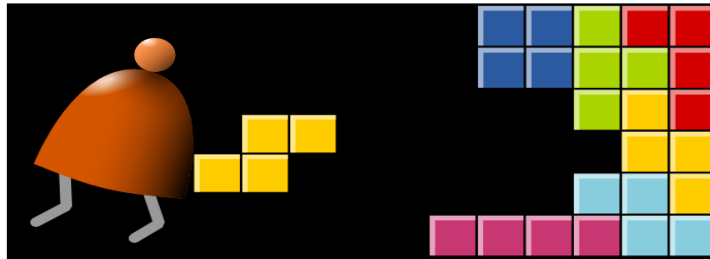


Boolean Logic



Cyber Systems Architecture

SY 303 – Fall AY2021

Usage and Copyright Notice:

Copyright © Noam Nisan and Shimon Schocken

This presentation accompanies the textbook “The Elements of Computing Systems” by Noam Nisan & Shimon Schocken, MIT Press, 2005.

We provide 13 such presentations.

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.

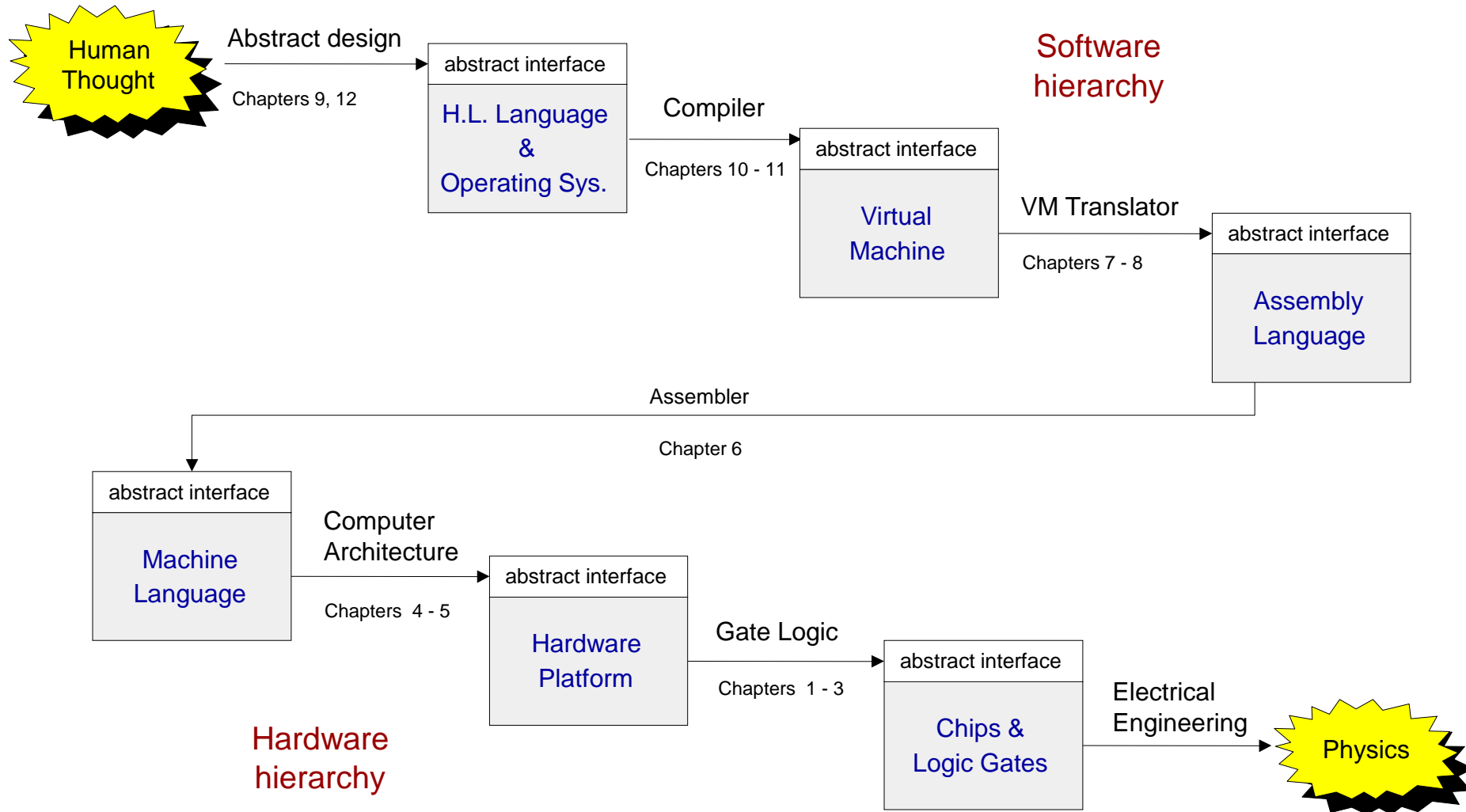
If you use our book and course materials, please include a reference to www.nand2tetris.org

If you have any questions or comments, please write us at nand2tetris@gmail.com



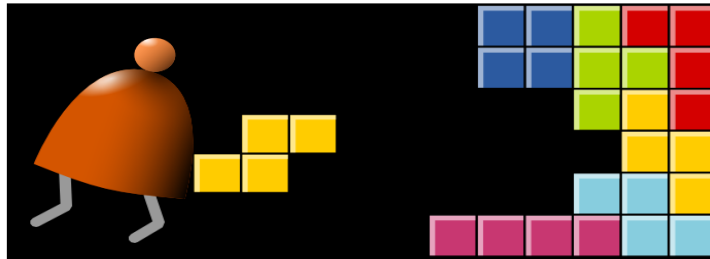
This work is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](http://creativecommons.org/licenses/by-nc-sa/3.0/).

Course theme and structure



(Abstraction–implementation paradigm)

Objectives



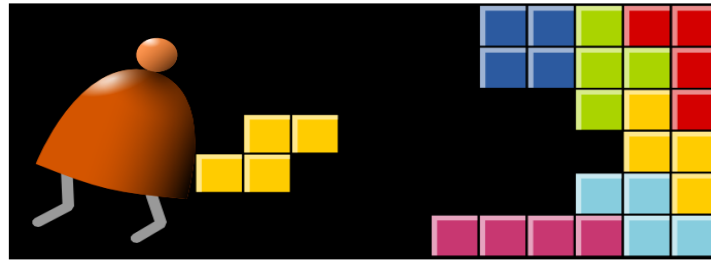
Cyber Systems Architecture

SY 303 – Fall AY2021

Objectives

1. Implement a complete set of logic gates by using a hardware description language to describe each gates logical functions.
2. Implement a complete set of logic gates using only NAND gates as a primitive along with any gates built using NAND gates.
3. Describe logical functions and convert their representations between Boolean expressions, truth tables, and gate diagrams.
4. Verify proper operation of a logic chip using a supplied hardware simulator and test input scripts.
5. Recognize the difference the between the interface to a system and the implementation of a system.

Boolean Functions



Cyber Systems Architecture

SY 303 – Fall AY2021

Boolean algebra

Some elementary Boolean functions:

- 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

- A Boolean function can be expressed using a functional expression or a truth table expression
- Important observation:
Every Boolean function can be expressed using And, Or, Not.

All Boolean functions of 2 variables

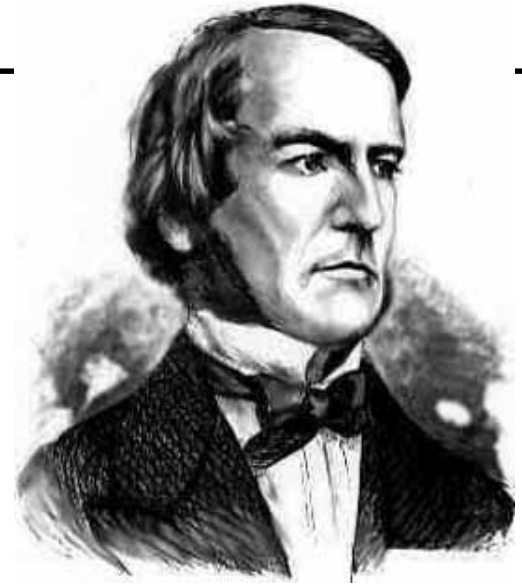
Function	x	0	0	1	1
	y	0	1	0	1
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \bar{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\bar{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \bar{y} + \bar{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x + y}$	1	0	0	0
Equivalence	$x \cdot y + \bar{x}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

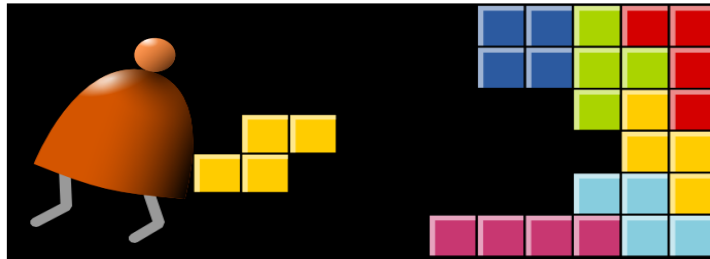
Whodunit story: Each suspect may or may not have an alibi (a), a motivation to commit the crime (m), and a relationship to the weapon found in the scene of the crime (w). The police decides to focus attention only on suspects for whom the proposition **Not(a) And (m Or w)** is true.

Truth table of the "suspect" 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: $s(a, m, w) = \bar{a} \bar{m} w + \bar{a} m \bar{w} + \bar{a} m w$

Gate Logic



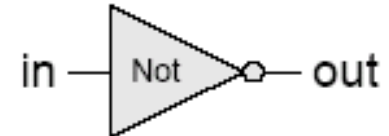
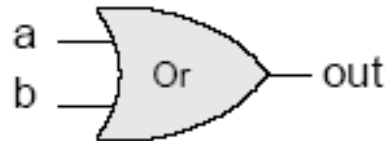
Cyber Systems Architecture

SY 303 – Fall AY2021

Gate logic

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

- Elementary gates:



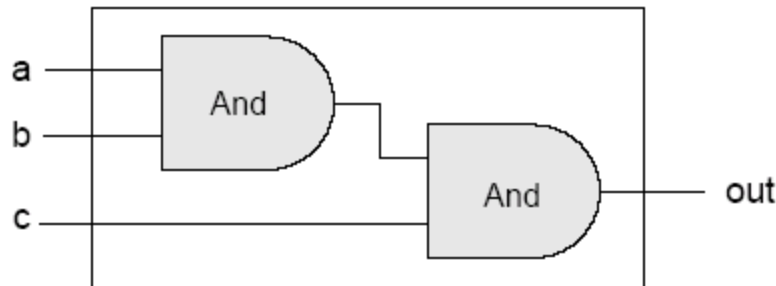
- Composite gates:

Gate interface



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

Gate implementation



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

Gates as Building Blocks



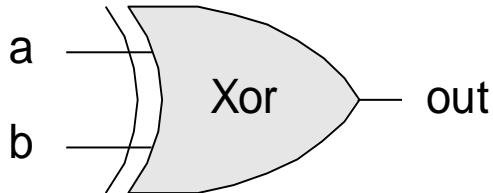
Gate logic



Claude Shannon, 1916-2001

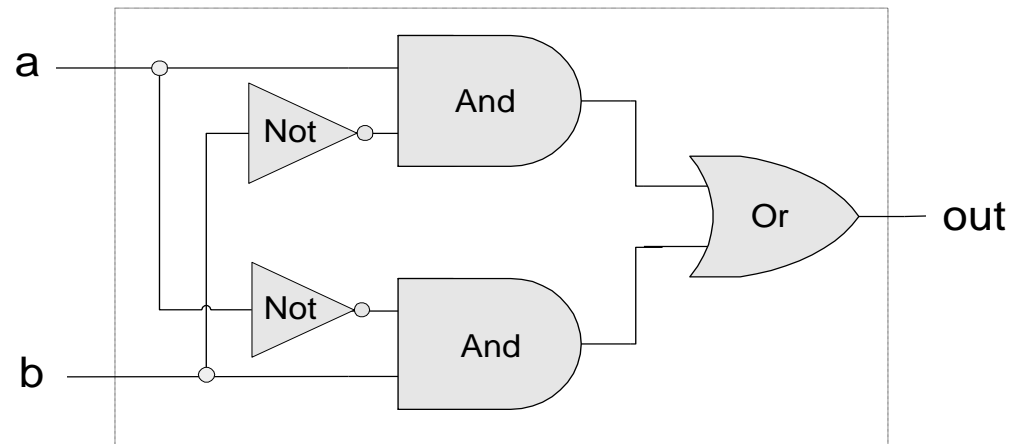
("Symbolic Analysis of Relay and Switching Circuits")

Interface



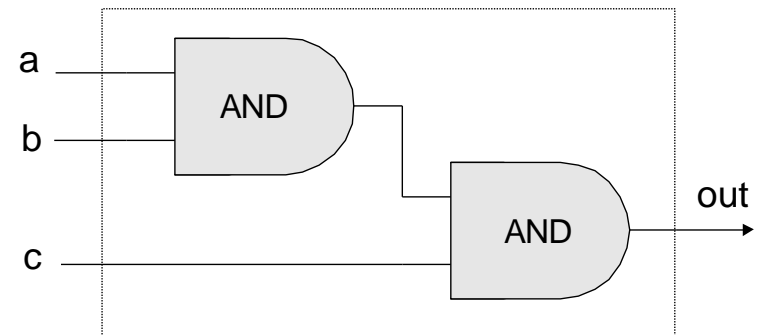
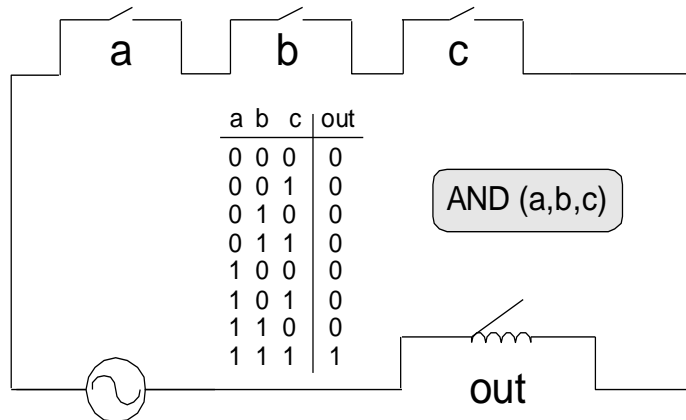
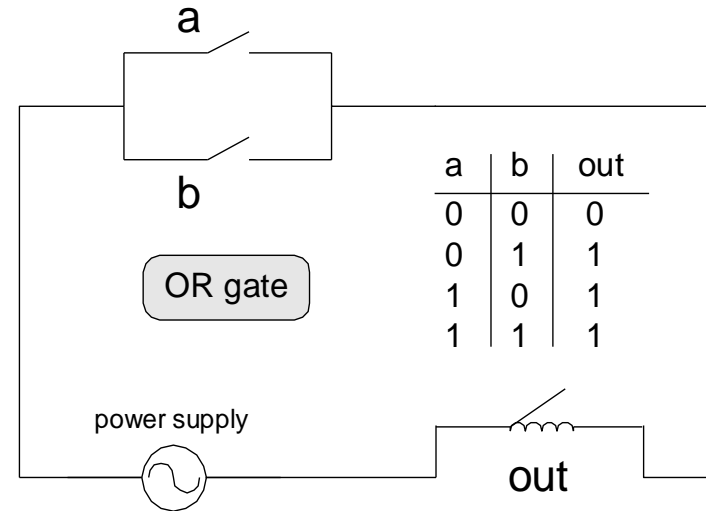
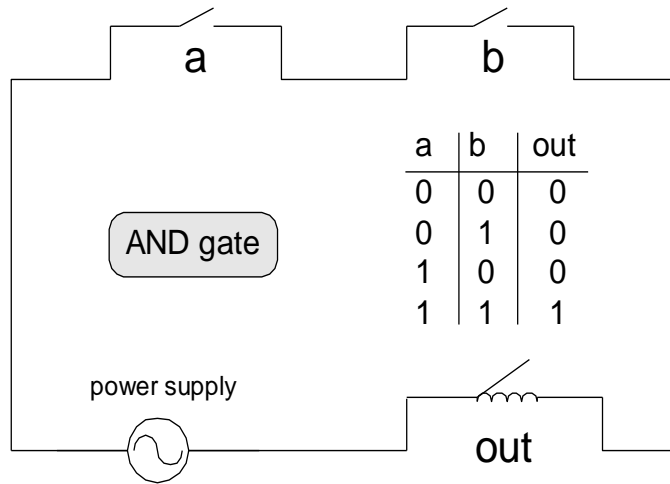
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation



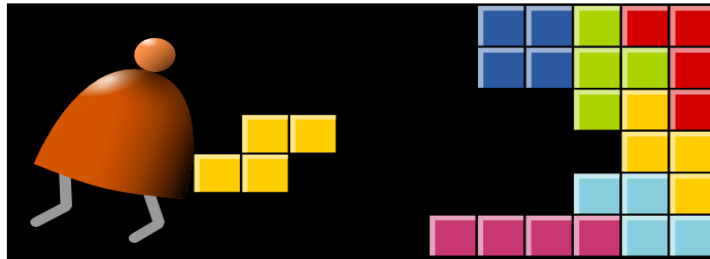
$$\text{Xor}(a,b) = \text{Or}(\text{And}(a,\text{Not}(b)),\text{And}(\text{Not}(a),b))$$

Circuit implementations



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

Project 1 Overview



Cyber Systems Architecture

SY 303 – Fall AY2021

Project 1: elementary logic gates

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

Build:

a	b	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.

Q: Why these particular 12 gates?

A: Since ...

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

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



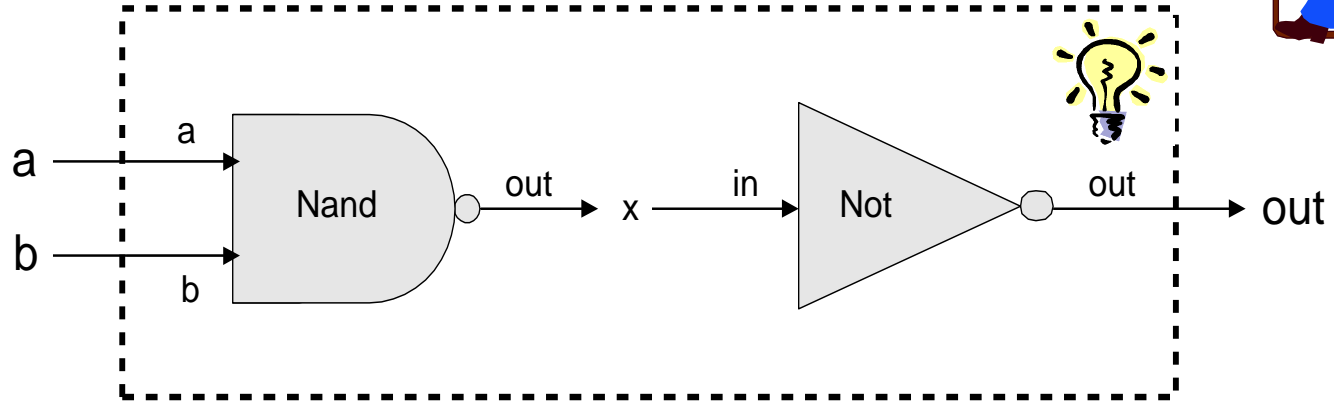
And.hdl

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

Building an And gate



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



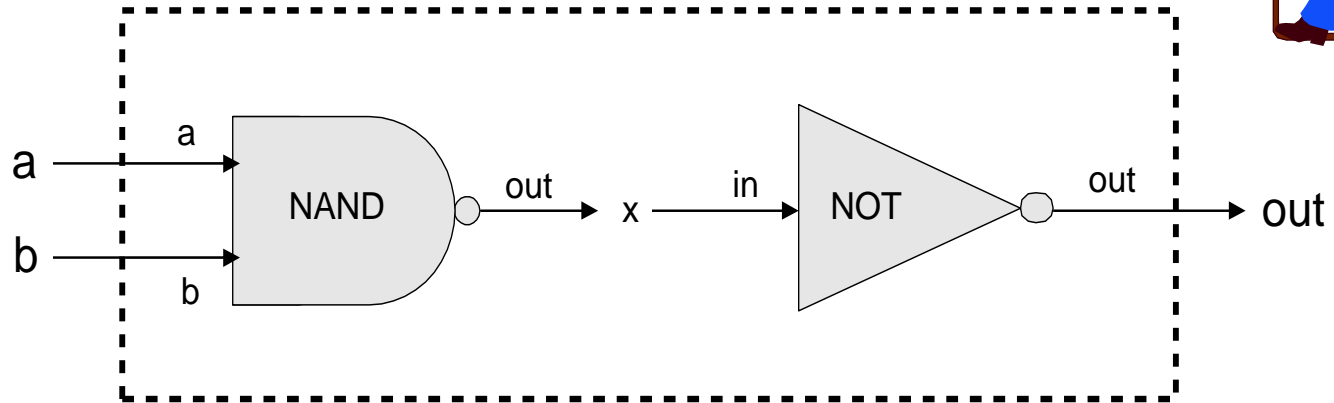
And.hdl

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

Building an And gate



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



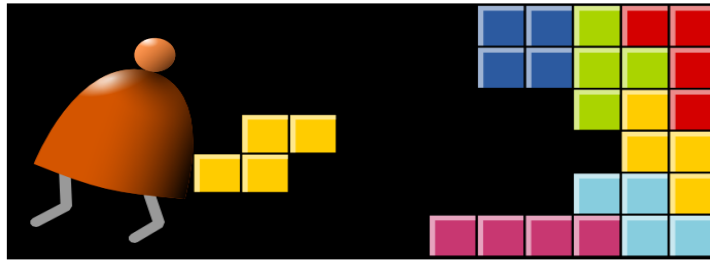
And.hdl

```
CHIP And
{
  IN  a, b;
  OUT out;

  Nand(a = a,
        b = b,
        out = x);
  Not(in = x, out = out)
}
```



Hardware Simulator Demo



Cyber Systems Architecture

SY 303 – Fall AY2021

Hardware simulator (demonstrating Xor gate construction)

The screenshot shows the Hardware Simulator window with the title "Hardware Simulator - D:\hack\Chips\Project 1\Xor.hdl". The menu bar includes File, View, Run, and Help. The toolbar contains icons for loading, running, and pausing, with the run icon (a blue double arrow) circled in red. Below the toolbar, there are fields for "Chip Name" and "Time". The main interface is divided into several sections:

- Input pins:** A table with columns "Name" and "Value". It contains two rows: "a" with value 0 and "b" with value 0.
- Output pins:** A table with columns "Name" and "Value". It contains one row: "out" with value 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);
}
```
- Internal pins:** A table with columns "Name" and "Value". It contains four rows: "nota" with value 1, "notb" with value 1, "x" with value 0, and "y" with value 0.
- Script:** A text area containing 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;
```

Two orange callout boxes are present: one labeled "HDL program" pointing to the HDL code area, and another labeled "test script" pointing to the script area. At the bottom left, a status bar indicates "Script restarted".

Hardware simulator

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

- Toolbar:** Includes icons for loading, running, and pausing. The run button (a blue arrow) is circled in red.
- Chip Name:** A text field containing "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=notb);
    And (a=a,b=notb,out=x);
    And (a=nota,b=b,out=y);
    Or (a=x,b=y,out=out);
}
```
- Internal pins table:**

Name	Value
nota	1
notb	1
x	0
y	0
- Script Execution Log:** A list of commands and their results, each in a red-bordered box:
 - 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".

Hardware simulator

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

- Chip Name:** Xor
- Time:** 0
- Input pins:**

Name	Value
a	1
b	1
- 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=noth);
    And (a=a,b=noth,out=x);
    And (a=nota,b=b,out=y);
    Or (a=x,b=y,out=out);
}
```
- Internal pins:**

Name	Value
nota	0
noth	0
x	0
y	0
- 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;

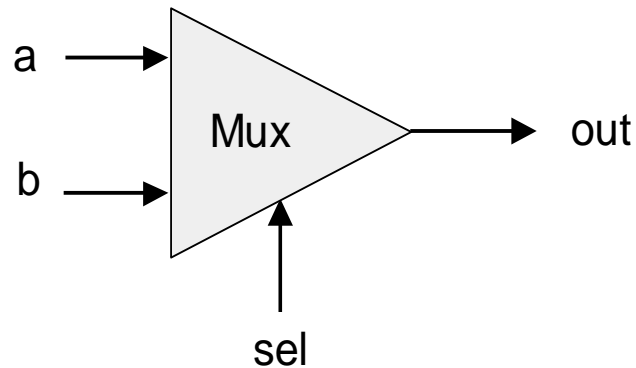
set a 1,
set b 1,
eval,
output;
```
- Output file:**

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

End of script - Comparison ended successfully

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.

Project 1 tips

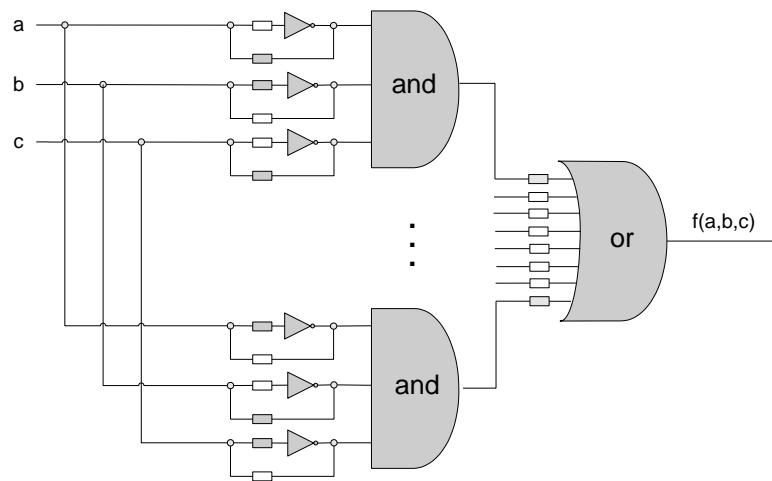
- Read the Introduction + Chapter 1 of the book
- Read Appendix A, sections A1-A6 to get familiar with HDL
- Download the book's software suite
- Go through the hardware simulator tutorial
- You're in business.

Project 1

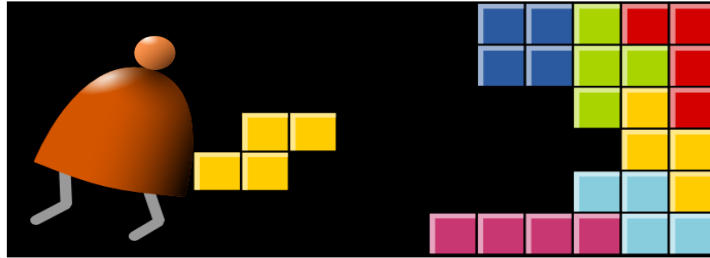
- Implement the logic gates presented in chapter 1.
- You may only use the primitive Nand gates and the composite gates that your team builds with them.
- Submit your working code and gate diagrams to Blackboard
 - If you need the extended time, submit the full project report

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, ...



Instructor Demo



Project 1

Not, Xor