Chapter 1:
Boolean Logic

Boolean algebra

Some elementary Boolean operators:

- Not(x)
- And(x,y)
- Or(x,y)
- Nand(x,y)

x	Not(x)
0	1
1	0

x	y	And(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

x	y	Or(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

x	y	Nand(x,x)
0	0	1
0	1	1
1	0	1
1	1	0

Boolean functions:

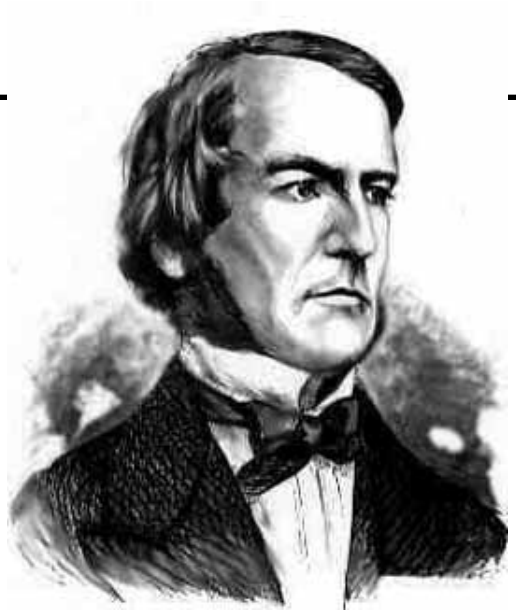
x	y	z	$f(x,y,z)=(x+y)\bar{z}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- Functional expression VS truth table expression
- Important result: Every Boolean function can be expressed using And, Or, Not.

All Boolean functions of 2 variables

Function	x	0	0	1	1
	y	0	1	0	1
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \overline{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\overline{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \overline{y} + \overline{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x + y}$	1	0	0	0
Equivalence	$x \cdot y + \overline{x} \cdot \overline{y}$	1	0	0	1
Not y	$\overline{y}$	1	0	1	0
If y then x	$x + \overline{y}$	1	0	1	1
Not x	$\overline{x}$	1	1	0	0
If x then y	$\overline{x} + y$	1	1	0	1
Nand	$\overline{x \cdot y}$	1	1	1	0
Constant 1	1	1	1	1	1

Boolean algebra



George Boole, 1815-1864

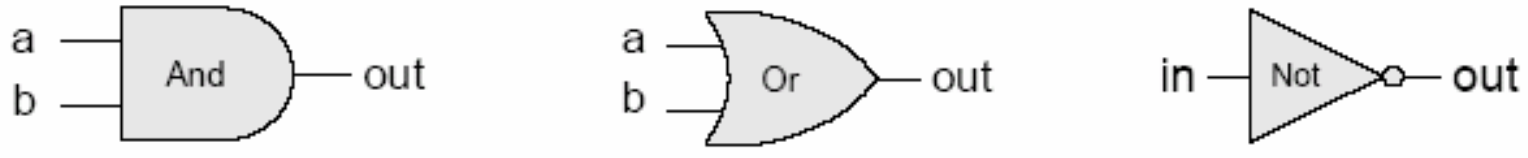
("*A Calculus of Logic*")

Given: **Nand(a,b), false**

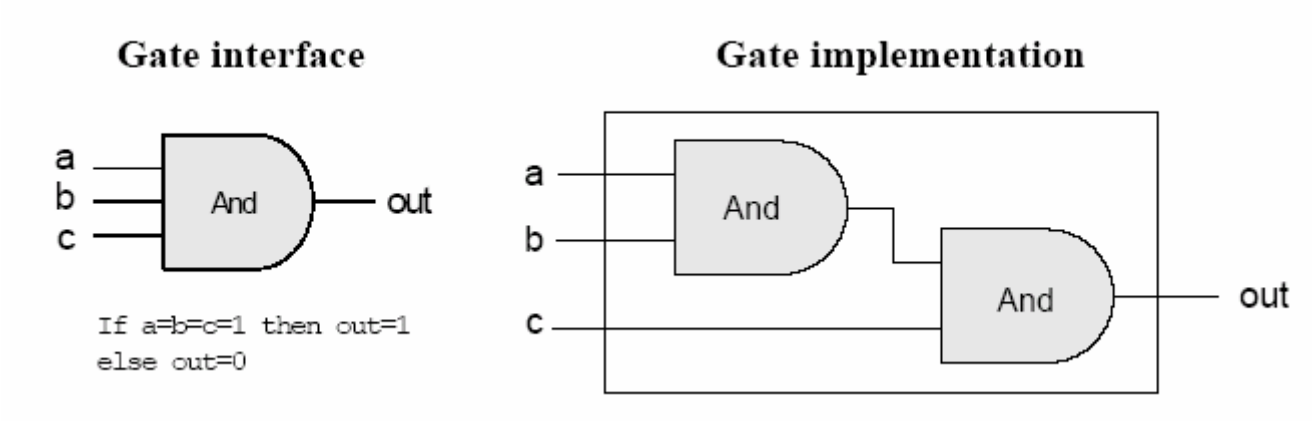
- **Not(a) = Nand(a,a)**
- **true = Not(false)**
- **And(a,b) = Not(Nand(a,b))**
- **Or(a,b) = Not(And(Not(a),Not(b)))**
- **Xor(a,b) = Or(And(a,Not(b)),And(Not(a),b)))**
- Etc.

Gate logic

- Gate logic – a gate architecture designed to implement a Boolean function
- Elementary gates:



- Composite gates:



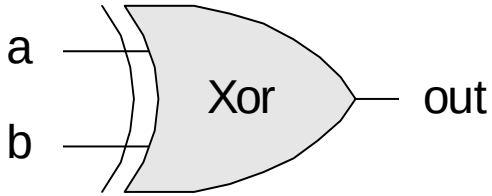
- Interface VS implementation.

Gate Logic

Claude Shannon, 1916-2001

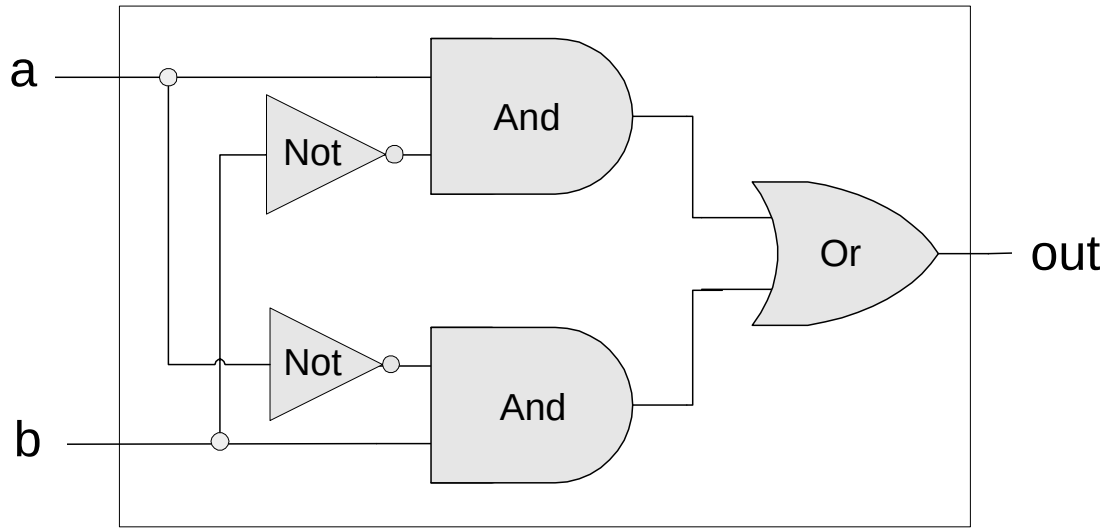
("Symbolic Analysis of Relay and Switching Circuits")

Interface



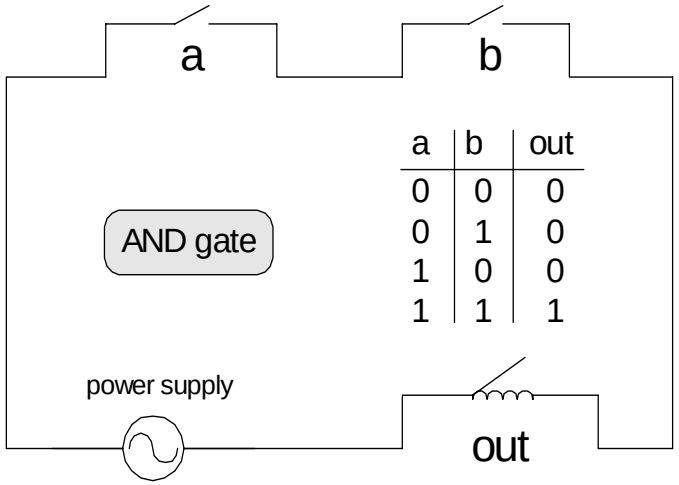
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation

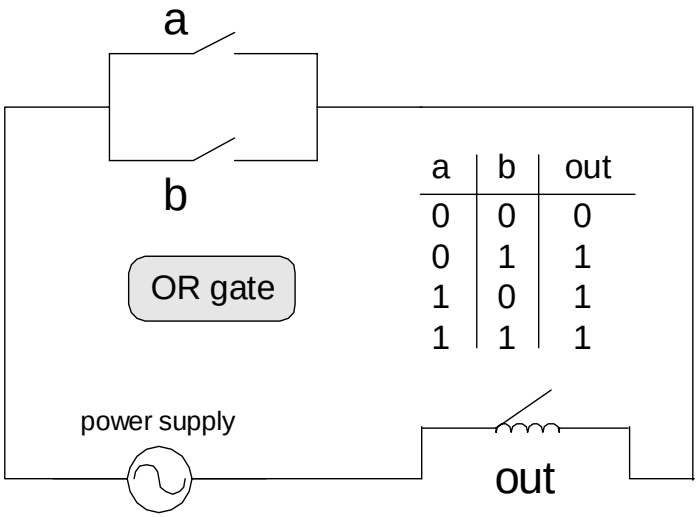


Xor(a,b) = Or(And(a,Not(b)),And(Not(a),b)))

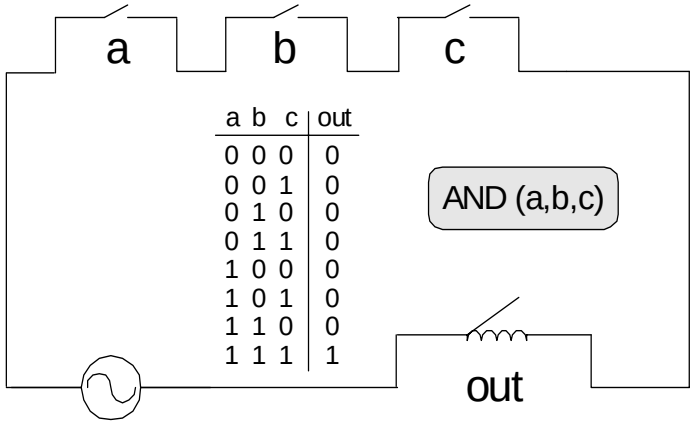
Circuit implementations



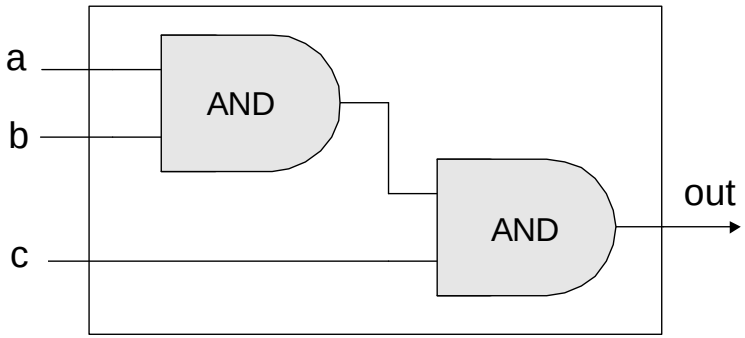
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



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



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



- From a computer science perspective, physical realizations of logic gates are irrelevant.

Project 1: elementary logic gates

Given: Nand(a,b), false

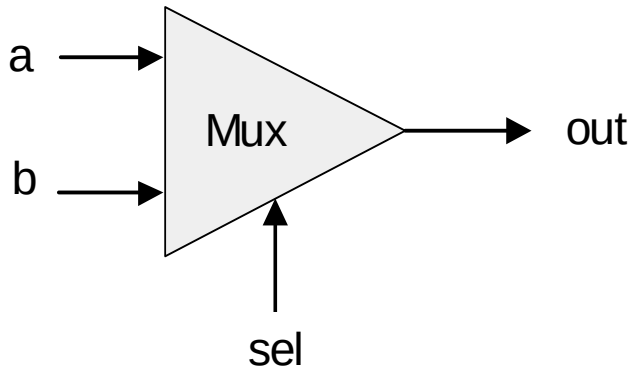
a	b	Nand(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

Build:

- Not(a) = ...
- true = ...
- And(a,b) = ...
- Or(a,b) = ...
- Mux(a,b,sel) = ...
- Etc. - 12 gates altogether.

Multiplexer

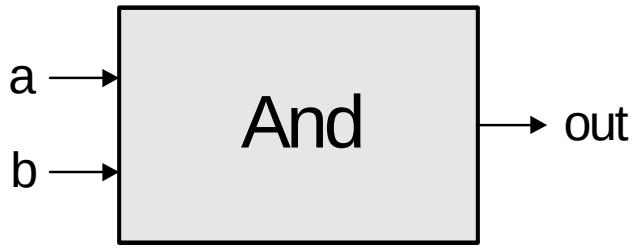
a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



sel	out
0	a
1	b

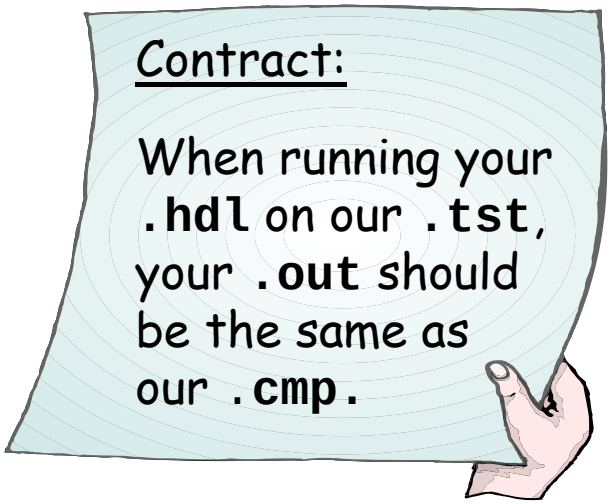
- Implementation: based on Not, And, Or gates.

Example: Building an **And** gate



And.cmp

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



And.hdl

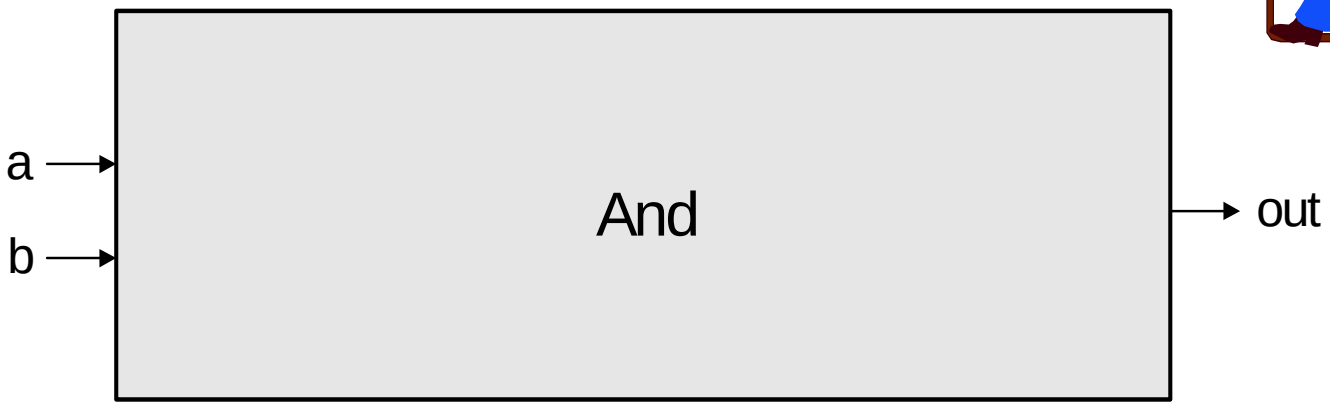
```
CHIP And
{   IN  a, b;
    OUT out;
    // implementation missing
}
```

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;
```

Building an **And** gate

Interface: And(a,b) = 1 exactly when a=b=1

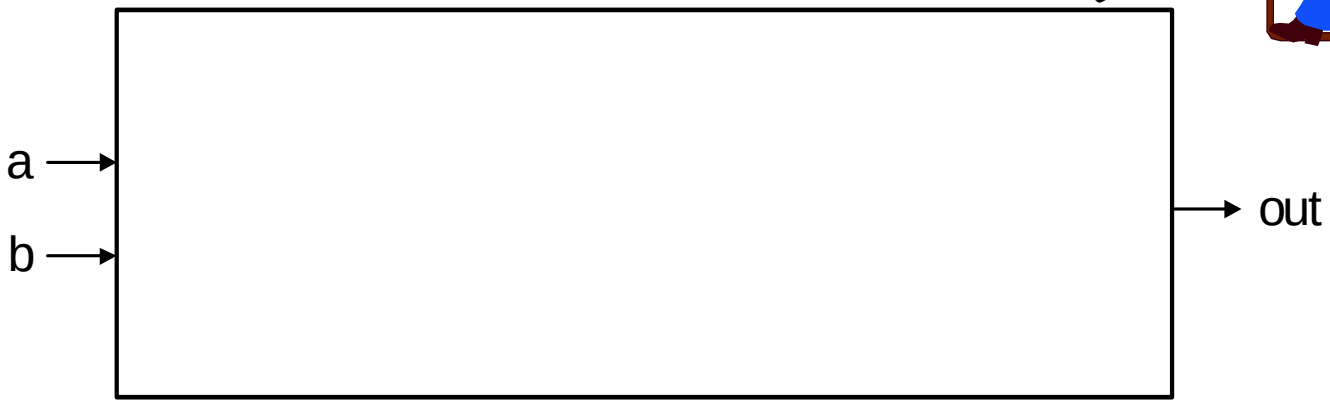


And.hdl

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

Building an **And** gate

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

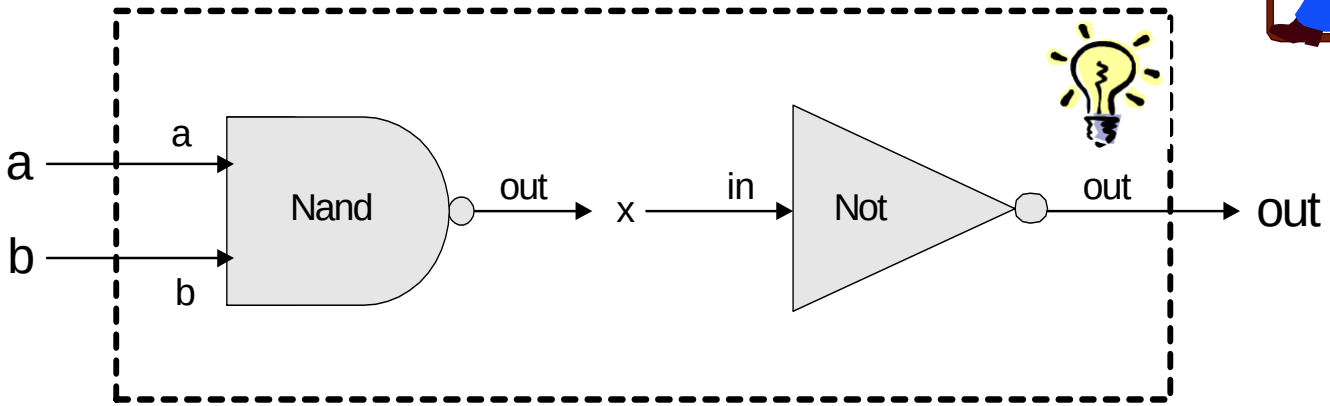


And.hdl

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

Building an **And** gate

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

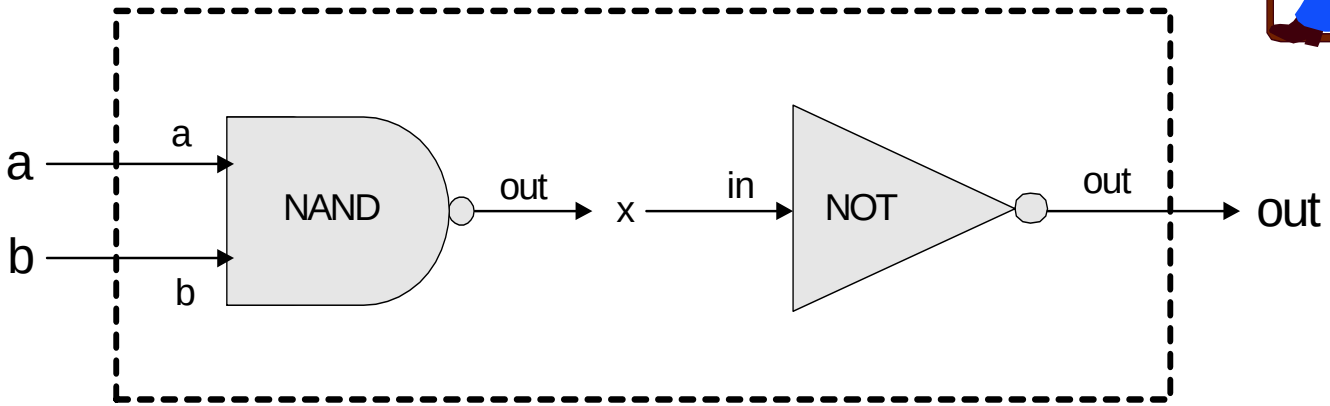


And.hdl

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

Building an **And** gate

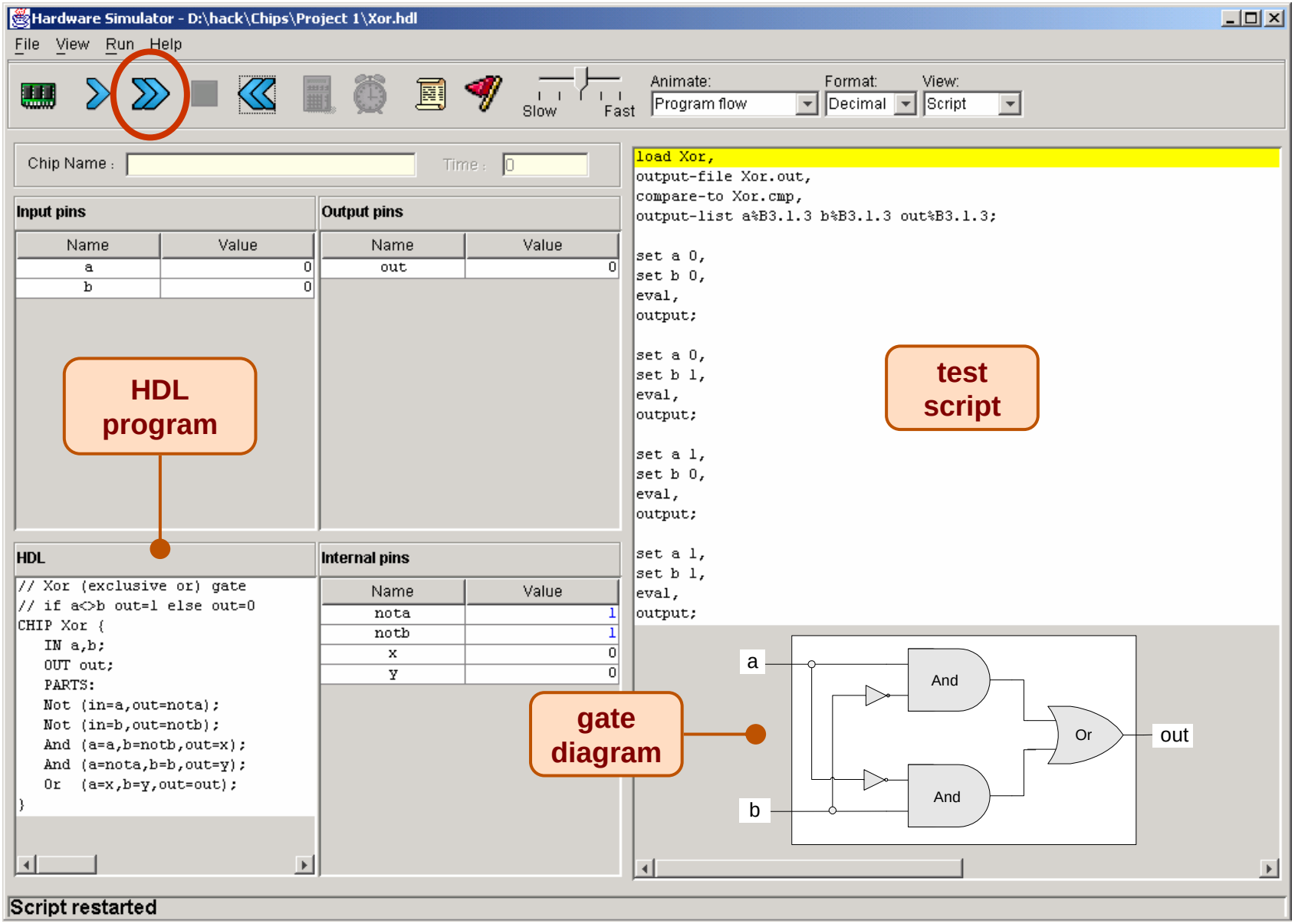
Implementation: And(a,b) = Not(Nand(a,b))



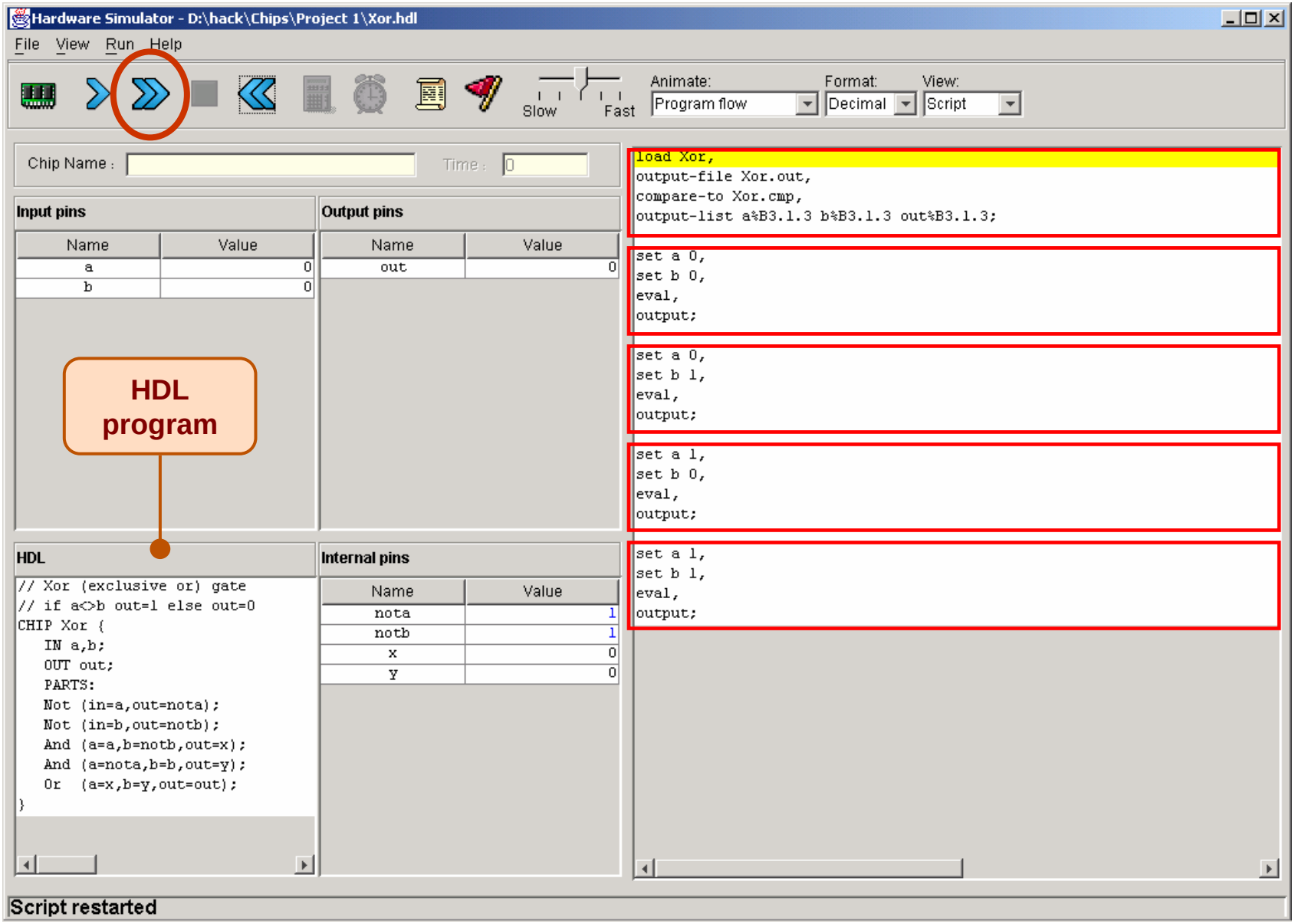
And.hdl

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

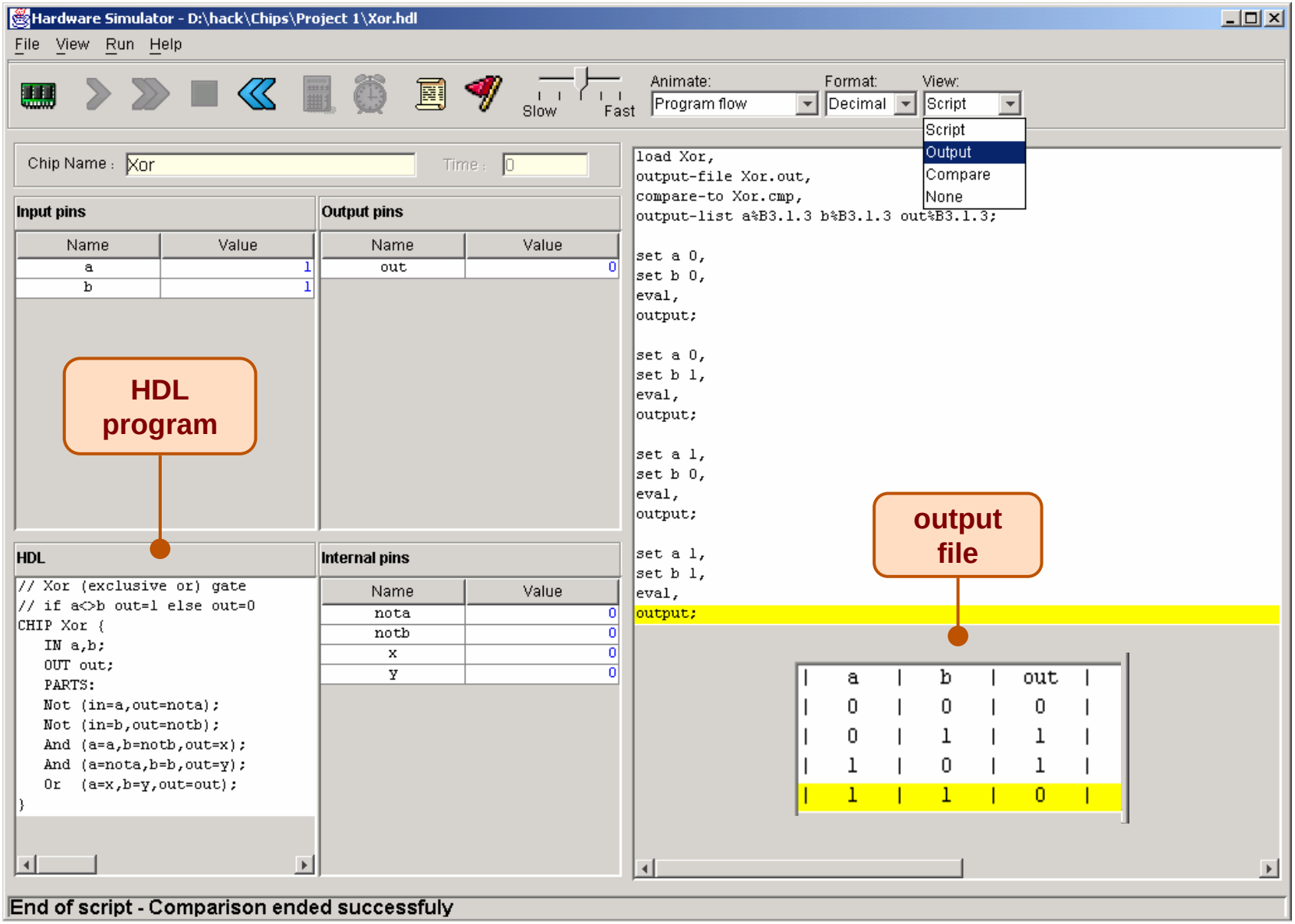
Hardware simulator



Hardware simulator



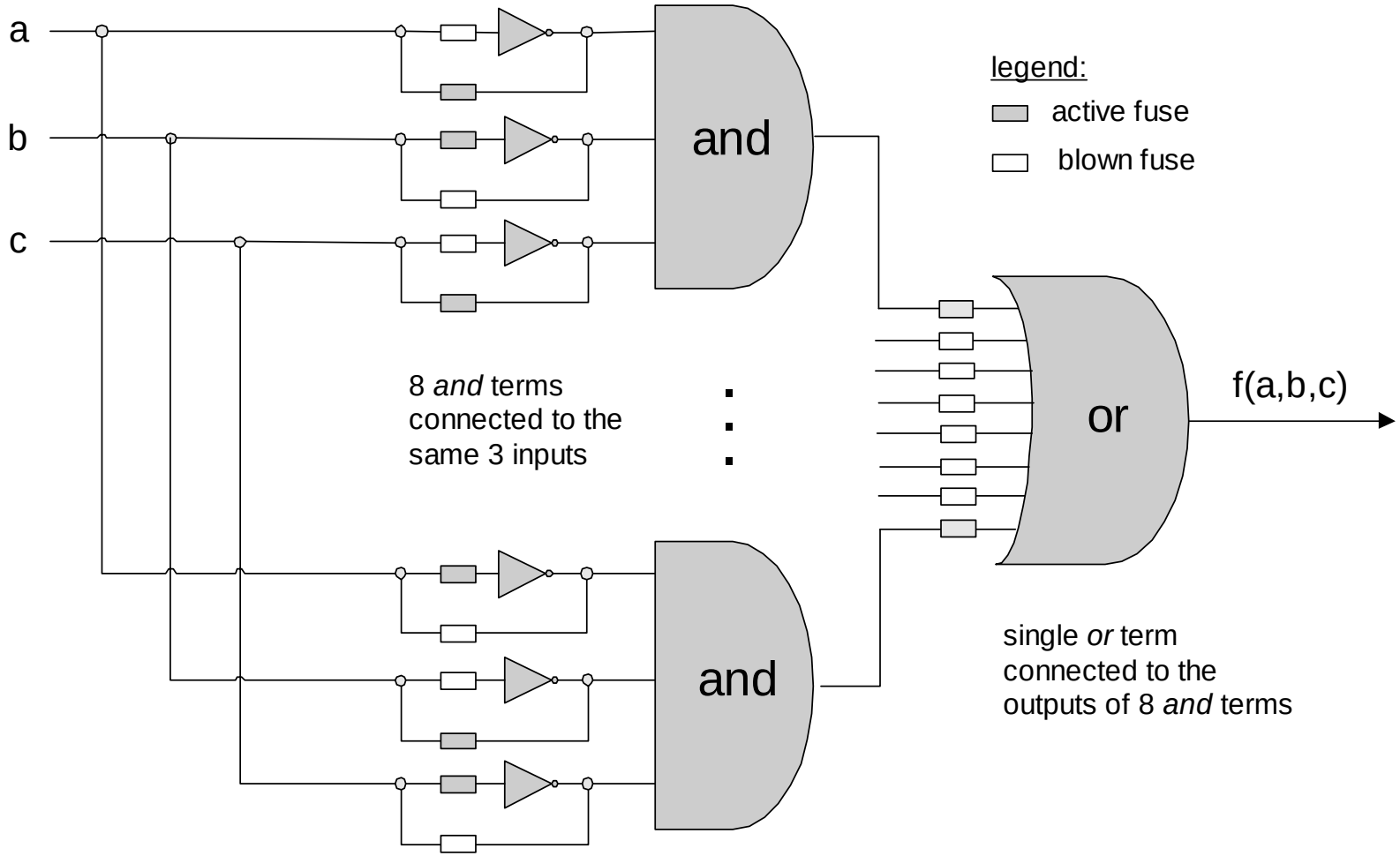
Hardware simulator



Project 1 tips

- Read Chapter 1 of the book
- Explore the book's web site (www.idc.ac.il/tecs)
- Download the book's software suite
- Go through the hardware simulator
- You're in business.

End note: Programmable Logic Device for 3-way functions



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

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

Perspective

- Each Boolean function has a canonical representation
- The canonical representation is expressed in terms of And, Not, Or
- And, Not, Or can be expressed in terms of Nand alone
- Ergo, every Boolean function can be realized by a standard PLD consisting of Nand gates only
- Mass production
- Universal building blocks, unique topology
- Gates, neurons, atoms, …