

Chapter 1: Boolean Logic

Usage and Copyright Notice:

Copyright 2005 © Noam Nisan and Shimon Schocken

This presentation contains lecture materials that accompany the textbook “The Elements of Computing Systems” by Noam Nisan & Shimon Schocken, MIT Press, 2005.

The book web site, www.idc.ac.il/tecs , features 13 such presentations, one for each book chapter. Each presentation is designed to support about 3 hours of classroom or self-study instruction.

You are welcome to use or edit this presentation for instructional and non-commercial purposes.

If you use our materials, we will appreciate it if you will include in them a reference to the book’s web site.

And, if you have any comments, you can reach us at tecs.ta@gmail.com

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,y)
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.

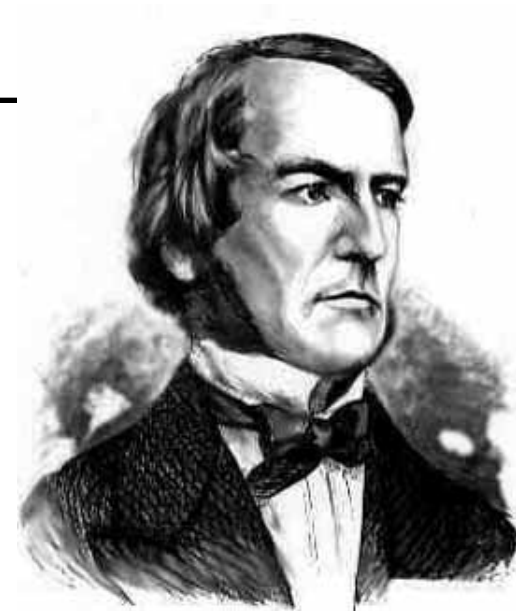
All Boolean functions of 2 variables

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

Boolean algebra

Given: $\text{Nand}(a,b)$, false

- $\text{Not}(a) = \text{Nand}(a,a)$
- $\text{true} = \text{Not}(\text{false})$
- $\text{And}(a,b) = \text{Not}(\text{Nand}(a,b))$
- $\text{Or}(a,b) = \text{Not}(\text{And}(\text{Not}(a), \text{Not}(b)))$
- $\text{Xor}(a,b) = \text{Or}(\text{And}(a, \text{Not}(b)), \text{And}(\text{Not}(a), b))$
- Etc.

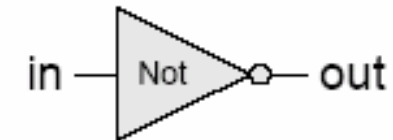
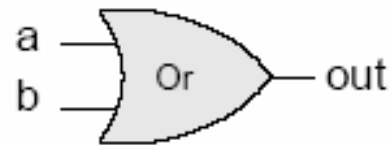
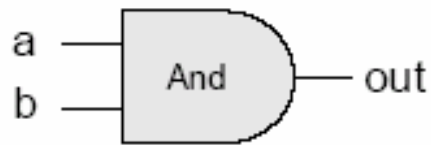


George Boole, 1815-1864
("A Calculus of Logic")

Gate logic

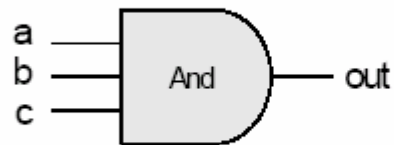
- Gate logic - a gate architecture designed to implement a Boolean function

- Elementary gates:



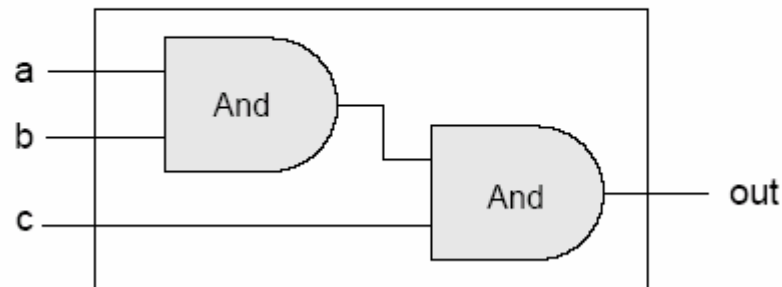
- Composite gates:

Gate interface



If $a=b=c=1$ then $out=1$
else $out=0$

Gate implementation



- Interface VS implementation.

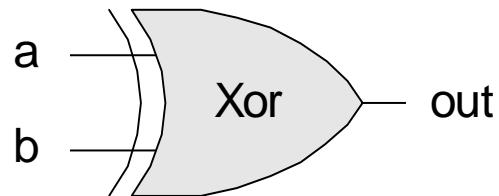
Gate Logic



Claude Shannon, 1916-2001

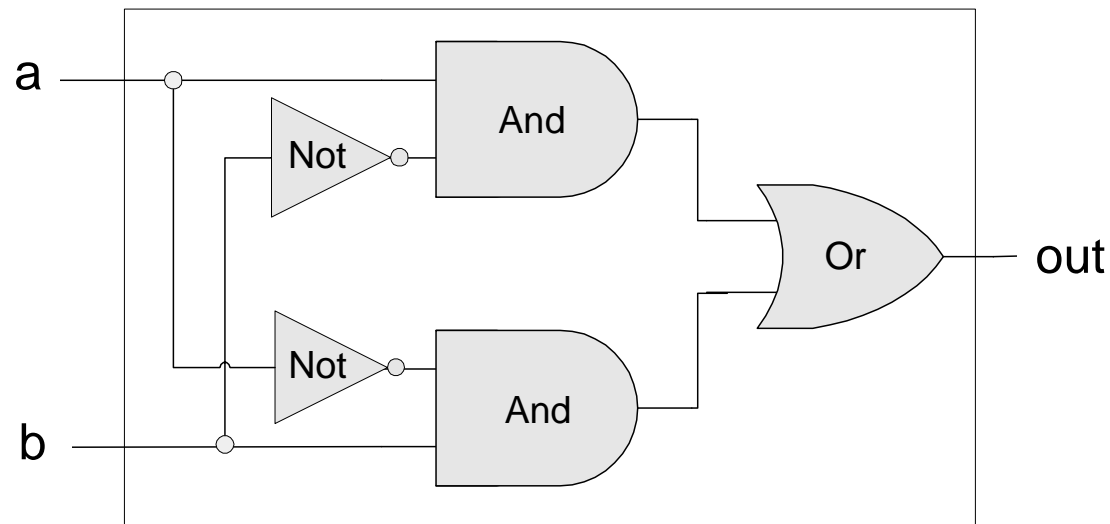
("Symbolic Analysis of Relay and Switching Circuits")

Interface



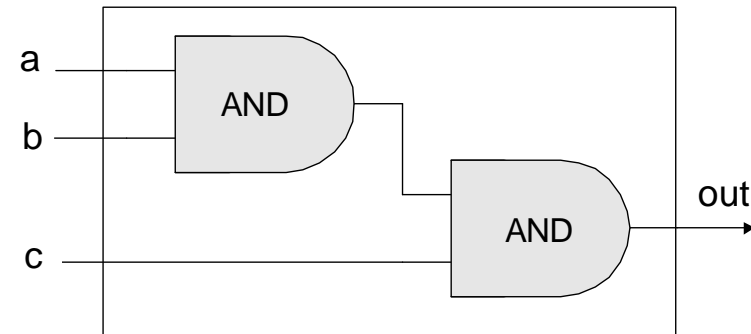
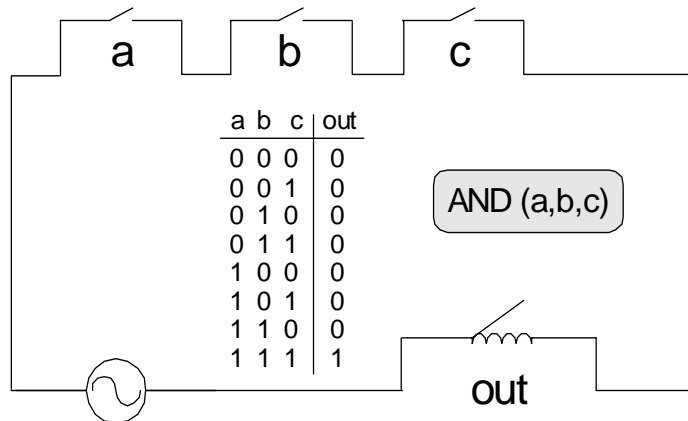
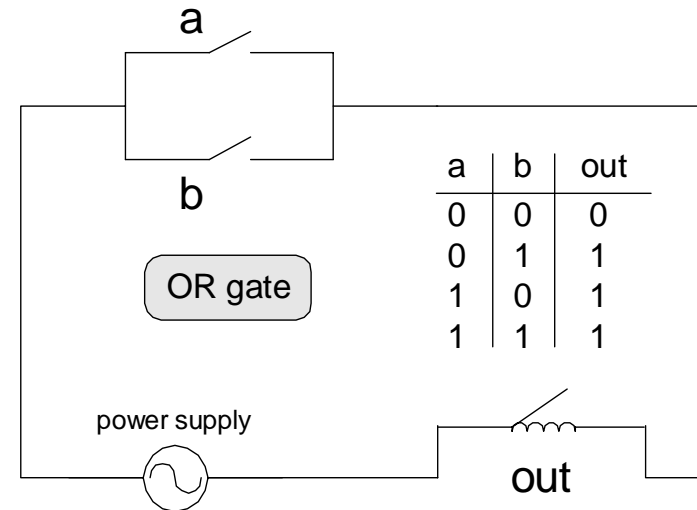
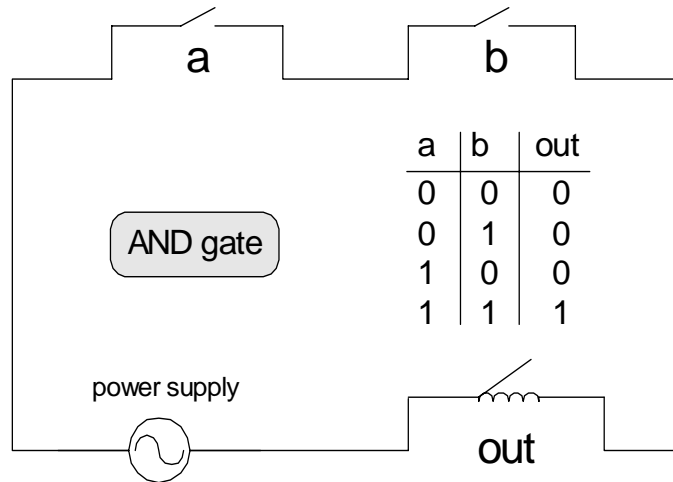
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



$$\text{Xor}(a,b) = \text{Or}(\text{And}(a,\text{Not}(b)),\text{And}(\text{Not}(a),b))$$

Circuit implementations



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

Project 1: elementary logic gates

Given: $\text{Nand}(a,b)$, false

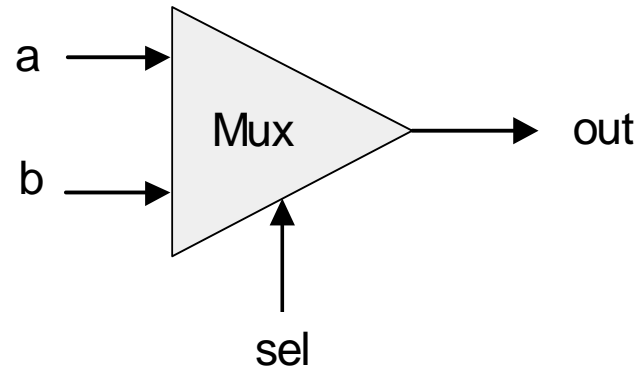
Build:

a	b	$\text{Nand}(a,b)$
0	0	1
0	1	1
1	0	1
1	1	0

- $\text{Not}(a) = \dots$
- $\text{true} = \dots$
- $\text{And}(a,b) = \dots$
- $\text{Or}(a,b) = \dots$
- $\text{Mux}(a,b,\text{sel}) = \dots$
- 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

Contract:

When running your `.hdl` on our `.tst`, your `.out` should be the same as our `.cmp`.

`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: $\text{And}(a,b) = 1$ exactly when $a=b=1$



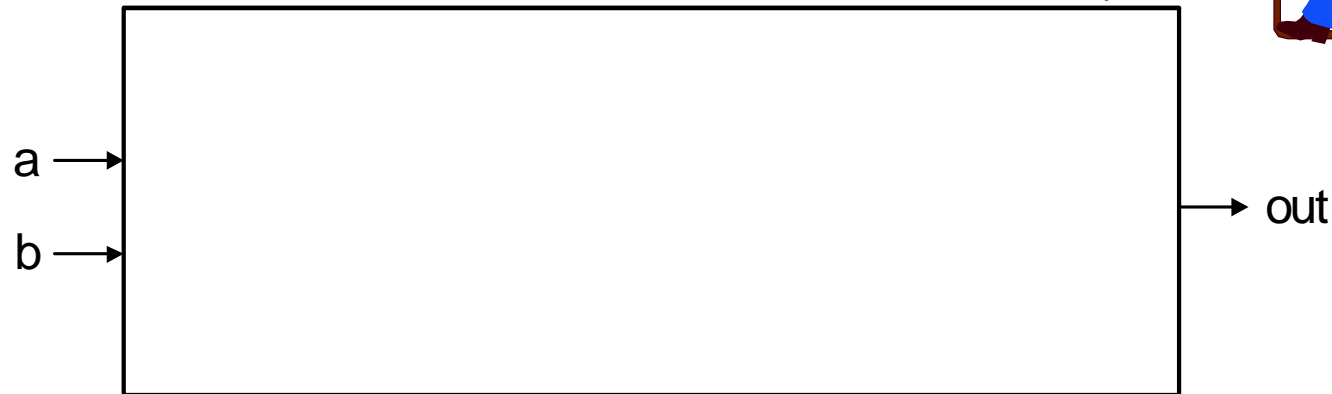
And.hdl

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

Building an And gate



Implementation: $\text{And}(a,b) = \text{Not}(\text{Nand}(a,b))$



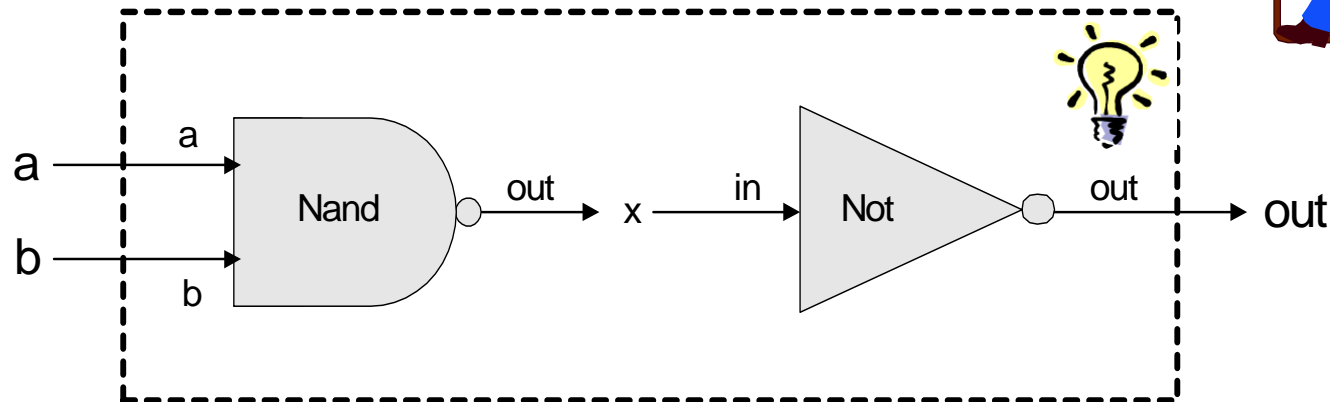
And.hdl

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

Building an And gate



Implementation: $\text{And}(a,b) = \text{Not}(\text{Nand}(a,b))$



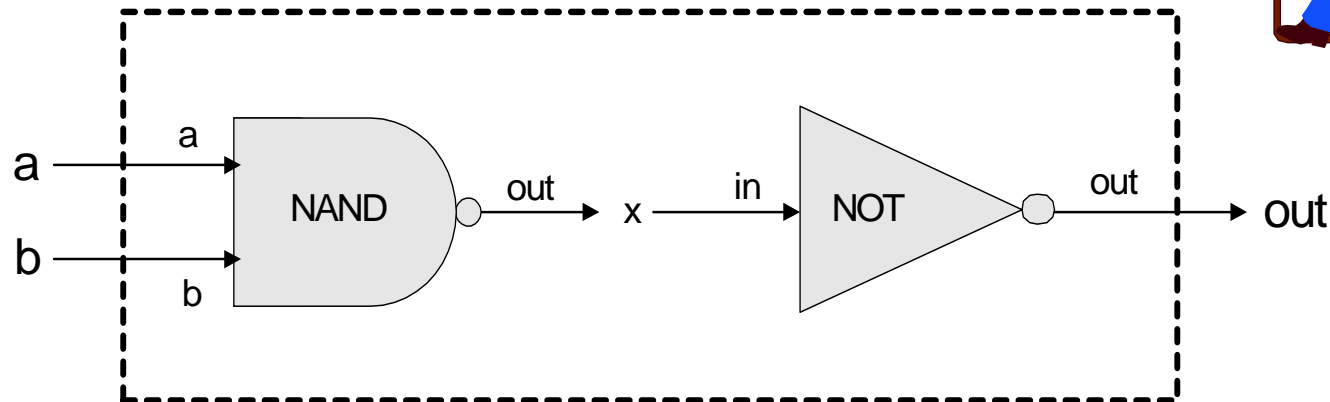
And.hdl

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

Building an And gate



Implementation: $\text{And}(a,b) = \text{Not}(\text{Nand}(a,b))$



And.hdl

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



Hardware simulator

The screenshot shows a hardware simulator window titled "Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl". The interface includes a menu bar (File, View, Run, Help), a toolbar with simulation controls (a red circle highlights the "Run" button), and several panels:

- Input pins:** A table with columns "Name" and "Value".

Name	Value
a	0
b	0
- Output pins:** A table with columns "Name" and "Value".

Name	Value
out	0
- HDL:** A text area containing Verilog code for an XOR gate.

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
  IN a,b;
  OUT out;
  PARTS:
    Not (in=a,out=nota);
    Not (in=b,out=notb);
    And (a=a,b=notb,out=x);
    And (a=nota,b=b,out=y);
    Or (a=x,b=y,out=out);
}
```
- Internal pins:** A table with columns "Name" and "Value".

Name	Value
nota	1
notb	1
x	0
y	0
- test script:** A text area containing a script for testing the chip.

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```
- gate diagram:** A logic diagram showing the implementation of the XOR gate using two AND gates, two NOT gates, and one OR gate. Inputs 'a' and 'b' are connected to the AND gates. The outputs of the AND gates are connected to the OR gate, which produces the output 'out'.

Annotations in orange boxes point to the HDL program, test script, and gate diagram.

Script restarted

Hardware simulator

The screenshot shows a hardware simulator window titled "Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl". The interface includes a menu bar (File, View, Run, Help), a toolbar with navigation icons (a red circle highlights the right arrow), and control options for animation (Slow/Fast) and output format (Program flow, Decimal, Script). The main area is divided into several sections:

- Chip Name:** A text field containing "Xor.hdl" and a "Time:" field showing "0".
- Input pins:** A table with columns "Name" and "Value".

Name	Value
a	0
b	0
- Output pins:** A table with columns "Name" and "Value".

Name	Value
out	0
- Internal pins:** A table with columns "Name" and "Value".

Name	Value
nota	1
notb	1
x	0
y	0
- HDL:** A text area containing the following code:

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
  IN a,b;
  OUT out;
  PARTS:
  Not (in=a,out=nota);
  Not (in=b,out=notb);
  And (a=a,b=notb,out=x);
  And (a=nota,b=b,out=y);
  Or (a=x,b=y,out=out);
}
```
- Script execution log:** A text area on the right showing the execution of a script. The first line is highlighted in yellow:

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

A red-bordered box highlights the script execution log. A callout box with the text "HDL program" points to the HDL code area. The status bar at the bottom indicates "Script restarted".

Hardware simulator

The screenshot shows a hardware simulator window titled "Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl". The interface includes a menu bar (File, View, Run, Help), a toolbar with simulation controls (Play, Stop, Step, Slow, Fast), and a "View" dropdown menu currently set to "Script".

The "Chip Name" is "Xor" and the "Time" is 0. The "Input pins" table shows:

Name	Value
a	1
b	1

The "Output pins" table shows:

Name	Value
out	0

The "HDL" section contains the following code:

```
// Xor (exclusive or) gate
// if a<>b out=1 else out=0
CHIP Xor {
  IN a,b;
  OUT out;
  PARTS:
    Not (in=a,out=nota);
    Not (in=b,out=notb);
    And (a=a,b=notb,out=x);
    And (a=nota,b=b,out=y);
    Or (a=x,b=y,out=out);
}
```

The "Internal pins" table shows:

Name	Value
nota	0
notb	0
x	0
y	0

The "Script" section shows the following commands:

```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

The "output:" line is highlighted in yellow. Below it, a truth table is displayed:

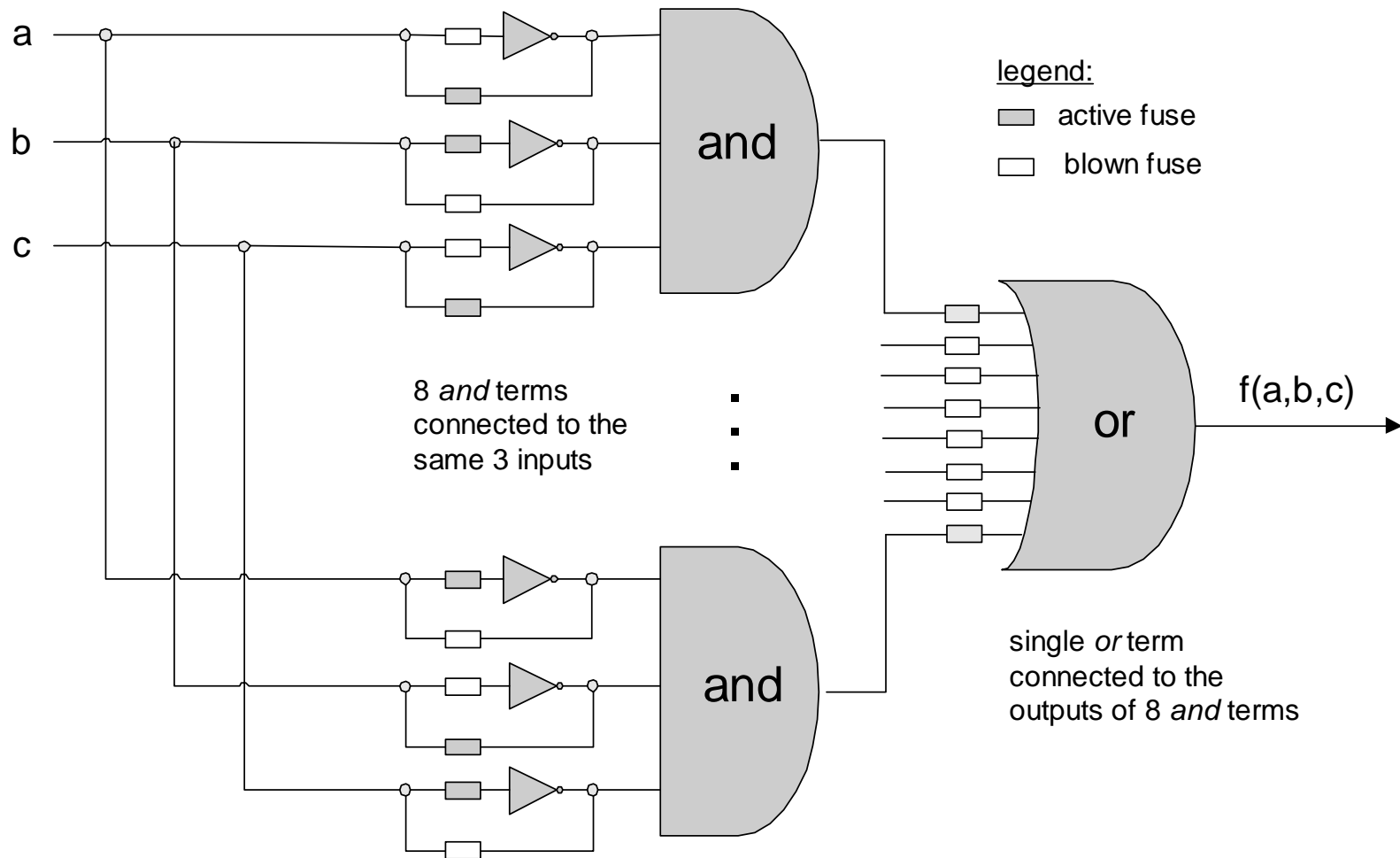
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

The last row of the truth table is highlighted in yellow. A status bar at the bottom reads "End of script - Comparison ended successfully".

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, ...

