

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.

We provide both PPT and PDF versions.

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 as you see fit 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.

If you have any questions or 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

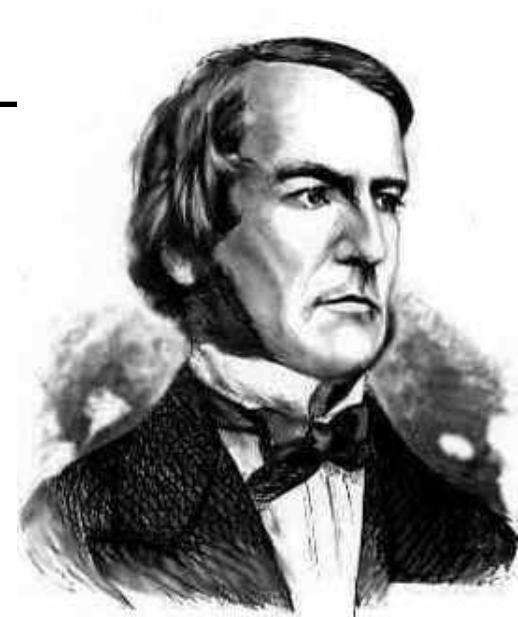
Function	x	0	0	1	1
	y	0	1	0	1
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \bar{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\bar{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \bar{y} + \bar{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x + y}$	1	0	0	0
Equivalence	$x \cdot y + \bar{x}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

We can build:

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

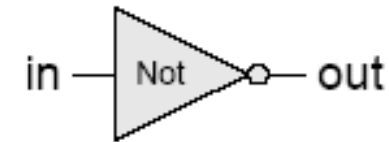
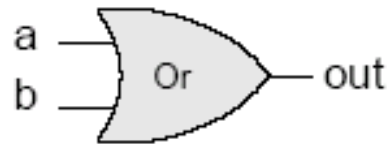
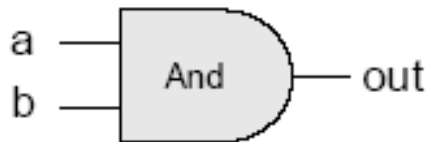


George Boole, 1815-1864
("A Calculus of Logic")

Gate logic

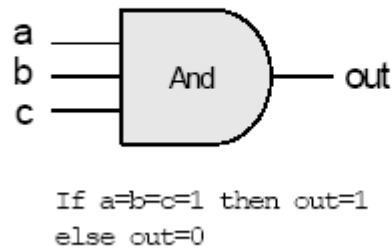
- Gate logic - a gate architecture designed to implement a Boolean function

- Elementary gates:

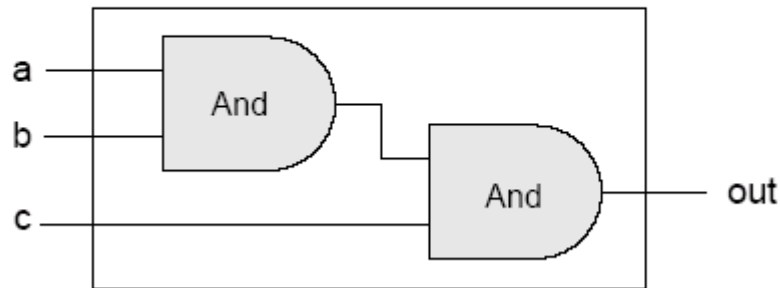


- Composite gates:

Gate interface



Gate implementation



- Important distinction: Interface (what) VS implementation (how).

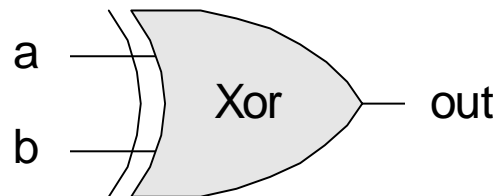
Gate logic



Claude Shannon, 1916-2001

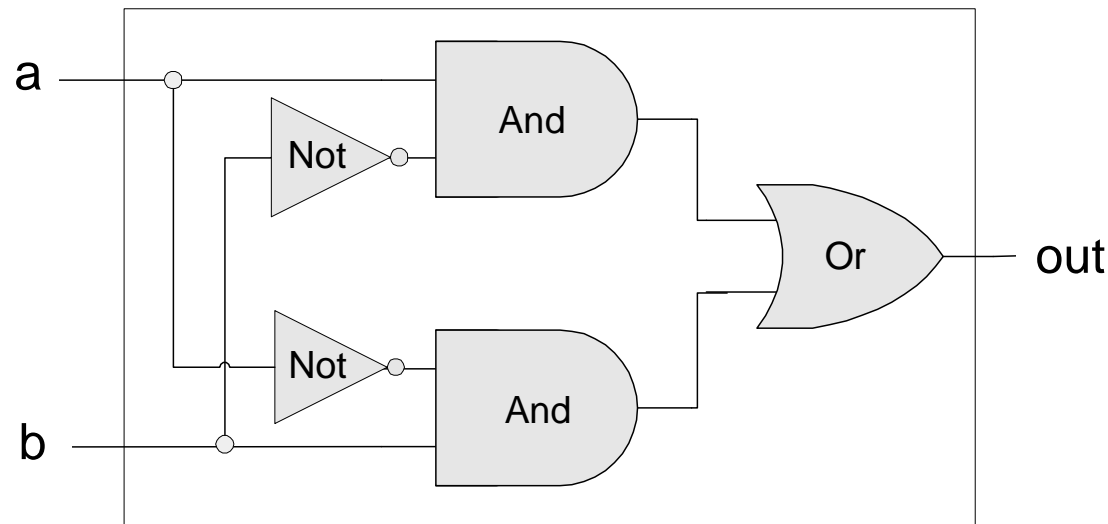
("Symbolic Analysis of Relay and Switching Circuits")

Interface



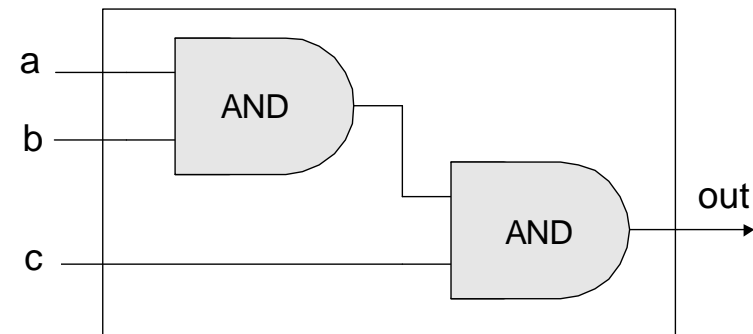
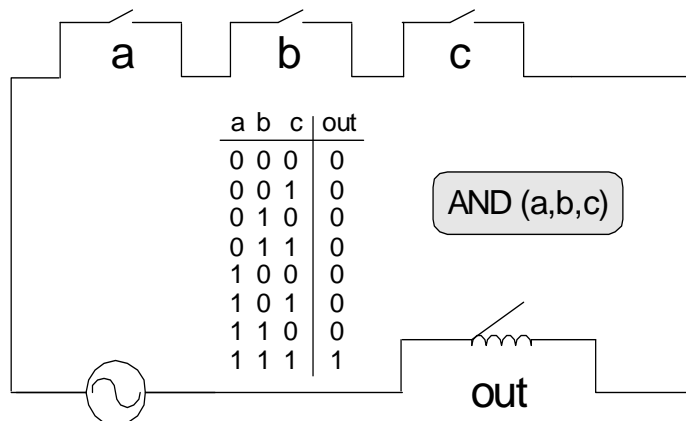
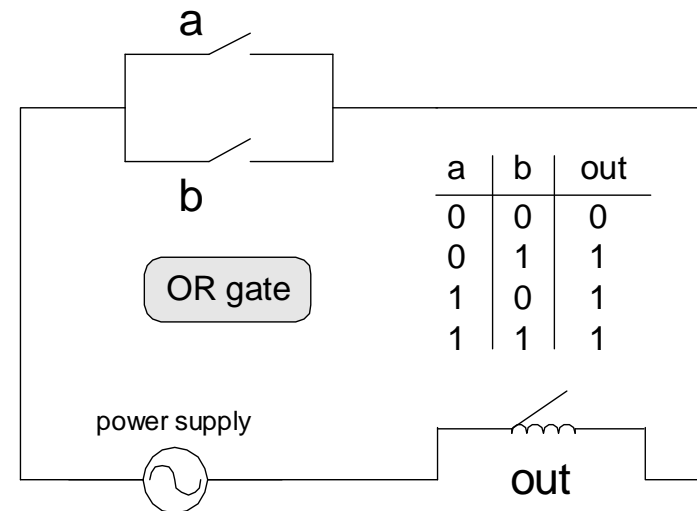
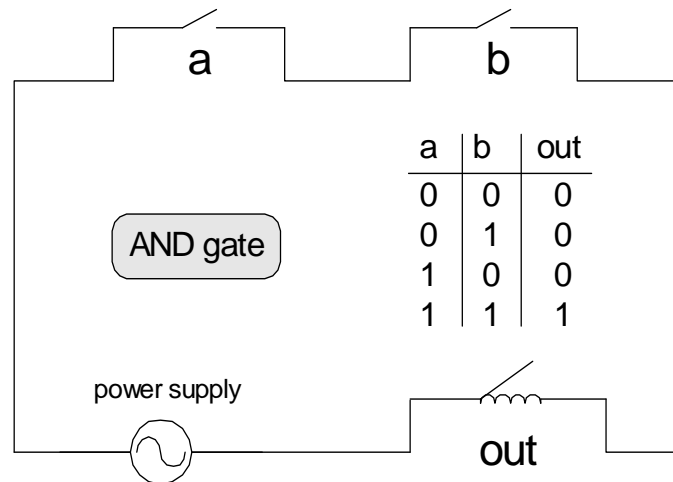
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

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

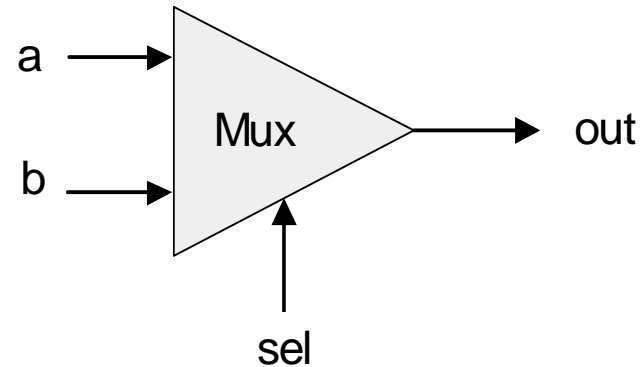
- $\text{Not}(a) = \dots$
- $\text{true} = \dots$
- $\text{And}(a,b) = \dots$
- $\text{Or}(a,b) = \dots$
- $\text{Mux}(a,b,\text{sel}) = \dots$
- Etc. - 12 gates altogether.

Why these particular 12 gates? Since ...

- They are commonly used gates
- They provide all the basic building blocks needed to build our computer.

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

- Proposed 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
}
```

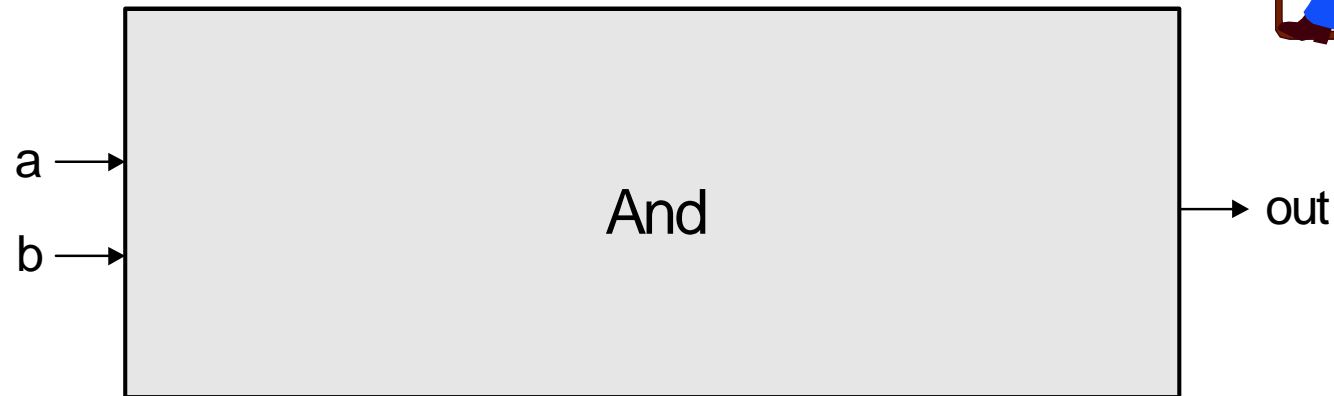
And.tst

```
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 $a=b=1$

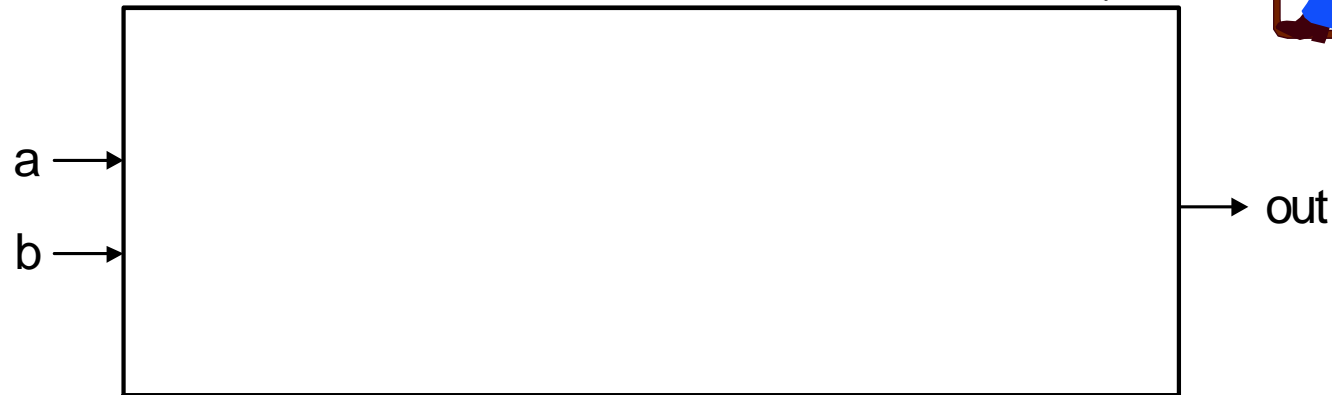


And.hdl

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

Building an **And** gate

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



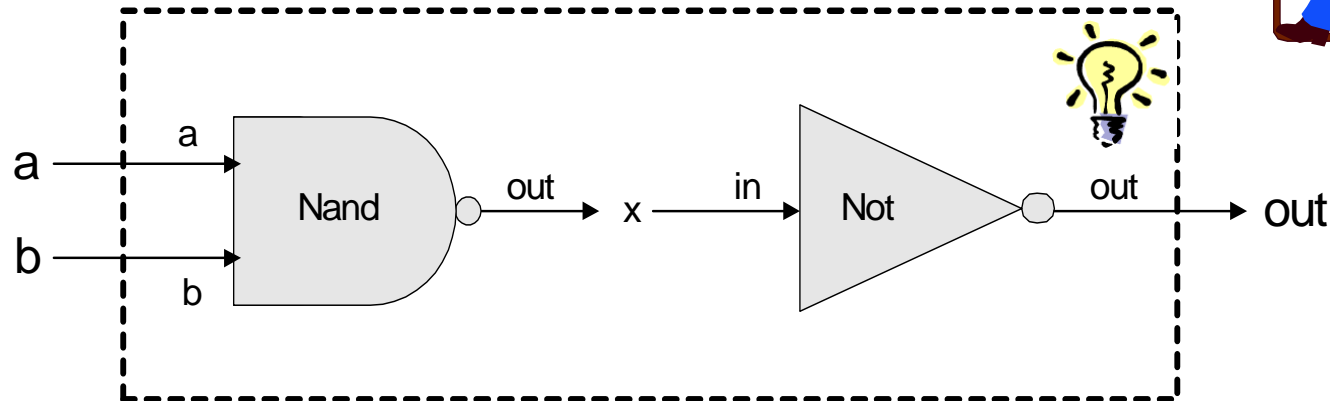
And.hdl

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

Building an And gate



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



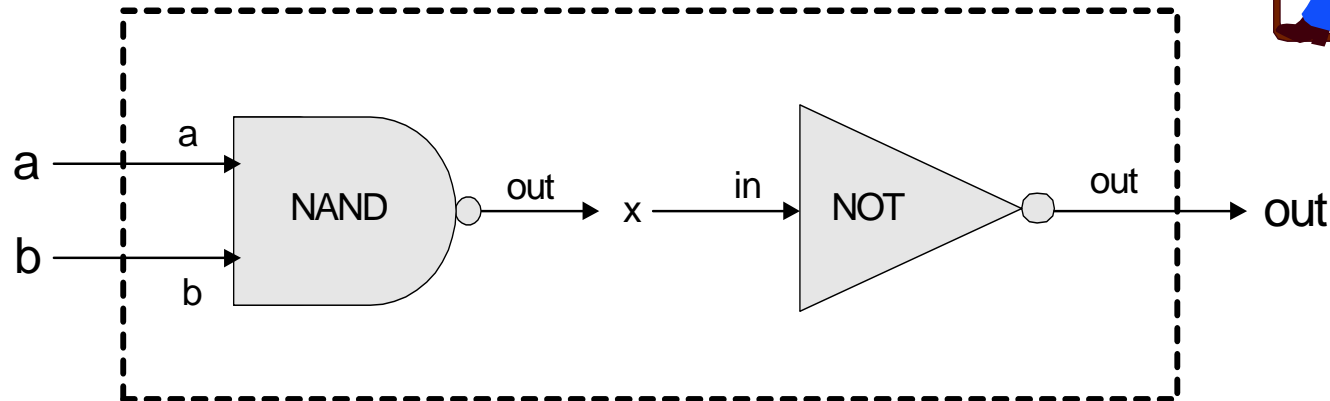
And.hdl

```
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 (demonstrating Xor gate construction)

The screenshot shows the 'Hardware Simulator' window with the following components:

- Toolbar:** Includes icons for file operations, simulation control (a red circle highlights the 'Run' button), and settings. Dropdown menus for 'Animate' (Program flow), 'Format' (Decimal), and 'View' (Script) are present.
- Chip Name:** A text field containing 'D:\hack\Chips\Project 1\Xor.hdl'.
- Time:** A display showing '0'.
- Input pins table:**

Name	Value
a	0
b	0
- Output pins table:**

Name	Value
out	0
- HDL 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=noth);
    And (a=a,b=noth,out=x);
    And (a=nota,b=b,out=y);
    Or (a=x,b=y,out=out);
}
```
- Internal pins table:**

Name	Value
nota	1
noth	1
x	0
y	0
- Test Script:**

```
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;
```
- Status Bar:** Displays 'Script restarted'.

Two orange callout boxes are present: 'HDL program' pointing to the HDL code area, and 'test script' pointing to the test script area.

Hardware simulator

The screenshot shows the Hardware Simulator window for a project named '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 a status bar at the bottom indicating 'Script restarted'.

The main workspace is divided into several panels:

- Input pins:** A table with columns 'Name' and 'Value'. It shows inputs 'a' and 'b', both with a value of 0.
- Output pins:** A table with columns 'Name' and 'Value'. It shows an output 'out' with a value of 0.
- HDL:** A text area containing Verilog code for an XOR gate. A red callout box labeled 'HDL program' points to this area.
- Internal pins:** A table with columns 'Name' and 'Value'. It shows internal signals 'nota' (1), 'notb' (1), 'x' (0), and 'y' (0).
- Script:** A large text area on the right showing the execution script. It includes commands like 'load Xor', 'output-file Xor.out', 'compare-to Xor.cmp', and 'output-list'. It also shows the evaluation of the HDL code for different input combinations of 'a' and 'b'.

The HDL code in the 'HDL' panel is as follows:

```
// 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);
}
```


Hardware simulator

The screenshot shows a 'Hardware Simulator' window with the title bar 'Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl'. The menu bar includes 'File', 'View', 'Run', and 'Help'. The toolbar contains icons for simulation control (play, pause, stop, step back, step forward, slow, fast) and a dropdown menu for 'Animate' (set to 'Program flow'), 'Format' (set to 'Decimal'), and 'View' (set to 'Script').

The 'Chip Name' field is 'Xor' and the 'Time' field is '0'. The 'Input pins' table has two columns: 'Name' and 'Value'. It contains two rows: 'a' with value '1' and 'b' with value '1'. The 'Output pins' table has two columns: 'Name' and 'Value'. It contains one row: 'out' with value '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 has two columns: 'Name' and 'Value'. It contains four rows: 'nota' with value '0', 'notb' with value '0', 'x' with value '0', and 'y' with value '0'.

The 'Script' view shows the following code:

```
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;

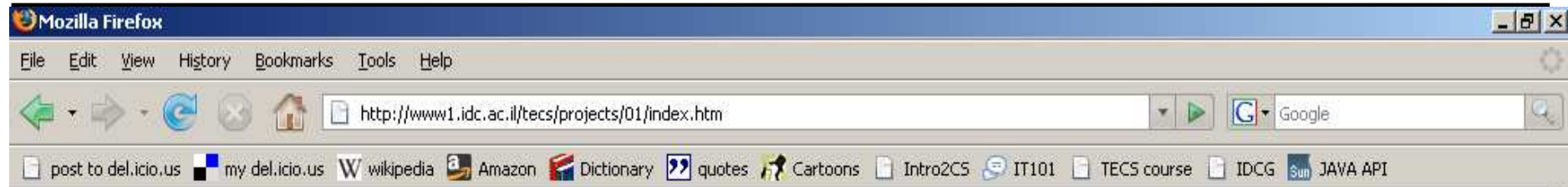
set a 1,
set b 1,
eval,
output;
```

The 'output file' section shows a table with the following data:

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

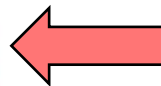
The status bar at the bottom reads 'End of script - Comparison ended successfully'.

Project materials



The Elements of Computing Systems / Nisan & Schocken / www.idc.ac.il/tecs

Project 1: Logic Gates



Project 1 on the course web site

Objective: Implement all the logic gates presented in Chapter 1. The only building blocks that you can use are primitive Nand gates and the composite gates that you will gradually build on top of them.

Resources: The only tool that you need for this project is the hardware simulator supplied with the book. All the chips should be implemented in the HDL language specified in Appendix A. For every one of the chips mentioned in Chapter 1, we provide a skeletal .hdl program (text file) with a missing implementation part. In addition, for each chip we provide a .tst script file that tells the hardware simulator how to test it, along with the correct output file that this script should generate, called .cmp or "compare file". Your job is to complete the missing implementation parts of all the supplied .hdl programs.

Contract: When loaded into the [Hardware Simulator](#), your chip design (modified .hdl program), tested on the supplied .tst file, should produce the outputs listed in the supplied .cmp file. If that is not the case, the simulator will let you know.

Chip (HDL)	Function	Test script	Compare file
Nand	Nand gate (primitive)		
Not	Not gate	Not.tst	Not.cmp
And	And gate	And.tst	And.cmp
Or	Or gate	Or.tst	Or.cmp
Xor	Xor gate	Xor.tst	Xor.cmp
Mux	Mux gate	Mux.tst	Mux.cmp



And.hdl , And.tst , And.cmp files

Project 1 tips

- Read Chapter 1 of the book
- Download the book's software suite
- Go through the hardware simulator tutorial
- Do Project 0 (optional)
- You're in business.

End notes: Canonical representation

Truth table of the function

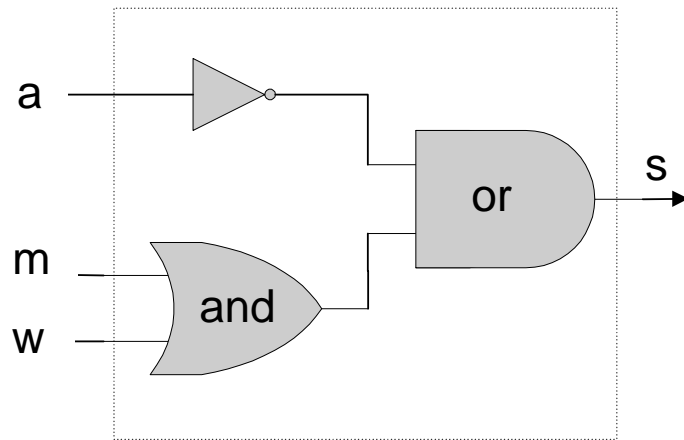
$$s(a, m, w) = \bar{a} \cdot (m + w)$$

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

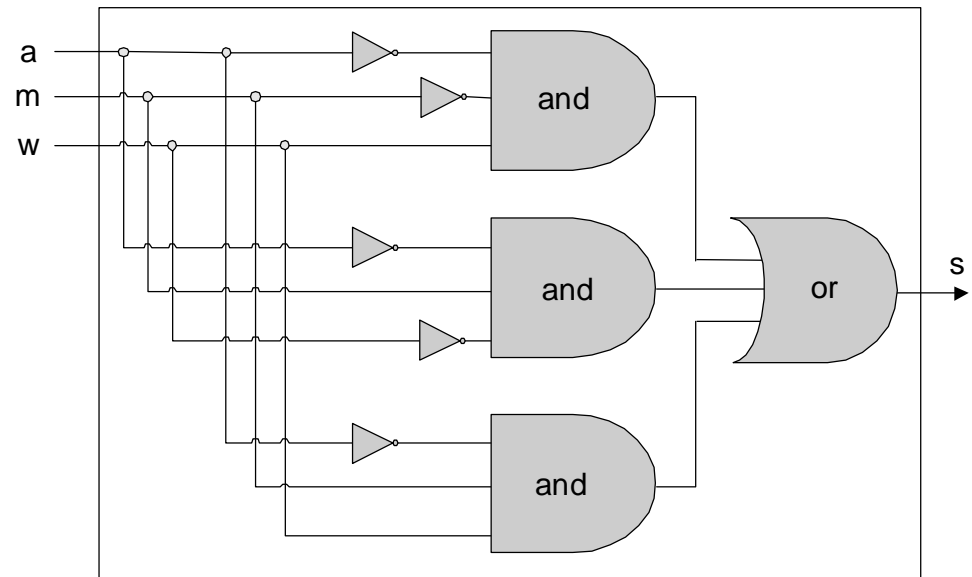
Canonical form: $s(a, m, w) = \bar{a} \bar{m} w + \bar{a} m \bar{w} + \bar{a} m w$

End notes: Canonical representation (cont.)

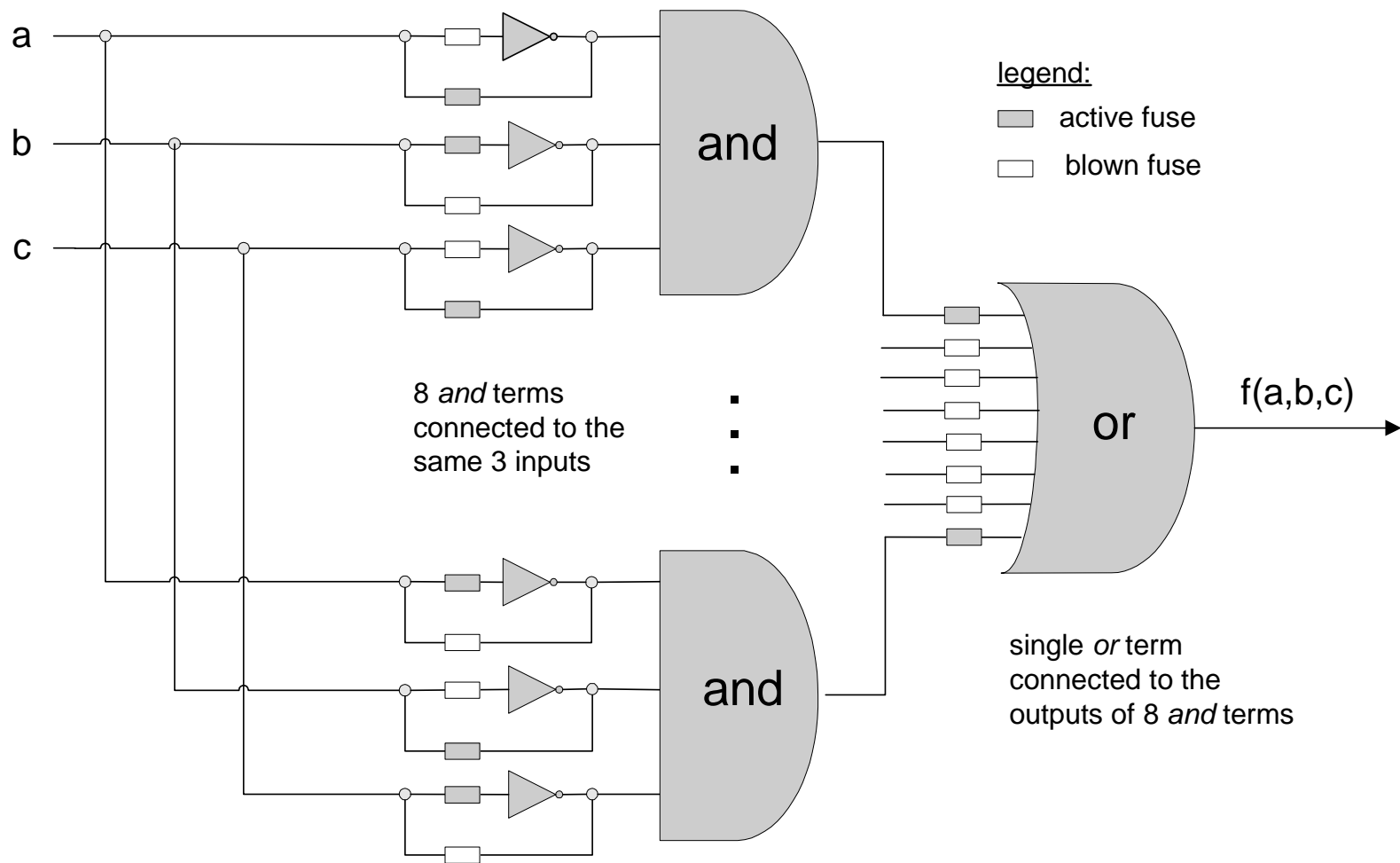
$$s(a, m, w) = \bar{a} \cdot (m + w)$$



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



End notes: Programmable Logic Device for 3-way functions

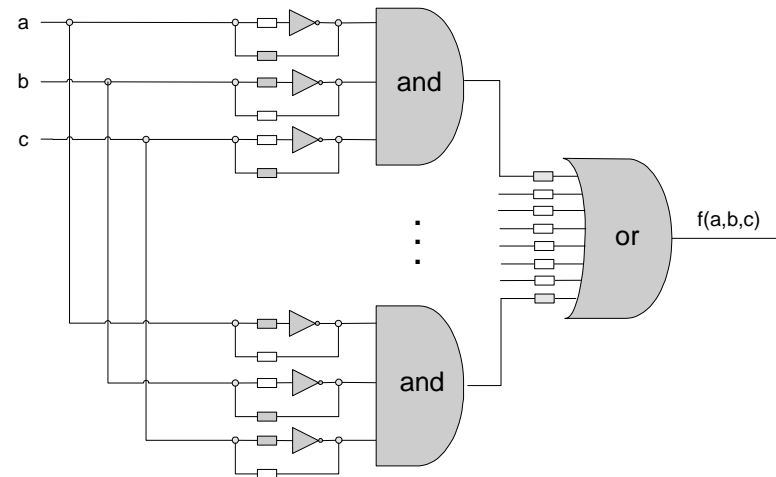


PLD implementation of $f(a,b,c) = a \bar{b} c + \bar{a} b \bar{c}$

(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, ...



End note: universal building blocks, unique topology

