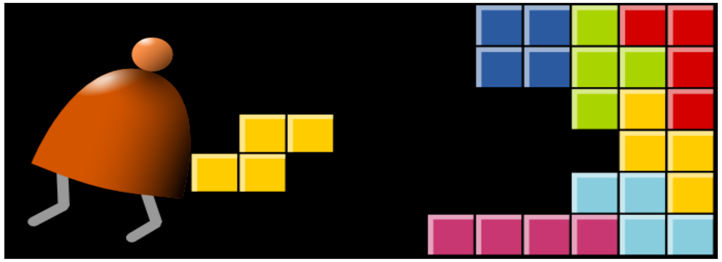
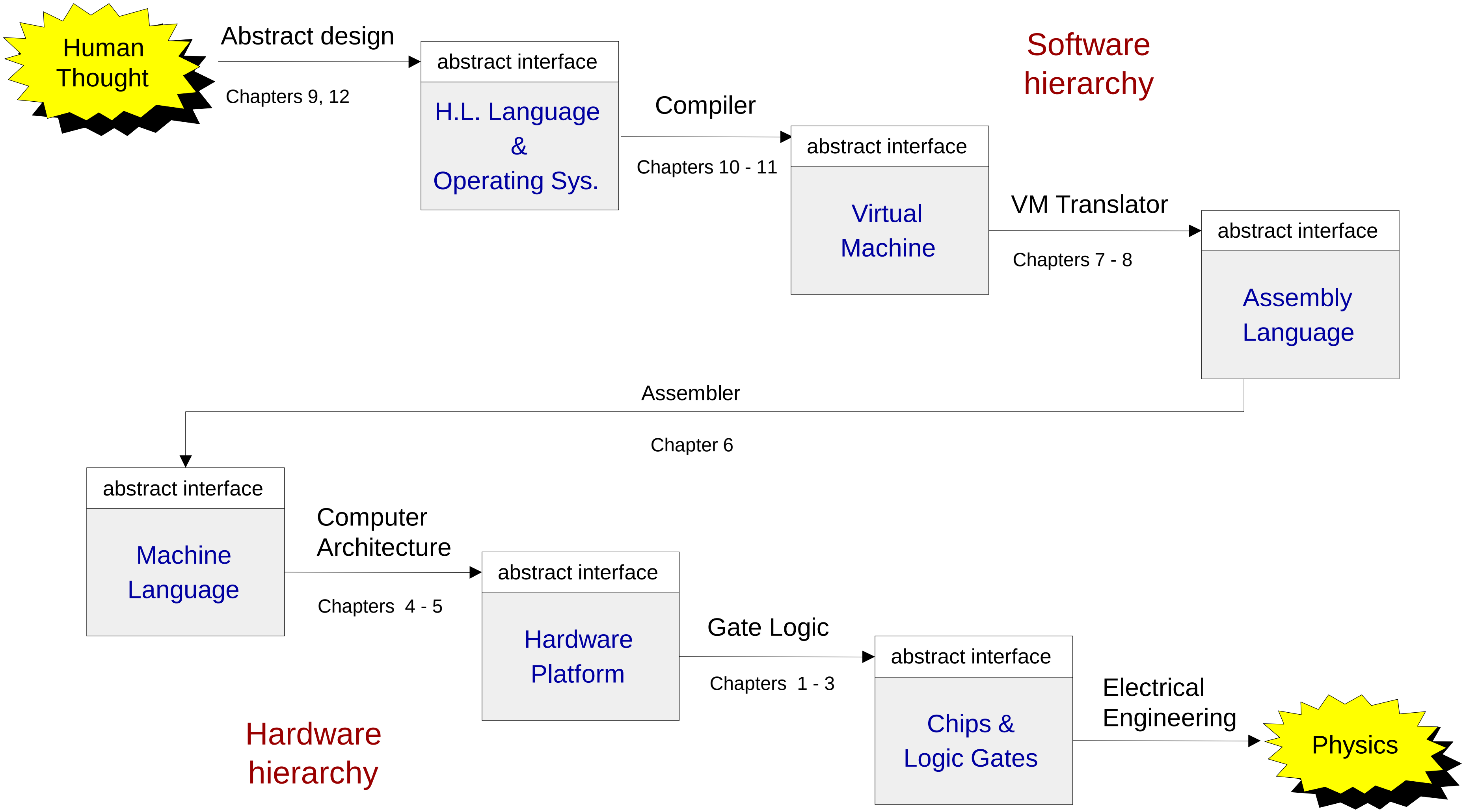
Boolean Logic



Cyber Systems Architecture

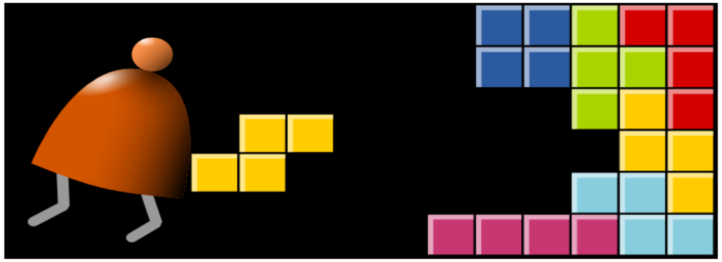
SY 303 – Fall AY2021

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.

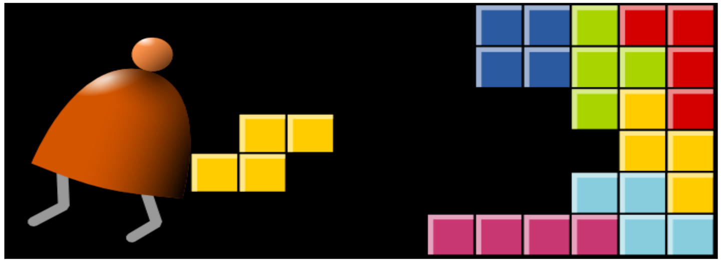
2. Implement a complete set of logic gates using only NAND gates as a primitive along with any gates built using NAND gates.

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:

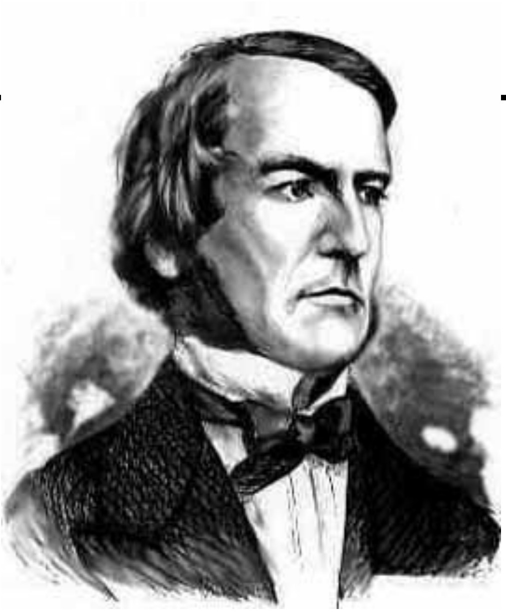
x	y	z	$f(x,y,z) = (x+y)\bar{z}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- 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



George Boole, 1815-1864

(*"A Calculus of Logic"*)

<ins>Given:</ins> `Nand(a,b), false`

<ins>We can build:</ins>

- `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.

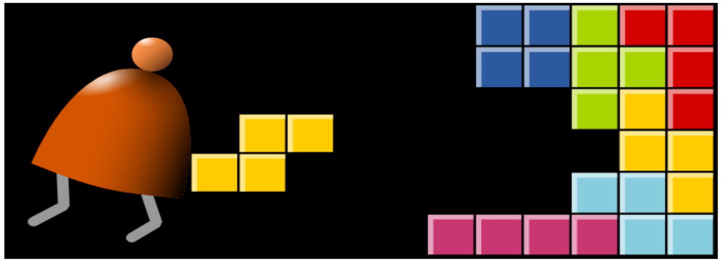
Whodunit story: Each suspect may or may not have an alibi (a), a motivation to commit the crime (m), and a relationship to the weapon found in the scene of the crime (w). The police decides to focus attention only on suspects for whom the proposition Not(a) And (m Or w) is true.

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



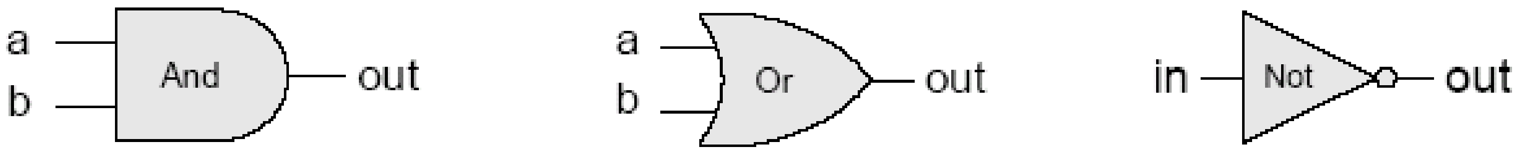
Cyber Systems Architecture

SY 303 – Fall AY2021

Gate logic

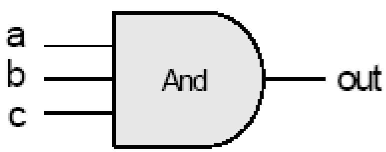
- Gate logic – a gate architecture designed to implement a Boolean function

- Elementary gates:



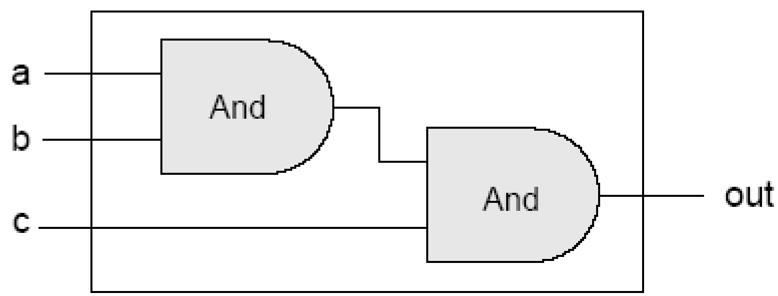
- Composite gates:

Gate interface



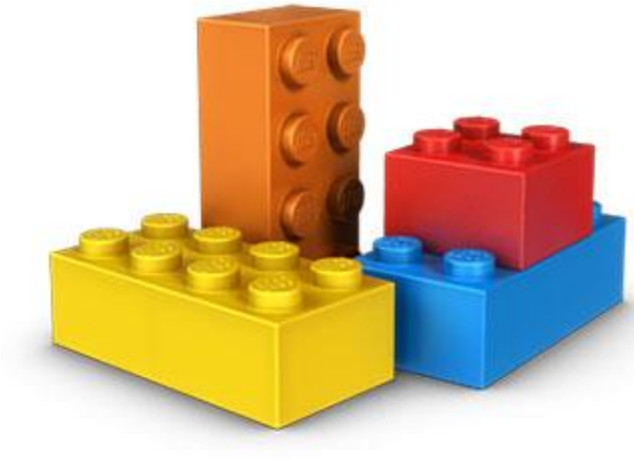

```
If a=b=c=1 then out=1
else out=0
```

Gate implementation



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

Gates as Building Blocks

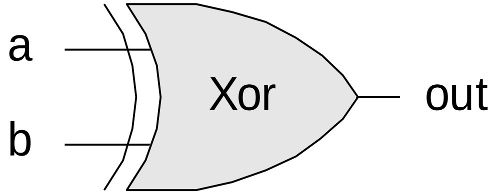


Gate logic

Claude Shannon, 1916-2001

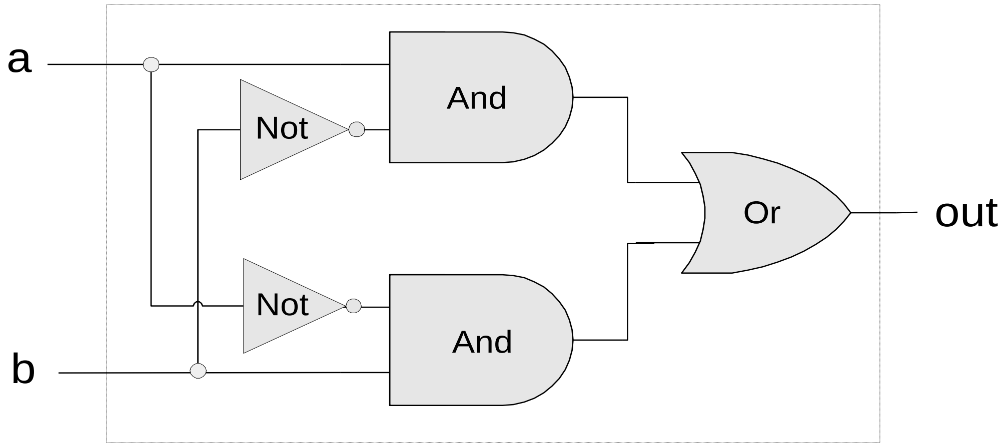
("*Symbolic Analysis of Relay and Switching Circuits*")

Interface



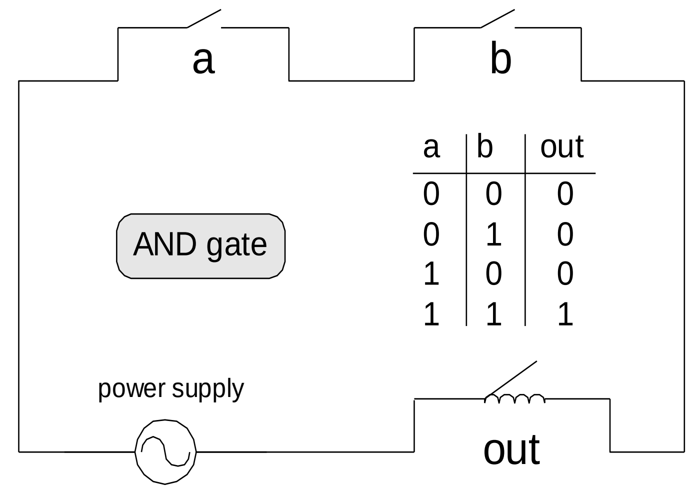
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Implementation

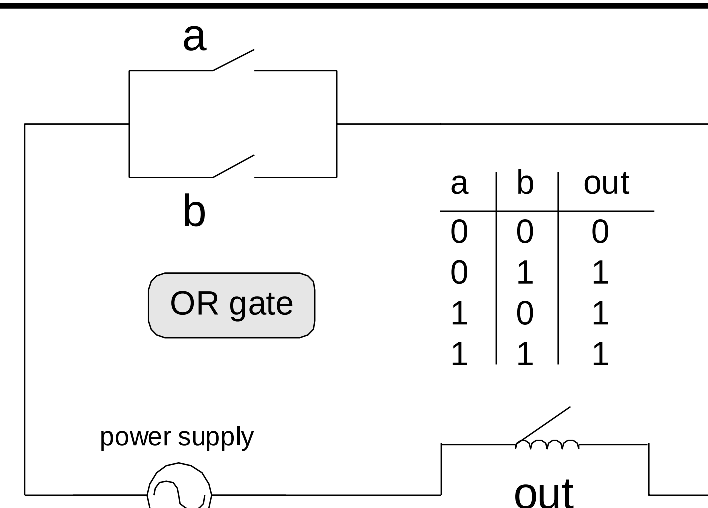


Xor(a,b) = Or(And(a,Not(b)),And(Not(a),b)))

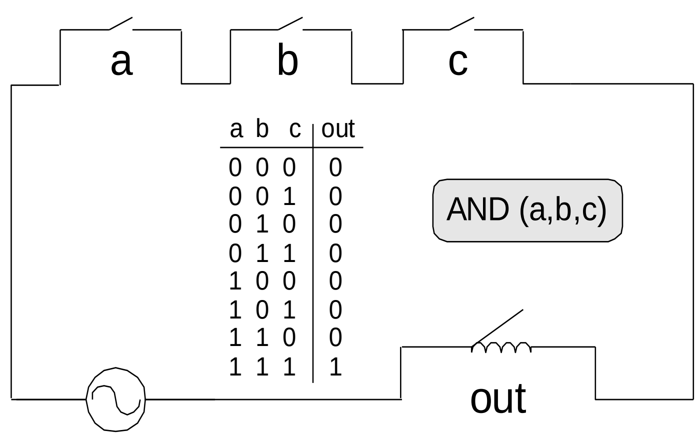
Circuit implementations



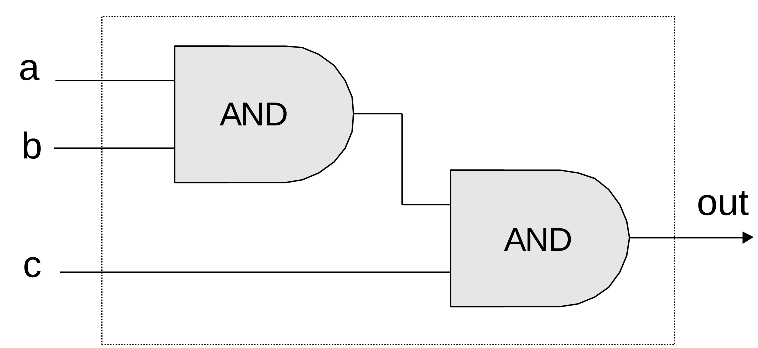
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



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

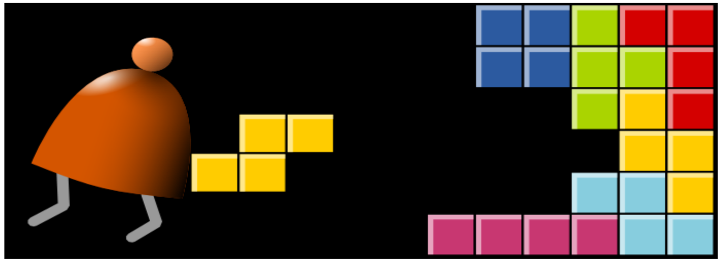


a	b	c	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



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

Project 1 Overview



Cyber Systems Architecture

SY 303 – Fall AY2021

Project 1: elementary logic gates

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

Build:

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

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

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

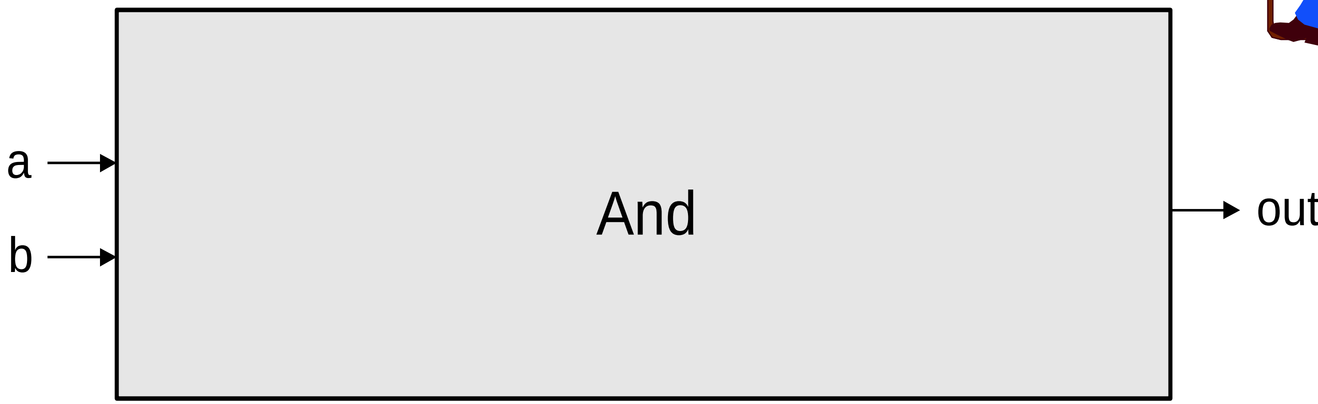
```
CHIP And
{   IN  a, b;
    OUT out;
    // implementation missing
}
```

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

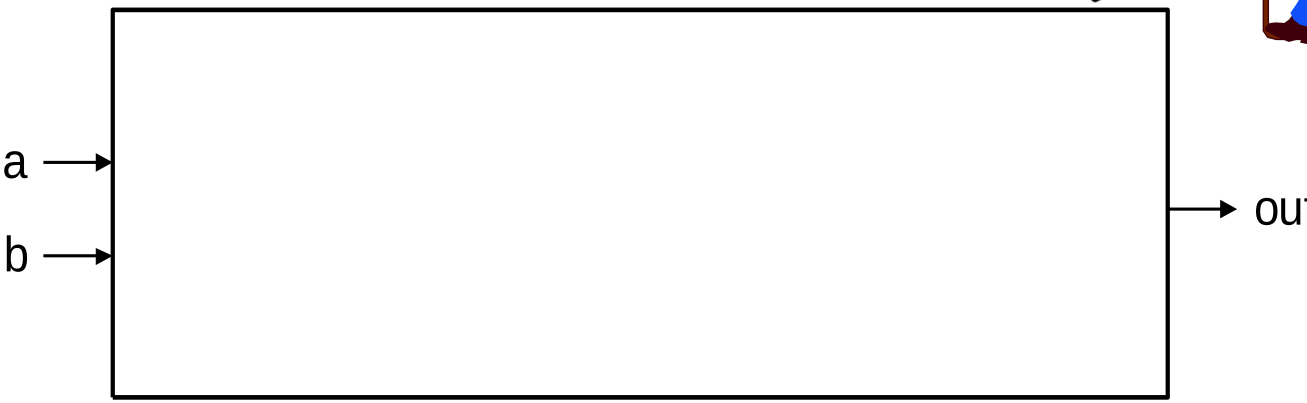


And.hdl

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

Building an **And** gate

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

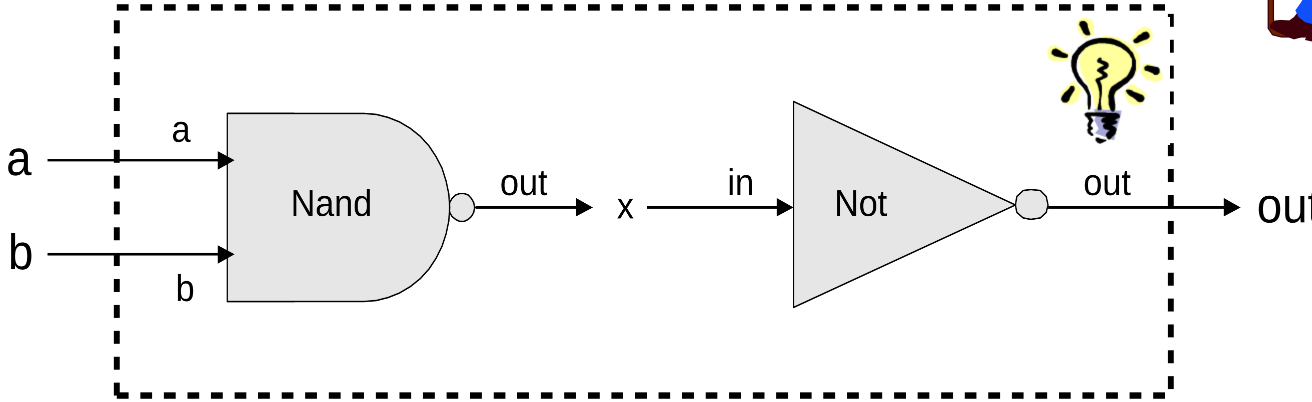


And.hdl

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

Building an **And** gate

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

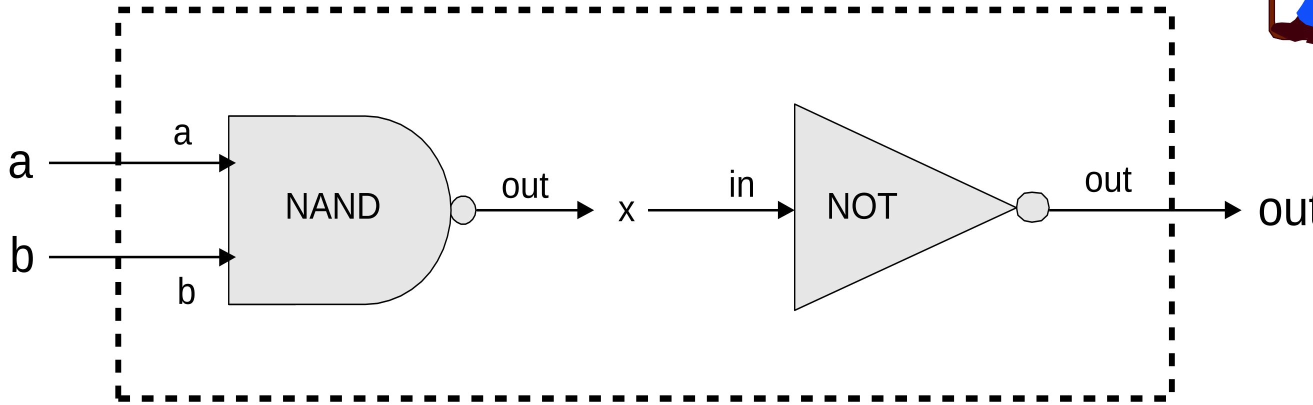


And.hdl

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

Building an **`And`** gate

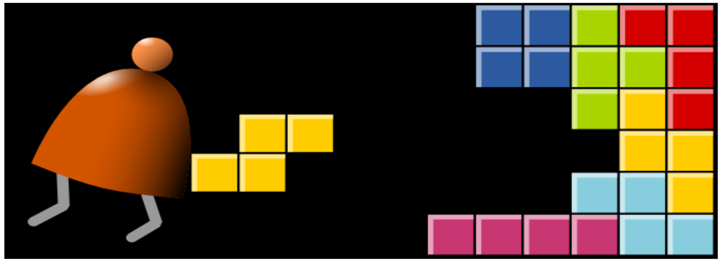
Implementation: And(a,b) = Not(Nand(a,b))



And.hdl

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

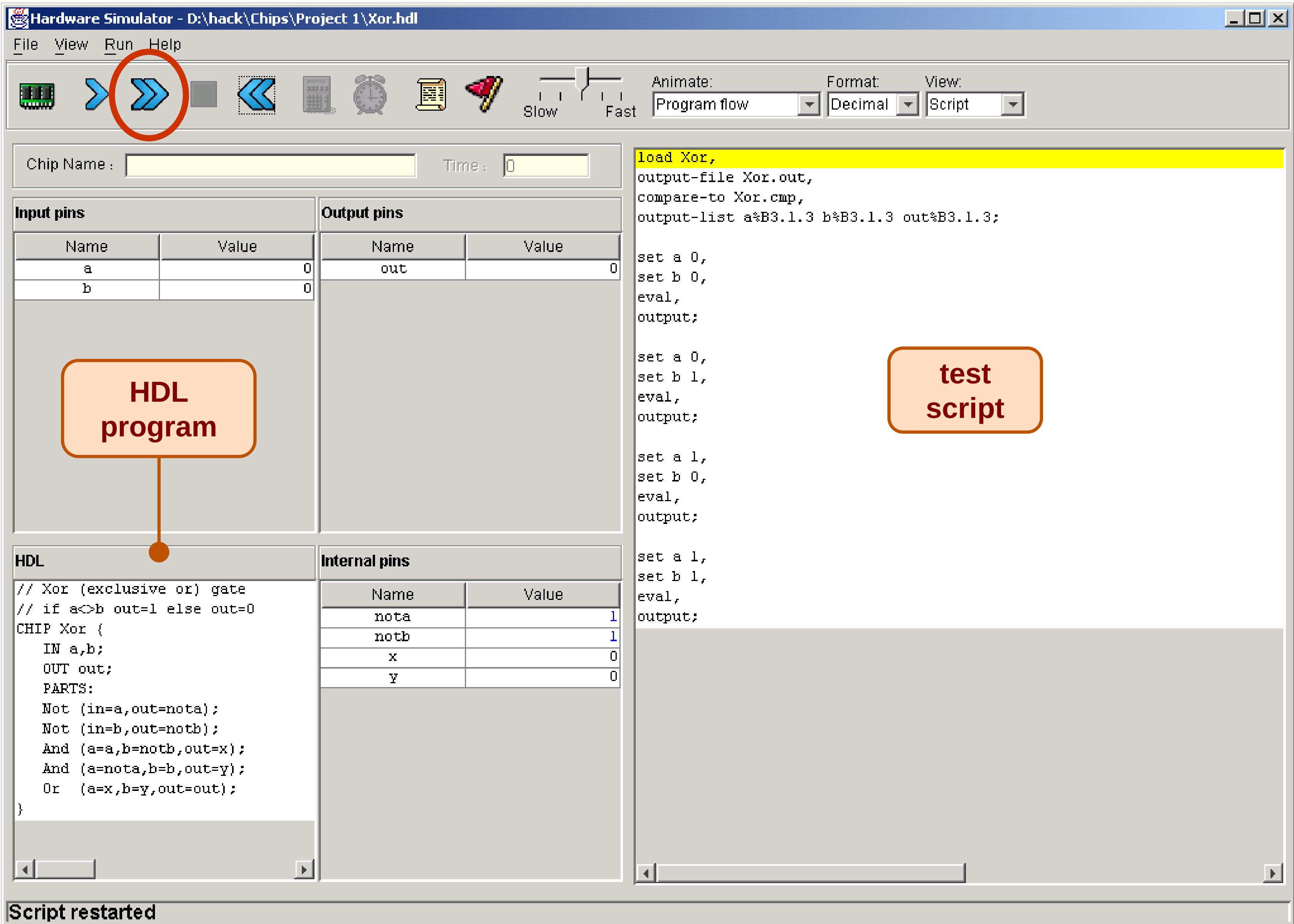
Hardware Simulator Demo



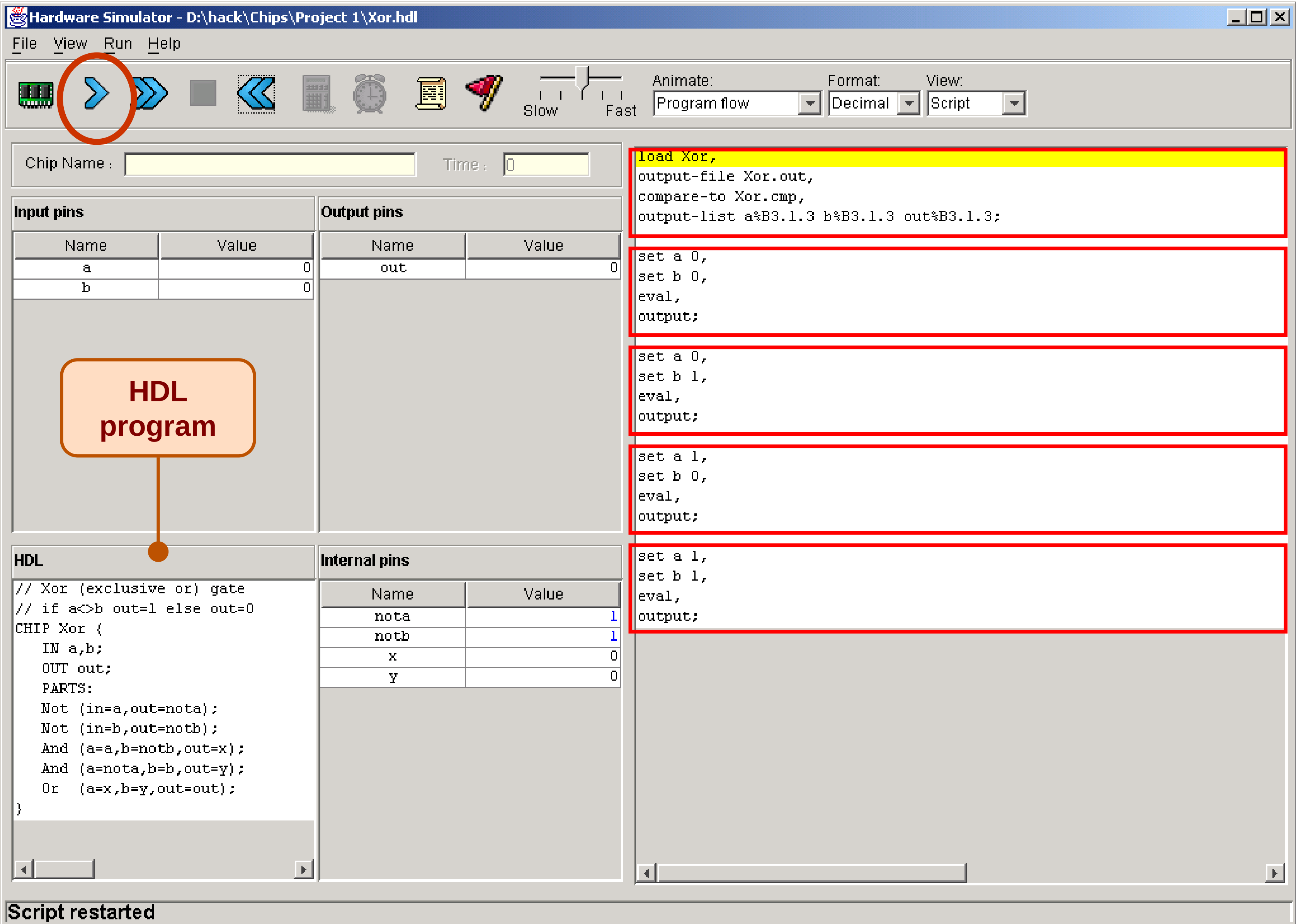
Cyber Systems Architecture

SY 303 – Fall AY2021

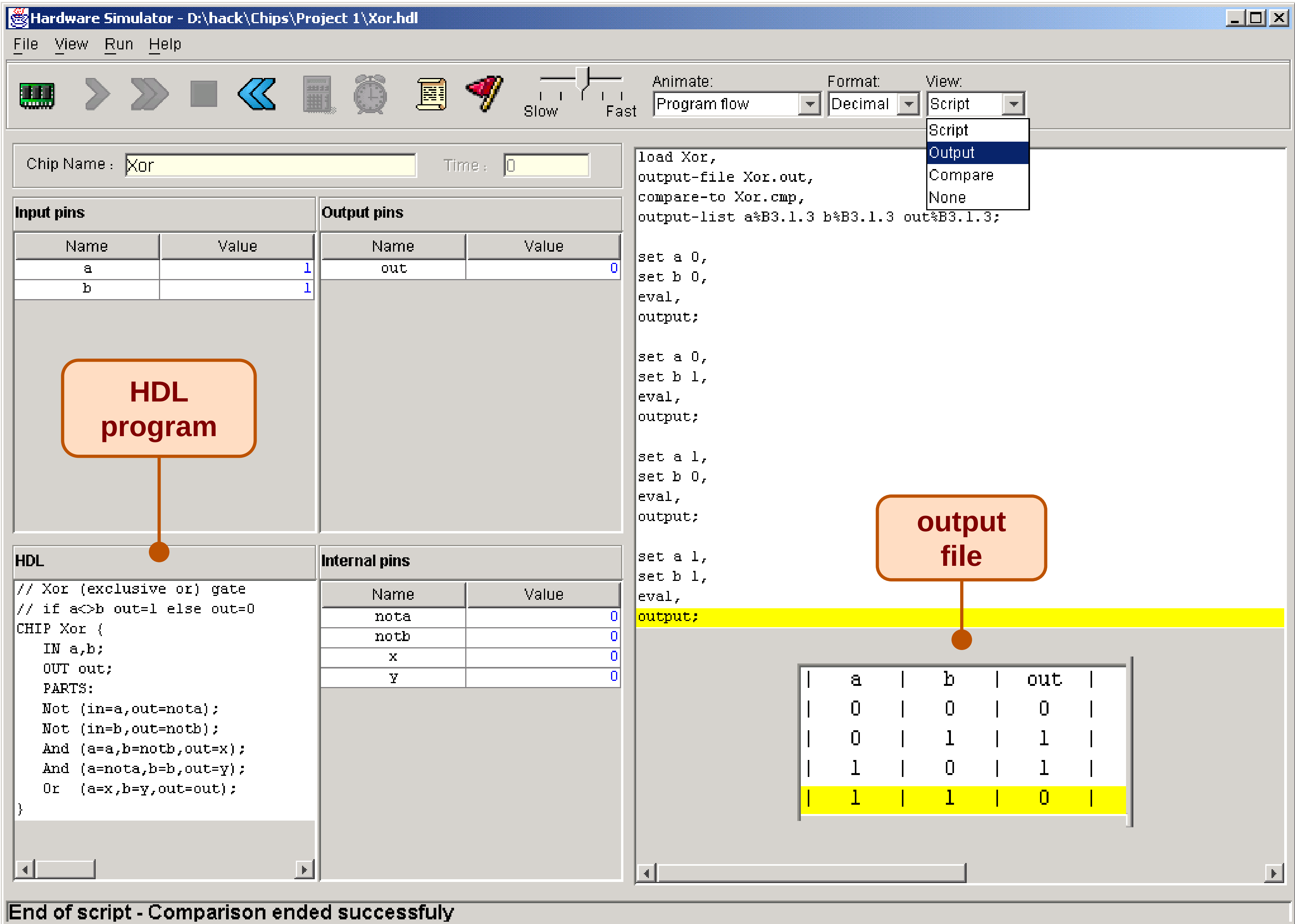
Hardware simulator (demonstrating Xor gate construction)



Hardware simulator

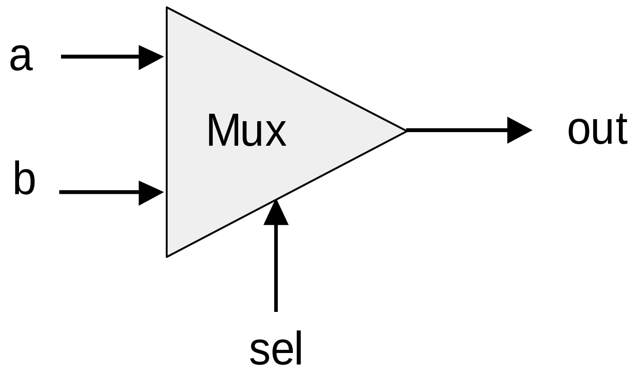


Hardware simulator



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

<u>Proposed Implementation:</u> based on Not, And, Or gates.

Project 1 tips

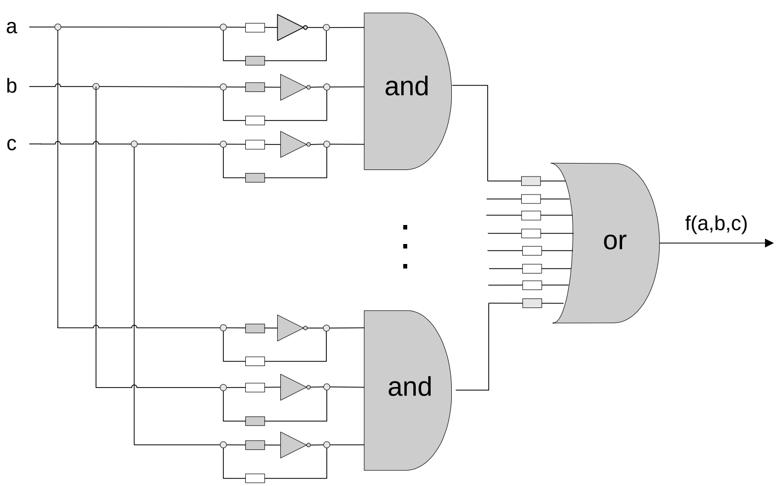
- Read the Introduction + Chapter 1 of the book
- Read Appendix A, sections A1-A6 to get familiar with HDL
- Download the book's software suite
- Go through the hardware simulator tutorial
- You're in business.

Project 1

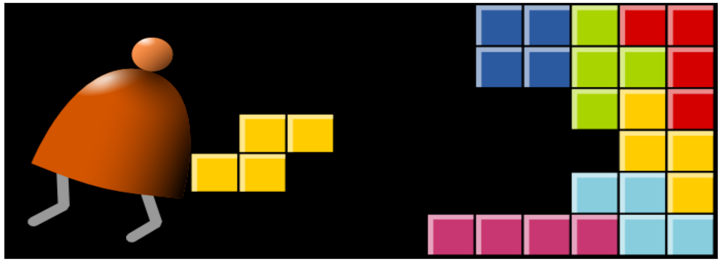
- Implement the logic gates presented in chapter 1.
- You may only use the primitive Nand gates and the composite gates that your team builds with them.
- Submit your working code and gate diagrams to Blackboard
  - If you need the extended time, submit the full project report

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