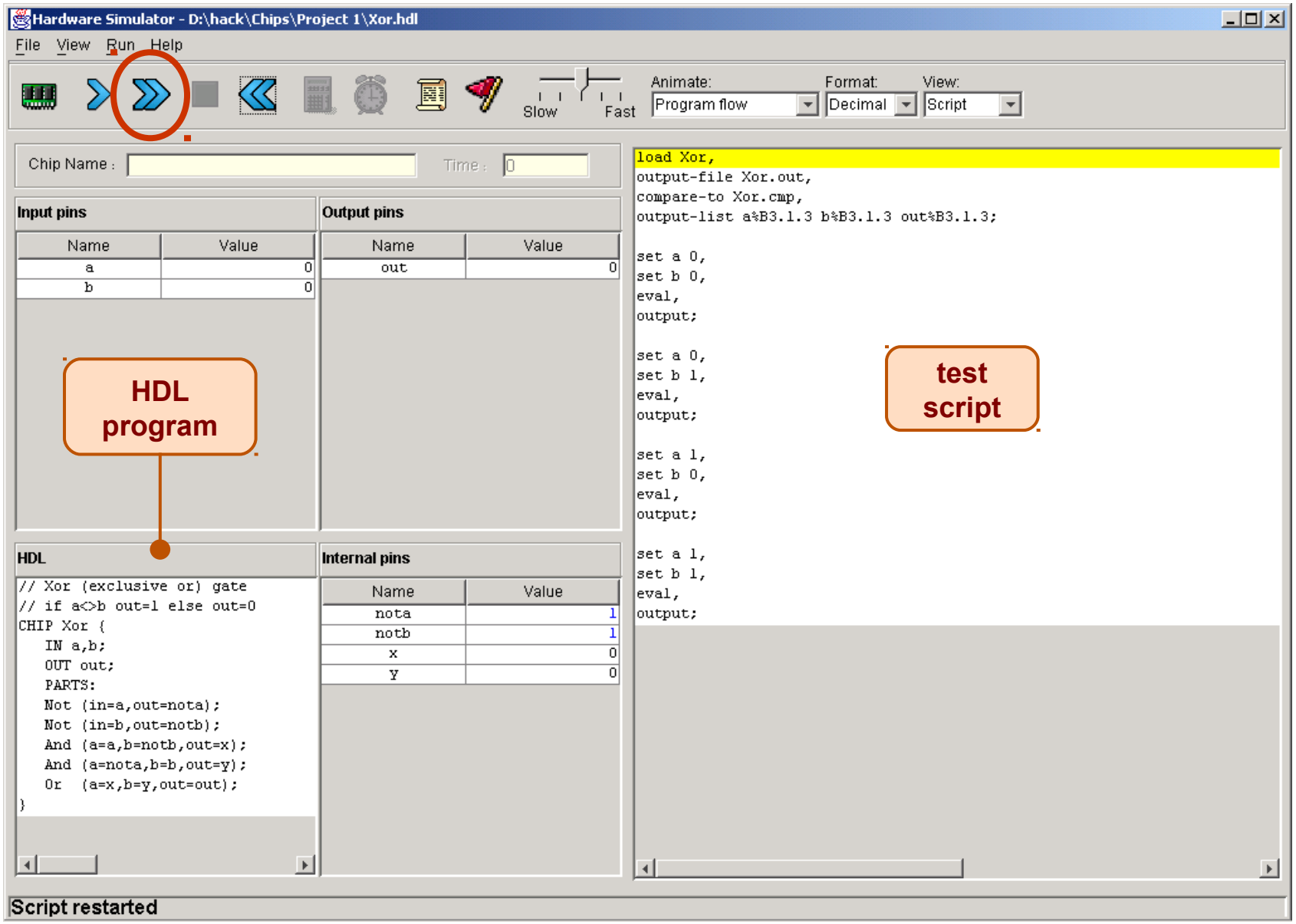
Implementazione: And(a,b) = Not(Nand(a,b))



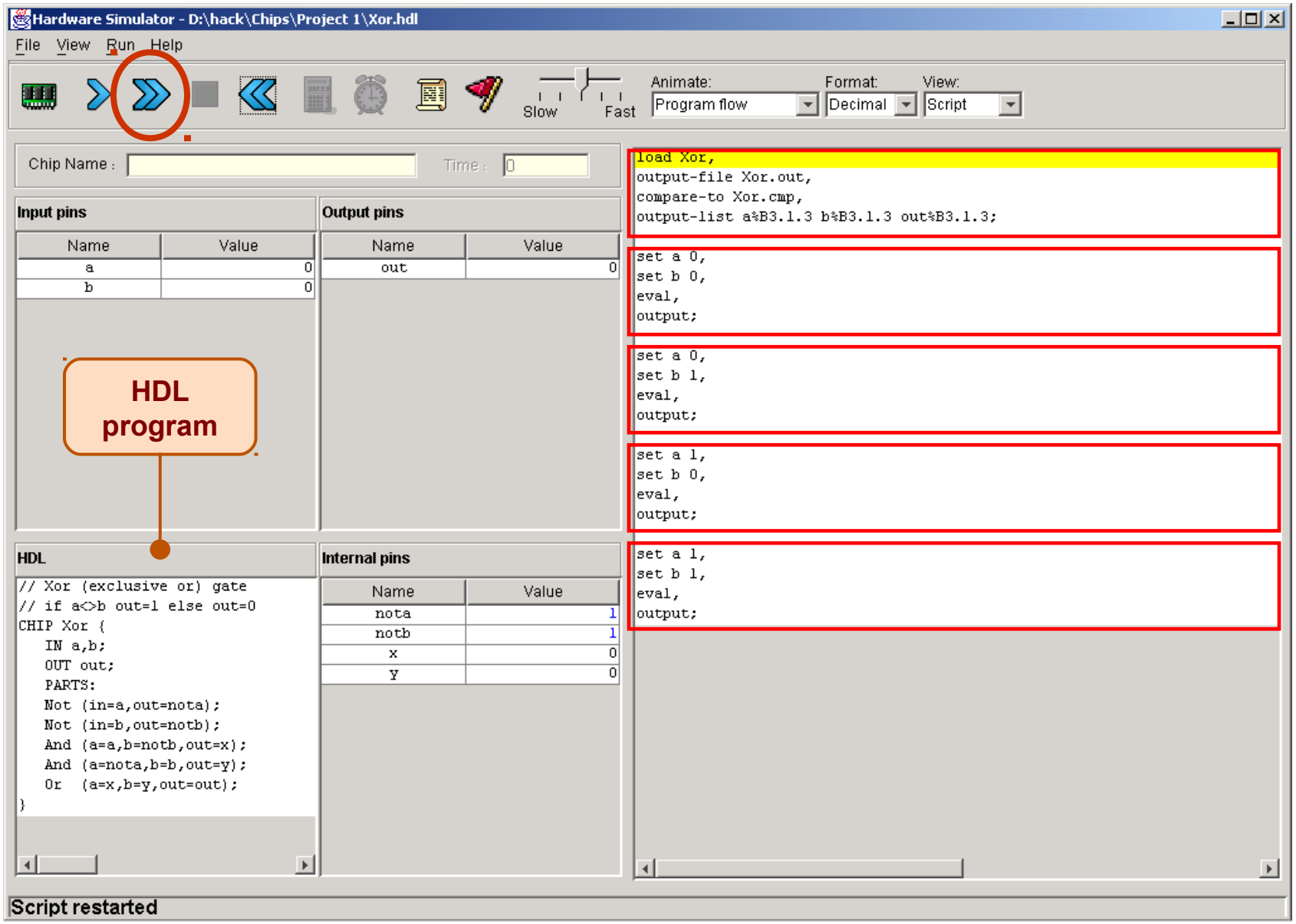
And.hdl

```
CHIP And
{    IN  a, b;
     OUT out;
     Nand(a = a,
          b = b,
          out = x);
     Not(in = x, out = out);
}
```

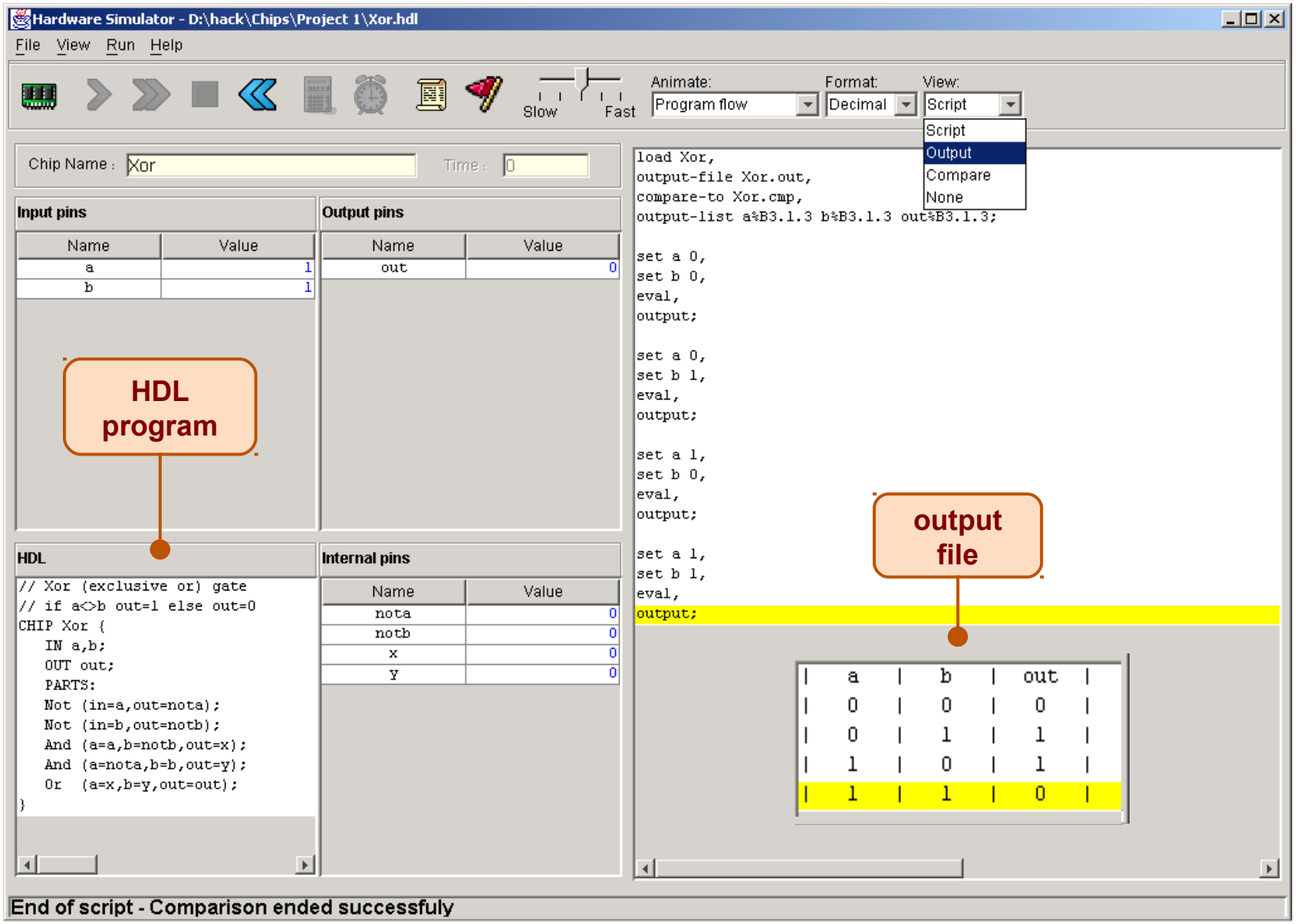
Hardware simulator



Hardware simulator

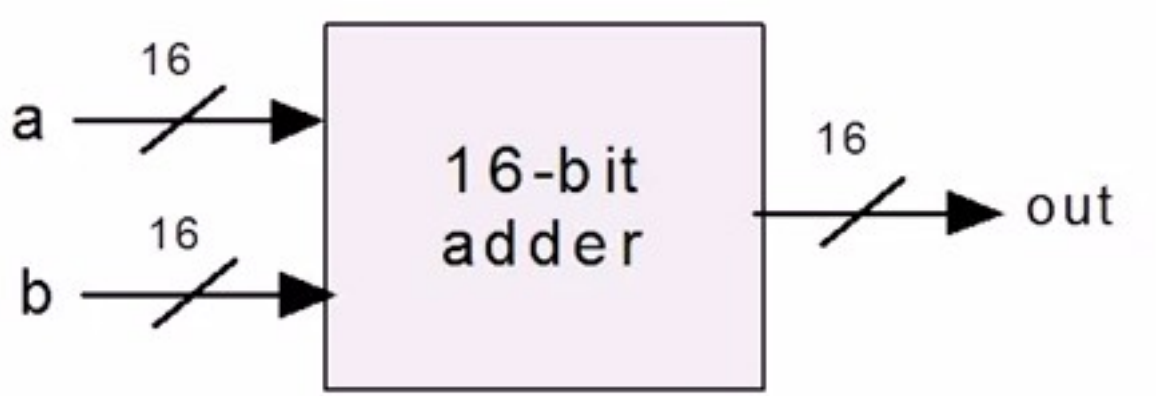


Hardware simulator



Bus multi-bit

- I circuiti combinatori a volte sono collegati a connessioni composte da più bit
  - Tali connessioni sono dette "bus"
  - Nell'immagine sotto è riportato un circuito con 32 ingressi e 16 uscite
    - Gli ingressi sono organizzati in 2 bus ("a" e "b") da 16 bit l'uno
    - Le uscite sono organizzate in un bus ("out") da 16 bit



Bus multi-bit in HDL

- Nel linguaggio HDL i bus vengono specificati indicando il nome del bus seguito dal suo numero di bit fra parentesi quadre

```
/*
 * Adds two 16-bit values.
 */
CHIP Add16 {
   IN a[16], b[16];
   OUT out[16];

   PARTS:
       …
}
```

  - Nell'esempio dichiariamo un circuito che somma due numeri da 16 bit dati in input tramite due bus da 16 bit ("a" e "b") e risultato dato in output tramite un bus da 16 bit ("out")

Bus multi-bit in HDL (esempi)

- Assumiamo ora di progettare un circuito che somma 3 numeri dati in input tramite 3 bus da 16 bit l'uno
  - Ecco la sua interfaccia:

```
/*
 * Adds three 16-bit values.
 */
CHIP Add3Way16 {
   IN first[16], second[16], third[16];
   OUT out[16];

   PARTS:

        // Put your code here:

}
```

Bus multi-bit in HDL (esempi)

- Ora lo implementiamo assumendo di usare due adder del tipo visto in precedenza con (2 bus da 16 input in ingresso)

```
/*
 * Adds three 16-bit values.
 */
CHIP Add3Way16 {
   IN first[16], second[16], third[16];
   OUT out[16];

   PARTS:

     Add16(a=first, b=second, out=temp);
     Add16(a=temp, b=third, out=out);
}
```

  - "temp" è anch'esso un bus a 16 bit, i cui bit si collegano bit-per-bit a quelli dell'"out" del primo adder e all'ingresso "a" del secondo adder

Bus multi-bit in HDL (esempi)

- E' possibile fare riferimento ai singoli bit di un bus usando il nome del bus e mettendo fra parentesi quadre l'identificativo del bit
  - Se un bus ha *n* bit, gli identificativi dei singoli bit vanno da 0 a *n*-1

```
/*
 * ANDs together all 4 bits of the input.
 */
CHIP And4Way {
   IN a[4];
   OUT out;

   PARTS:
       AND(a=a[0], b=a[1], out=t01);
       AND(a=t01, b=a[2], out=t012);
       AND(a=t012, b=a[3], out=out);
}
```

  - L'esempio riporta un'implementazione di un circuito combinatorio con un bus di 4 bit in ingresso, ed in uscita il risultato dell'AND fra i 4 bit in ingresso

Bus multi-bit in HDL (esempi)

- Nell'esempio qui sotto si progetta invece un circuito con due bus da 4 bit in ingresso e ed un bus da 4 bit in uscita
  - In uscita si propone il risultato dell'AND bit-a-bit

```
/*
 * Computes a bit-wise and of its two 4-bit
 * input buses
 */
CHIP And4 {
   IN a[4], b[4];
   OUT out[4];

   PARTS:
       AND(a=a[0], b=b[0], out=out[0]);
       AND(a=a[1], b=b[1], out=out[1]);
       AND(a=a[2], b=b[2], out=out[2]);
       AND(a=a[3], b=b[3], out=out[3]);
}
```

Bus multi-bit in HDL (esempi)

- E' possibile decomporre un bus multi-bit in sotto-bus, ognuno composto da un sotto-gruppo di bit
  - Nell'esempio qui sotto si considerano due bus da 8 bit in ingresso: "lsb" per i bit meno significativi (least significant bits) e "msb" per i bit più significativi (most significant bits)
    - Assumiamo che siano due byte che rappresentano una "word" da 16 bit
  - Per dare in input la intera "word" all'ingresso "a" di un adder con input a 16 bit si decompone "a" in due sotto-bus da un byte l'uno: "a[0..7]" e "a[8..15]"

```
…
IN lsb[8], msb[8], …
…
Add16(a[0..7]=lsb, a[8..15]=msb, b=…, out=…);
```