Boolean Logic

Usage and Copyright Notice:

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.

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

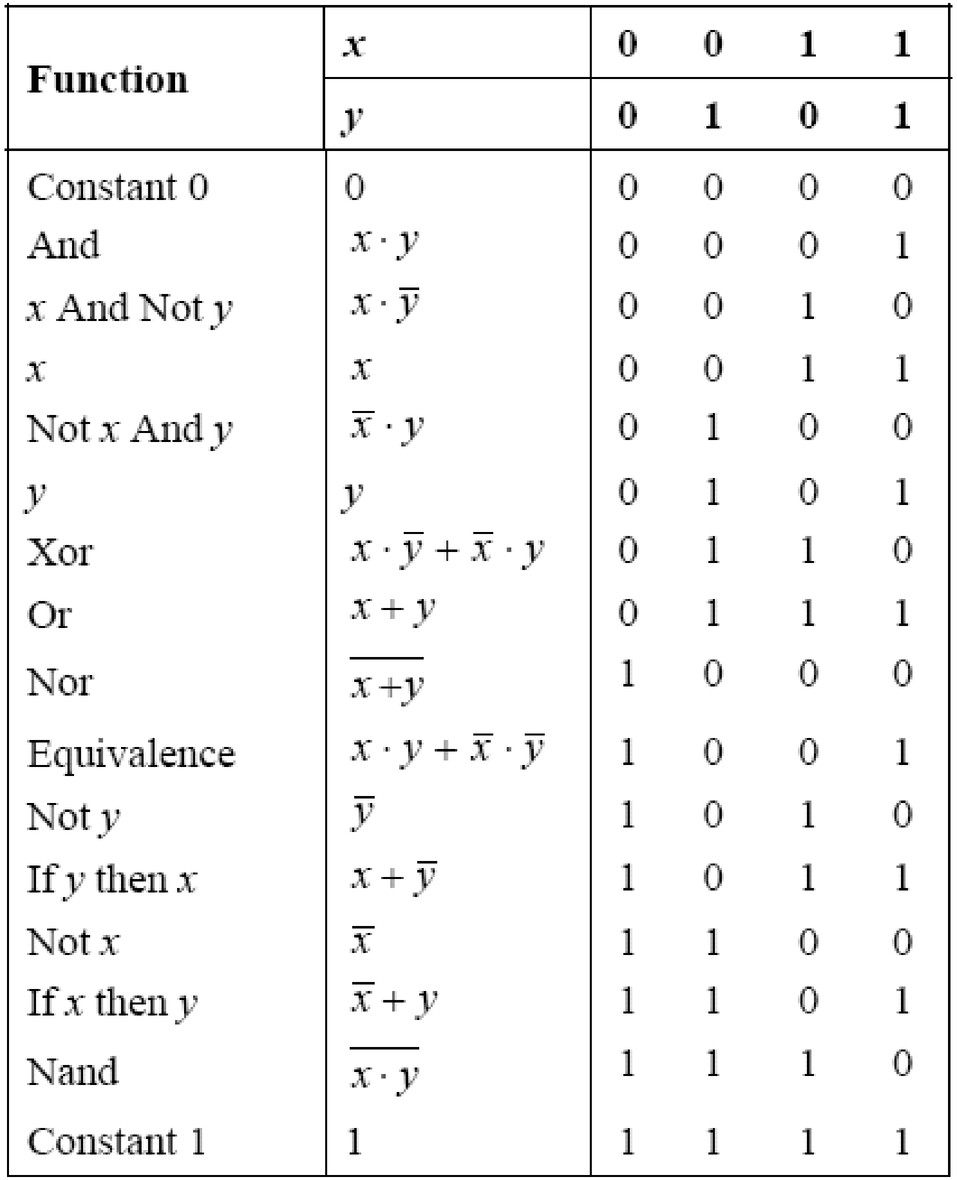
x	y	Nand(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

Boolean functions:

x	y	z	$f(x,y,z)=(x+y)\overline{z}$
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

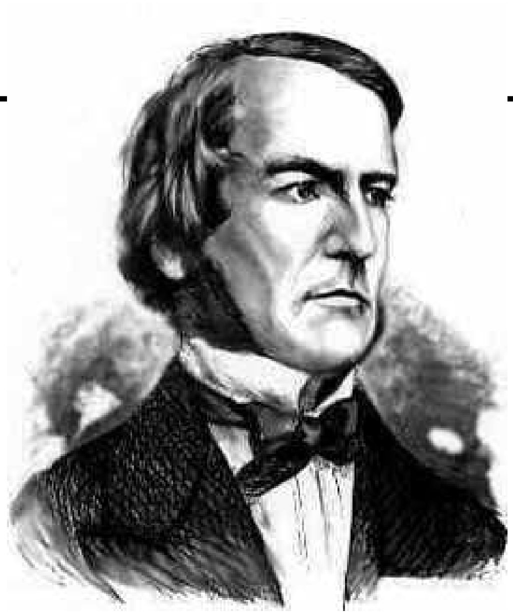
- Functional expression VS truth table expression
- Important result: Every Boolean function can be expressed using And, Or, Not.

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`**

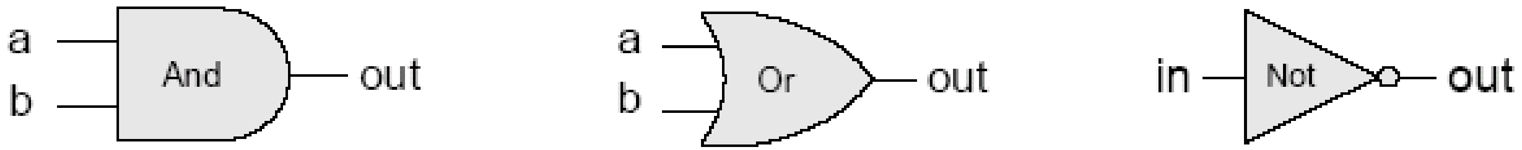
We can build:

- **`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

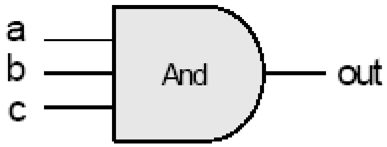
- Gate logic – a gate architecture designed to implement a Boolean function

- Elementary gates:



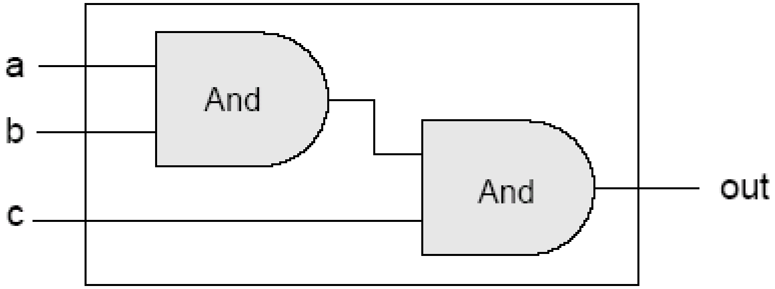
- Composite gates:

Gate interface



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

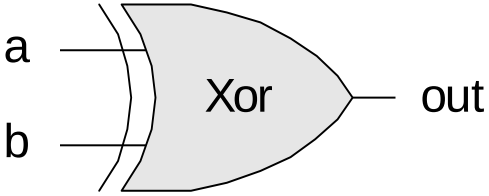
Gate implementation



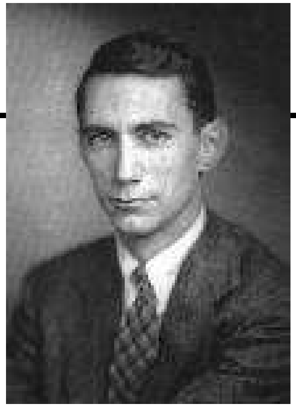
- <u>Important distinction:</u> Interface (what) VS implementation (how).

Gate logic

Interface



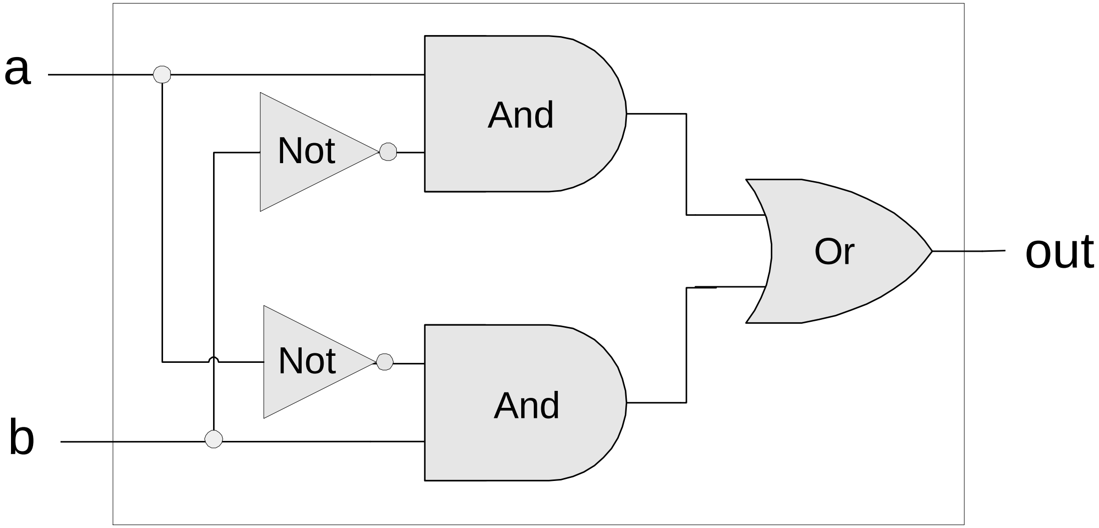
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0



Claude Shannon, 1916-2001

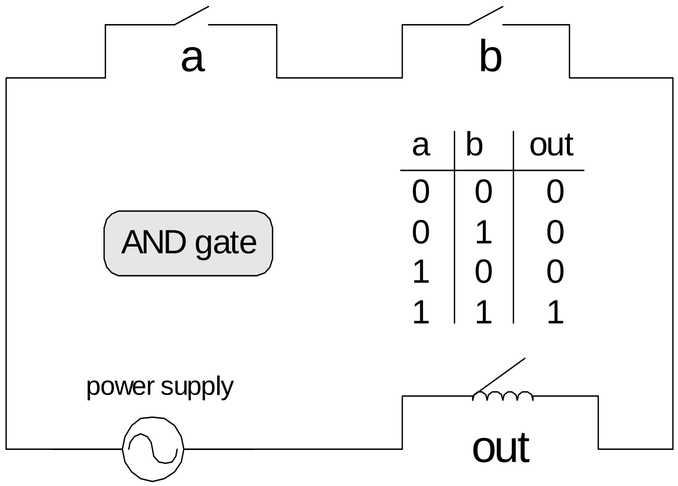
("Symbolic Analysis of Relay and Switching Circuits")

Implementation

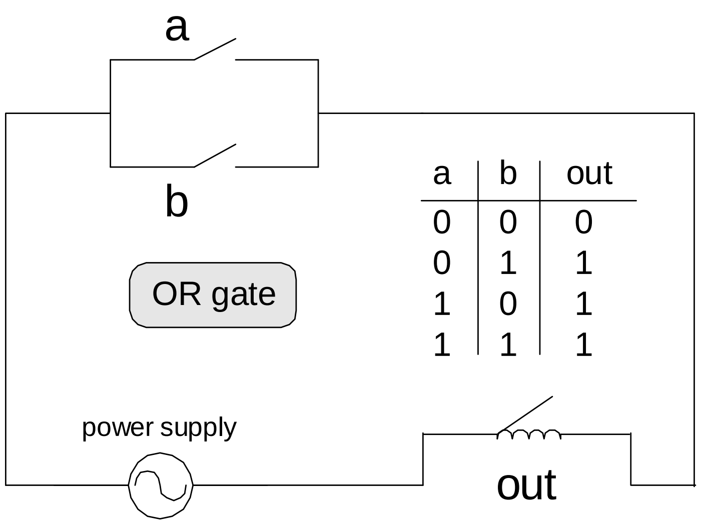


Xor(a,b) = Or(And(a,Not(b)),And(Not(a),b)))

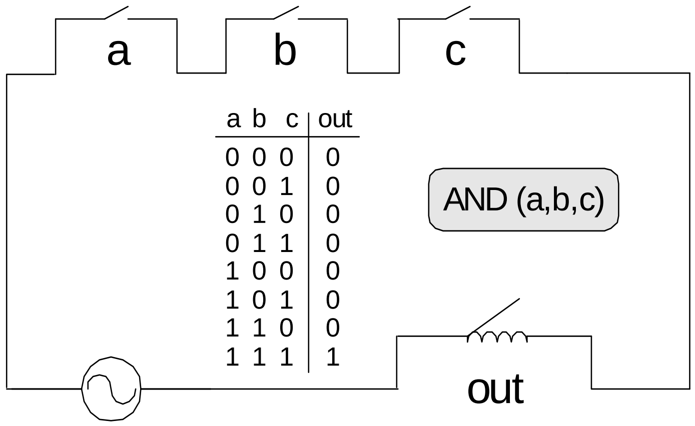
Circuit implementations



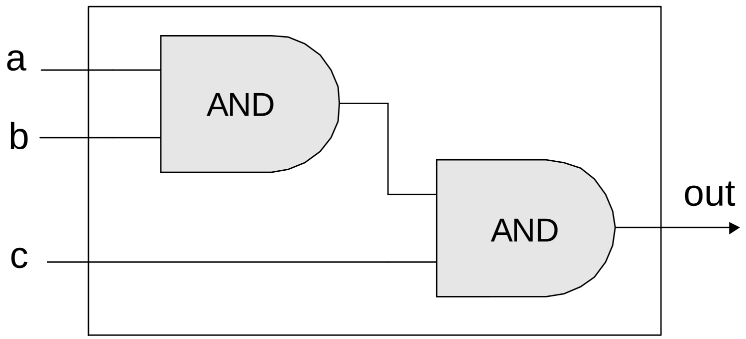
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



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:

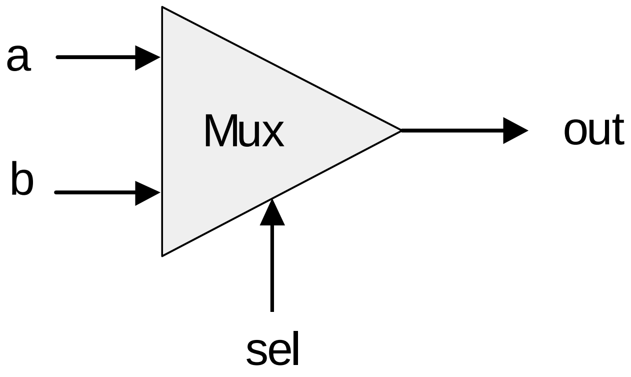
- `Not(a) = ...`
- `true = ...`
- `And(a,b) = ...`
- `Or(a,b) = ...`
- `Mux(a,b,sel) = ...`
- Etc. - 12 gates altogether.

Why these particular 12 gates? Since …

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

Multiplexer

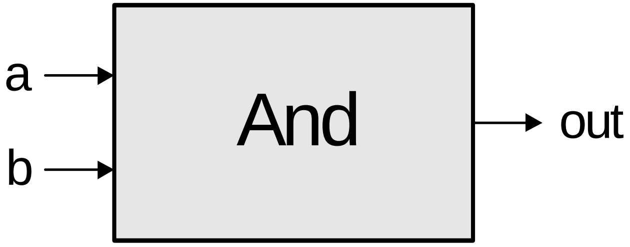
a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



sel	out
0	a
1	b

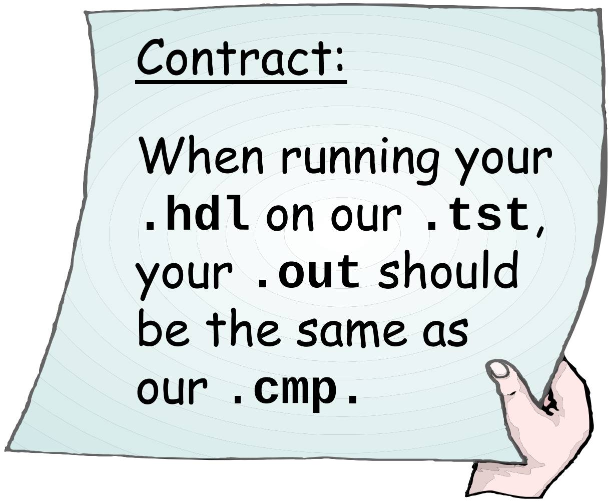
- __Proposed Implementation:__ based on Not, And, Or gates.

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

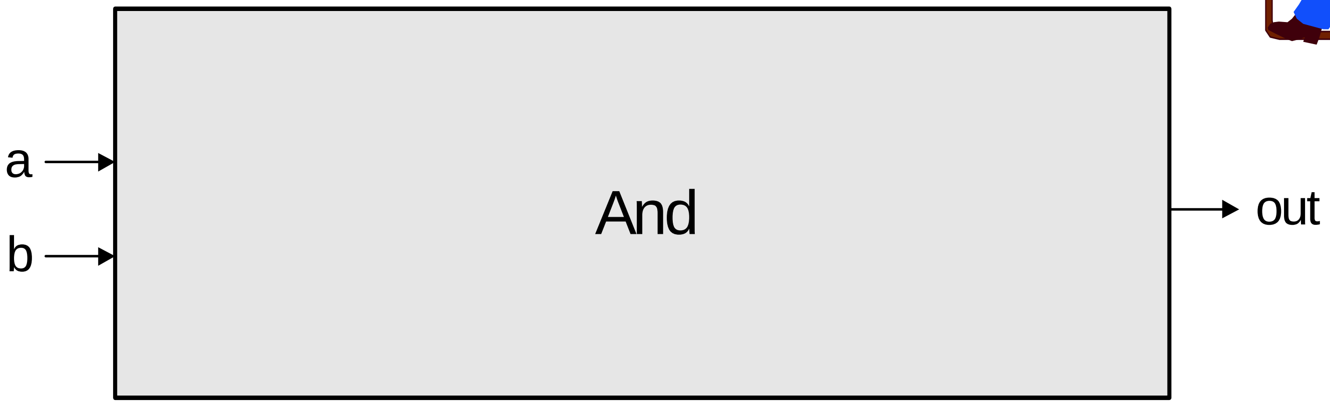
```
CHIP And
{   IN  a, b;
    OUT out;
    // implementation missing
}
```

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



And.hdl

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

Building an **And** gate

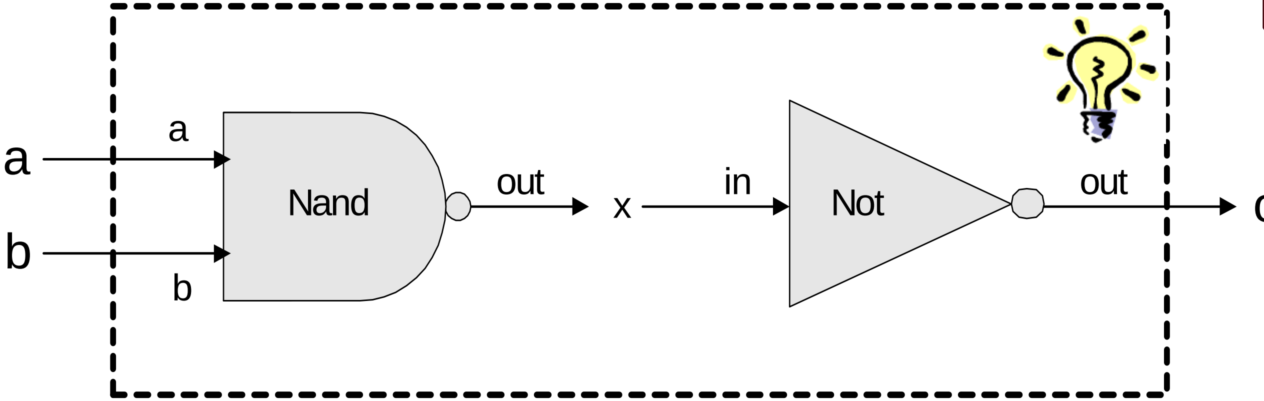
Implementation: And(a,b) = Not(Nand(a,b))

And.hdl

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

Building an **And** gate

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

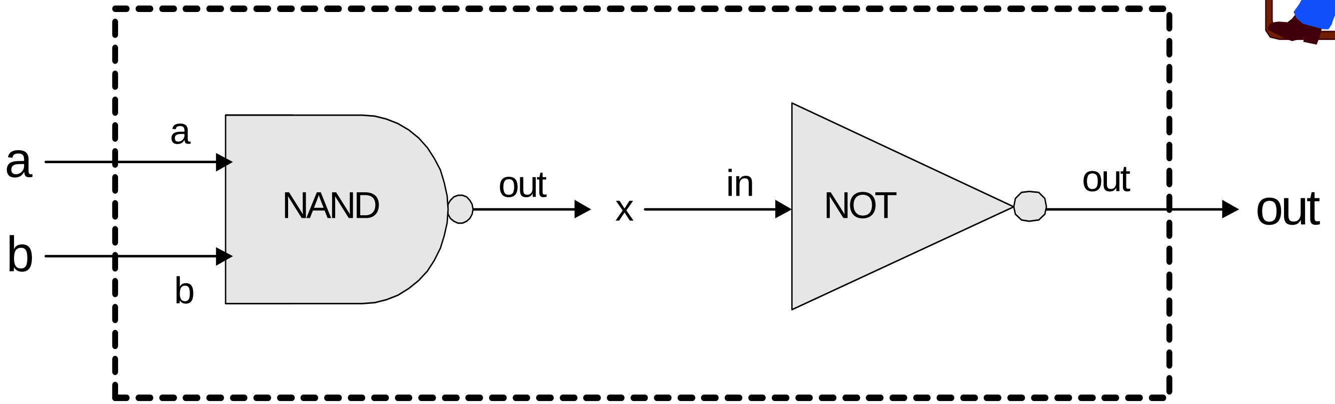


And.hdl

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

Building an **And** gate

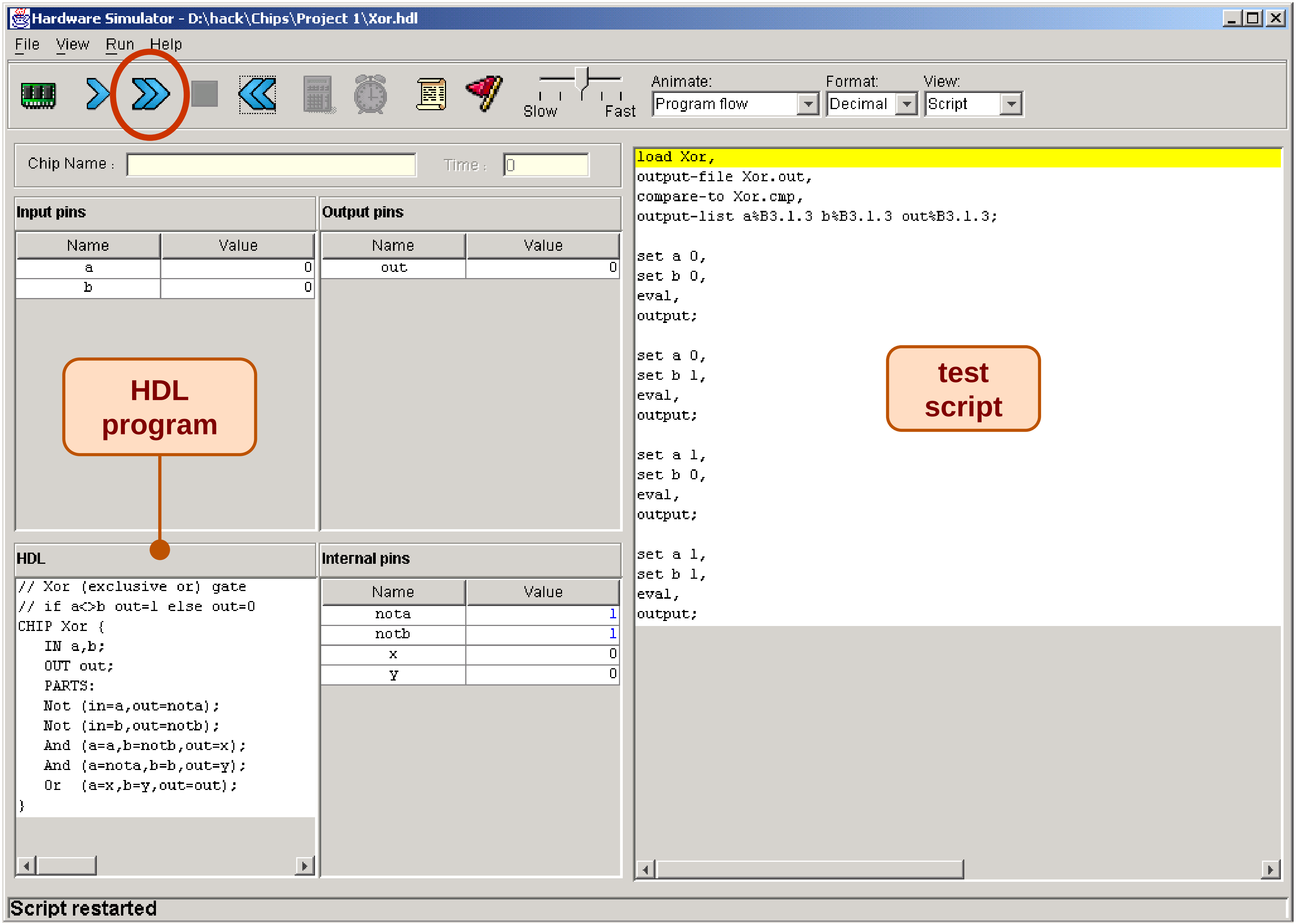
Implementation: And(a,b) = Not(Nand(a,b))



And.hdl

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

Hardware simulator (demonstrating Xor gate construction)



Animate: Program flow

Format: Decimal

View: Script

Chip Name :

Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
// 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

Name	Value
nota	1
notb	1
x	0
y	0

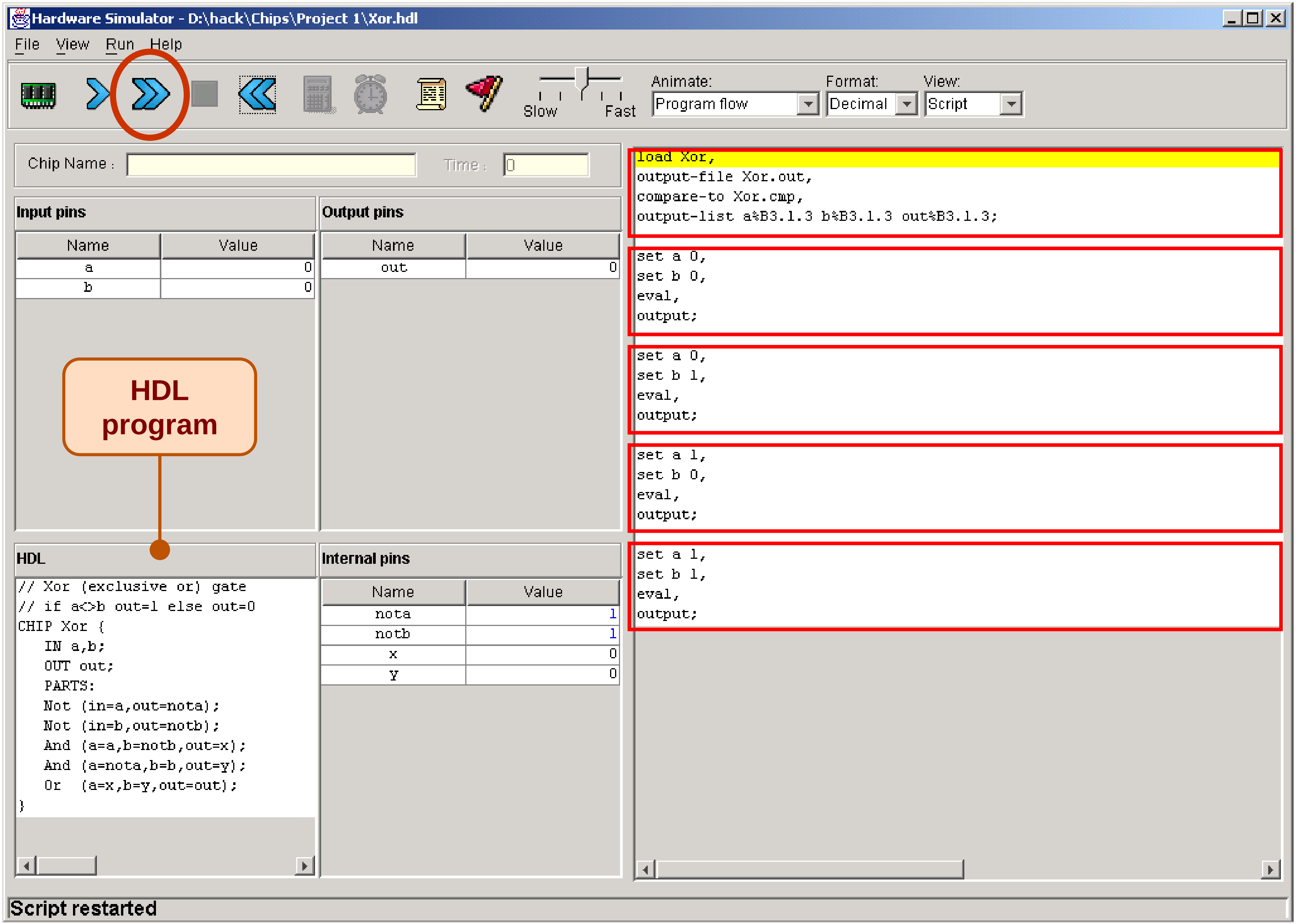
```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

Script restarted

HDL program

test script

Hardware simulator



Animate: Program flow
Format: Decimal
View: Script

Chip Name :
Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
// 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);
}
```

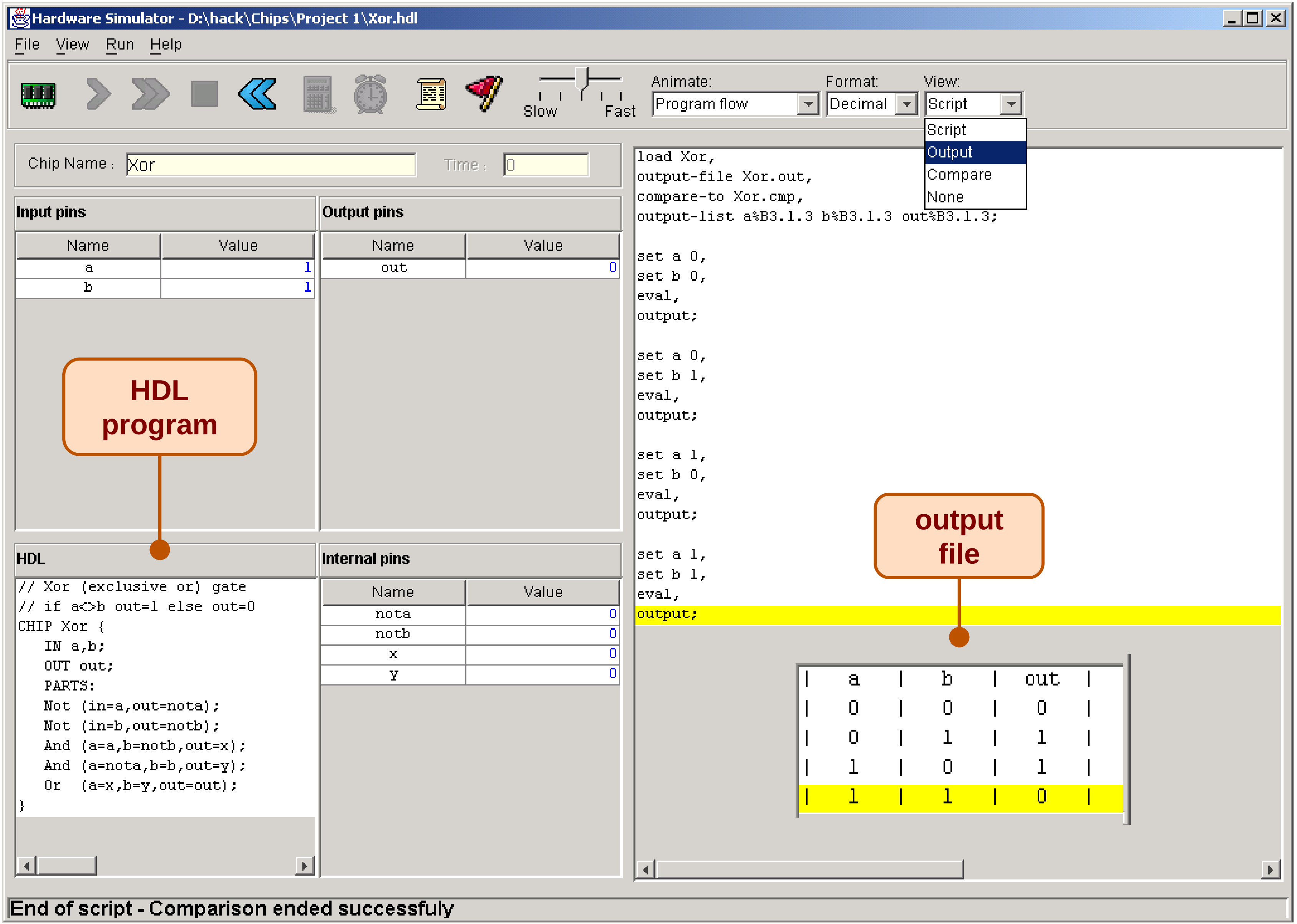
Internal pins

Name	Value
nota	1
notb	1
x	0
y	0

```
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;
```

Script restarted

Hardware simulator



Animate: Program flow

Format: Decimal

View: Script

Chip Name : Xor

Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	0

HDL program

HDL

```
// 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

Name	Value
nota	0
notb	0
x	0
y	0

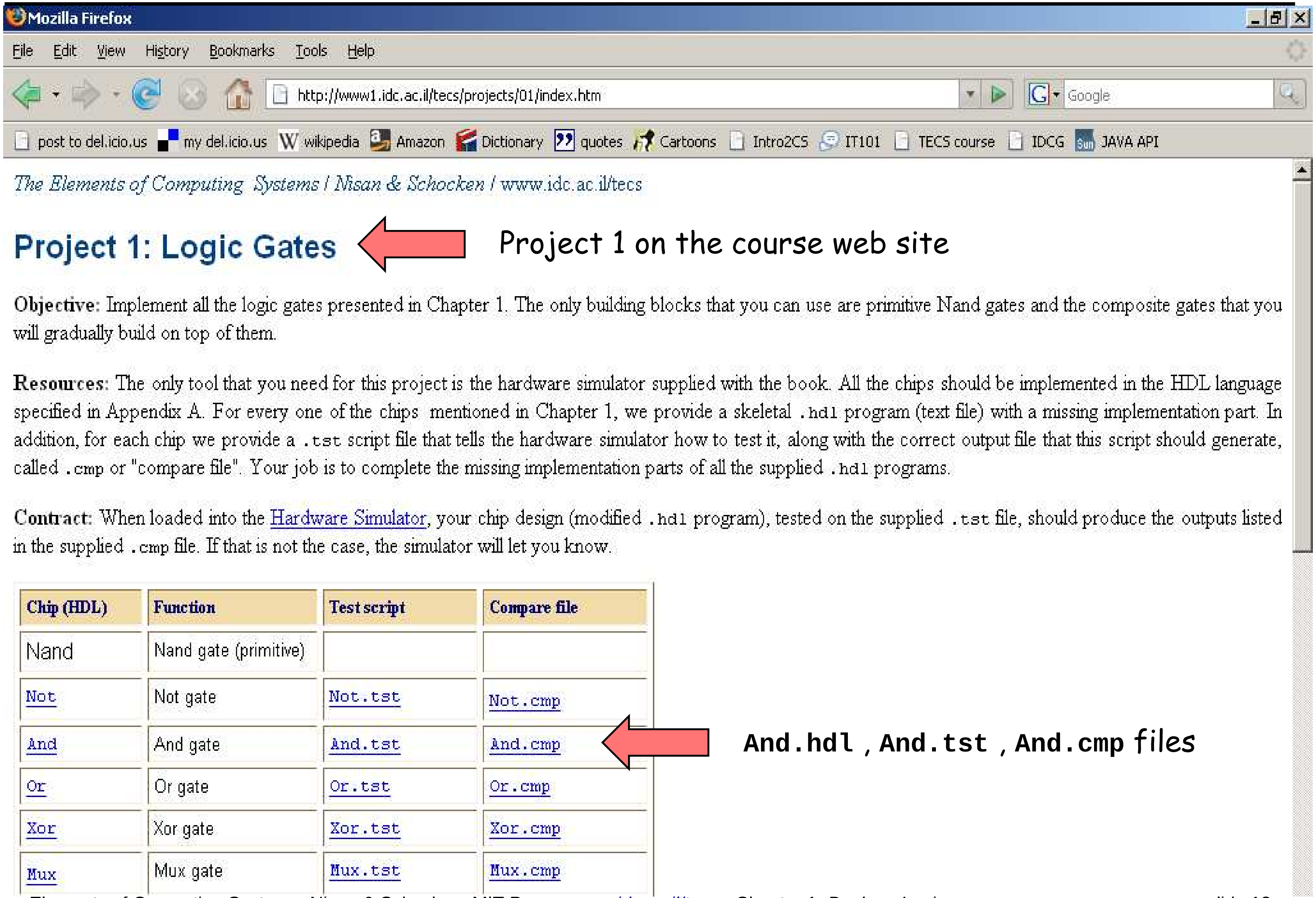
```
load Xor,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

output file

```
|   a   |   b   |  out  |
|   0   |   0   |   0   |
|   0   |   1   |   1   |
|   1   |   0   |   1   |
|   1   |   1   |   0   |
```

End of script - Comparison ended successfuly

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

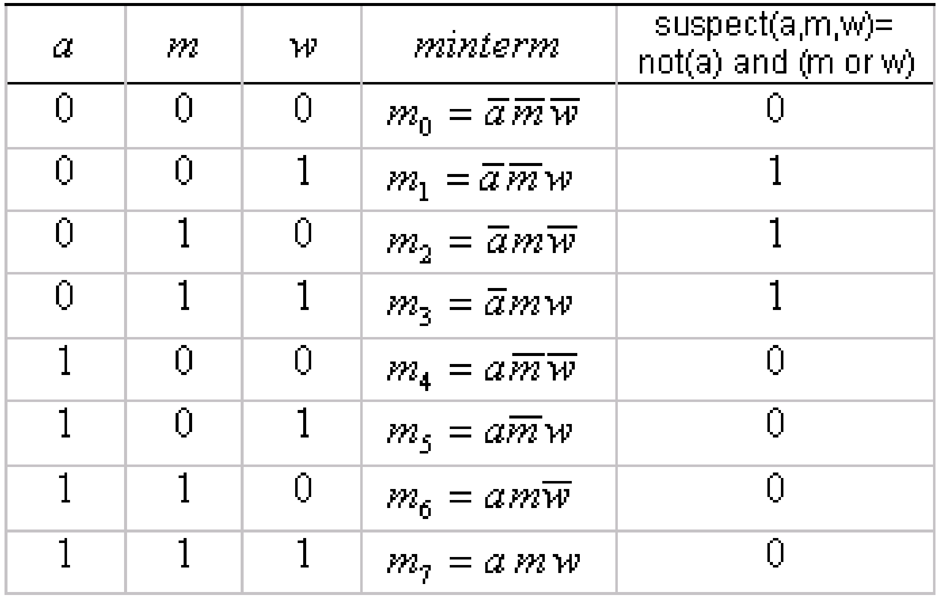
And.hdl , And.tst , And.cmp files

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

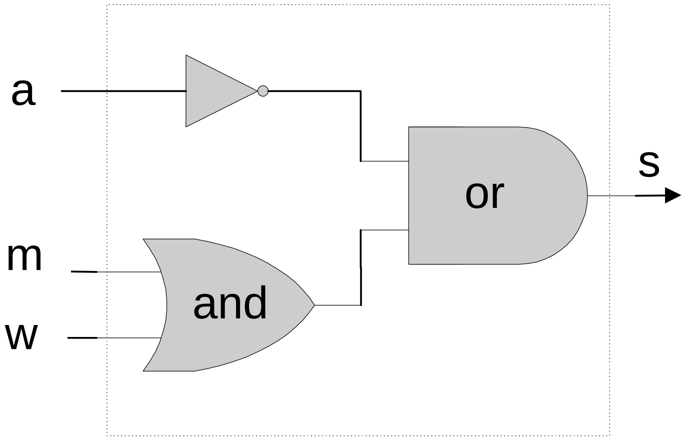


a	m	w	$minterm$	suspect(a,m,w)= not(a) and (m or w)
0	0	0	$m_0 = \bar{a}\bar{m}\bar{w}$	0
0	0	1	$m_1 = \bar{a}\bar{m}w$	1
0	1	0	$m_2 = \bar{a}m\bar{w}$	1
0	1	1	$m_3 = \bar{a}mw$	1
1	0	0	$m_4 = a\bar{m}\bar{w}$	0
1	0	1	$m_5 = a\bar{m}w$	0
1	1	0	$m_6 = am\bar{w}$	0
1	1	1	$m_7 = a\,m\,w$	0

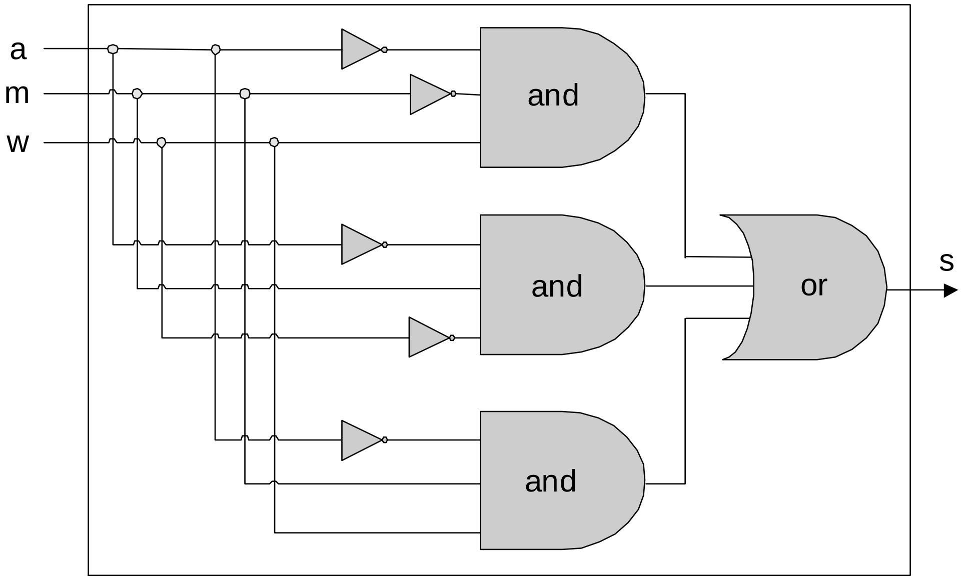
Canonical form: $s(a,m,w) = \bar{a}\,\bar{m}\,w + \bar{a}\,m\bar{w} + \bar{a}\,mw$

End notes: Canonical representation (cont.)

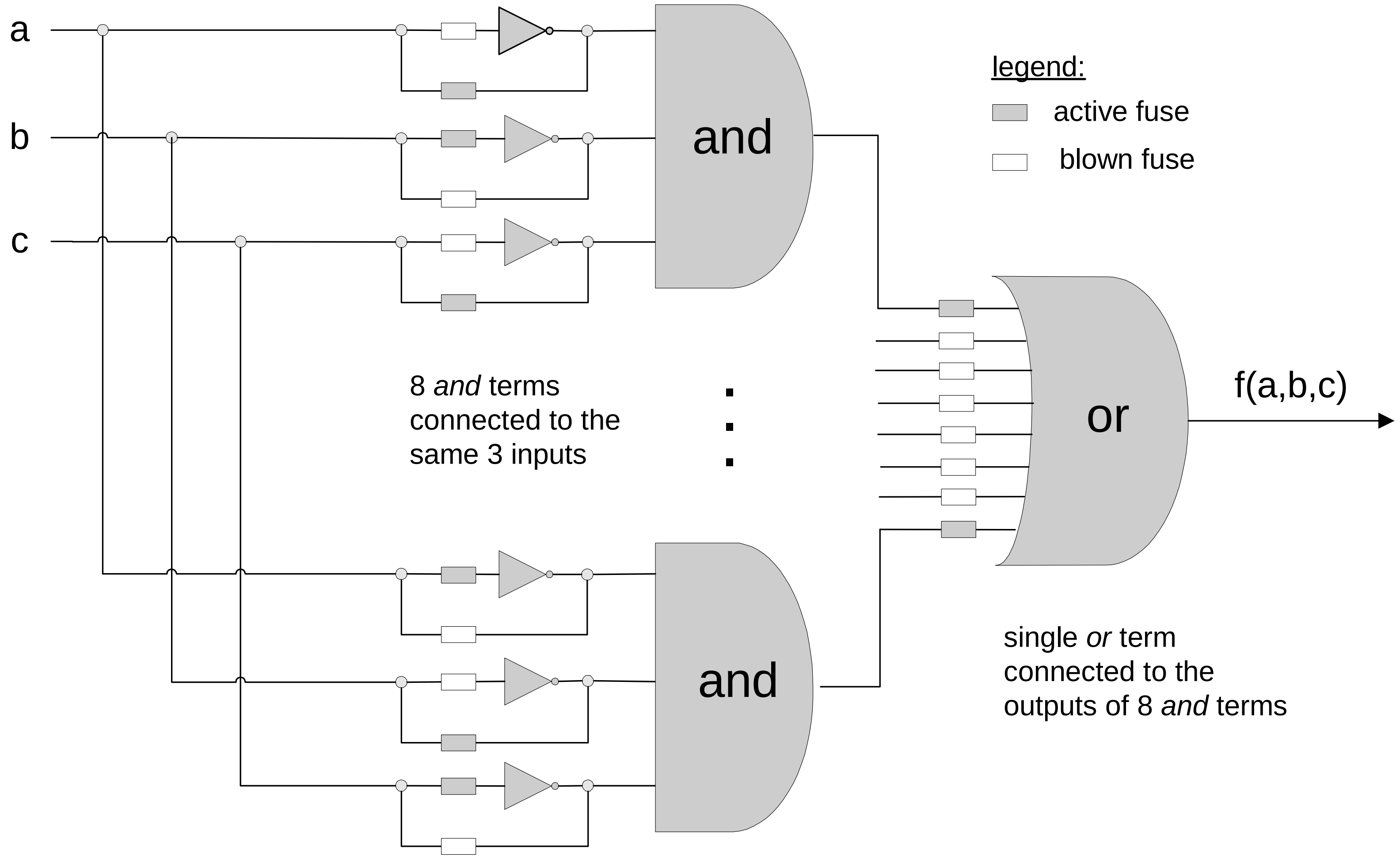
$$s(a,m,w) = \overline{a} \cdot (m + w)$$



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



End notes: Programmable Logic Device for 3-way functions



legend:

▨ active fuse

□ blown fuse

8 *and* terms connected to the same 3 inputs

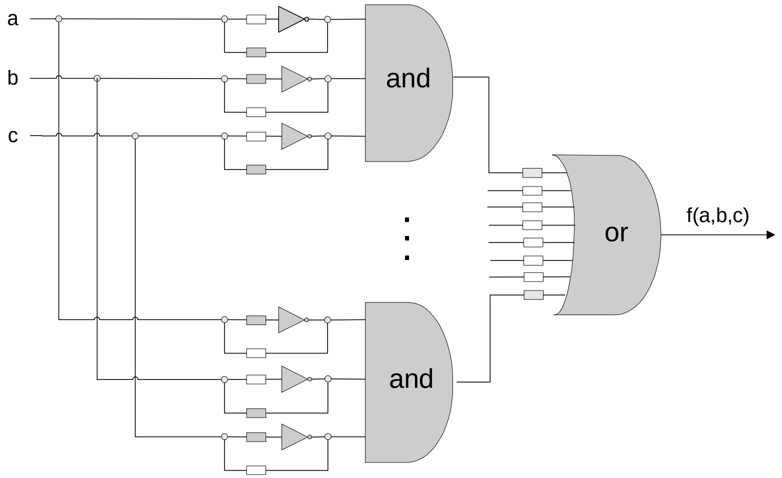
single *or* term connected to the outputs of 8 *and* terms

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

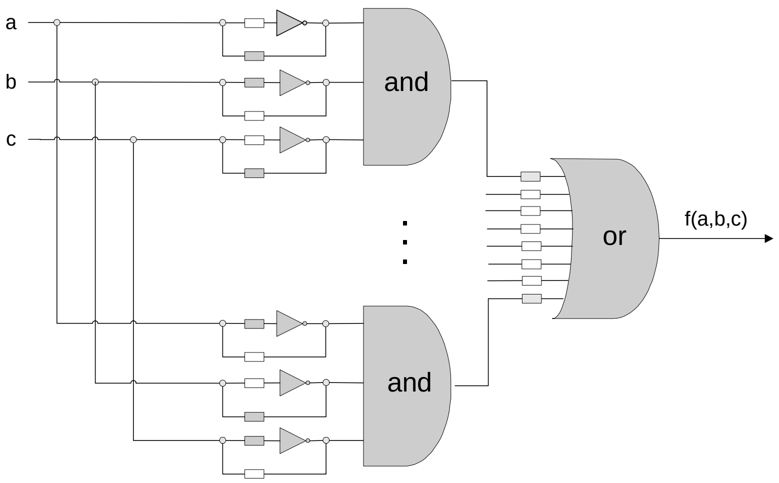
(the on/off states of the fuses determine which gates participate in the computation)

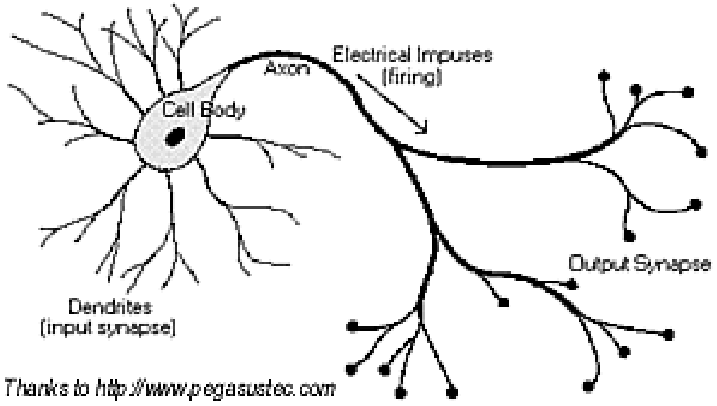
Perspective

- 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, …

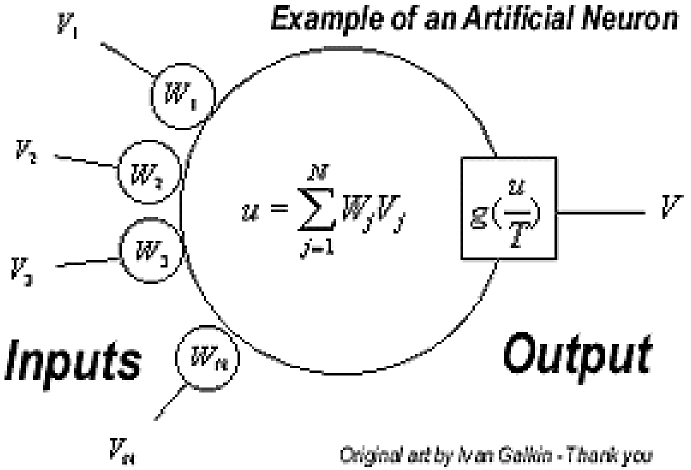


End note: universal building blocks, unique topology





Thanks to http://www.pegasustec.com



Example of an Artificial Neuron

Original art by Ivan Galkin - Thank you