

Chapter 1

Boolean Logic

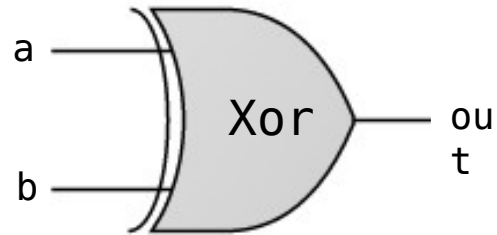
These slides support chapter 1 of the book

The Elements of Computing Systems

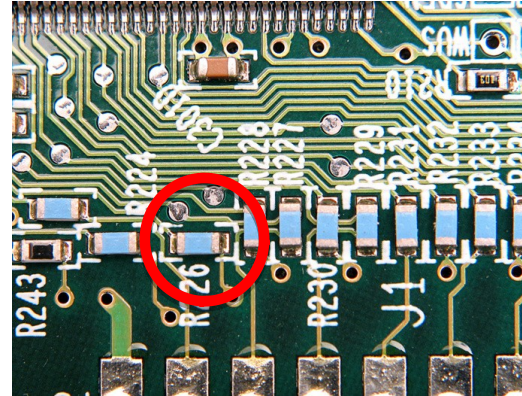
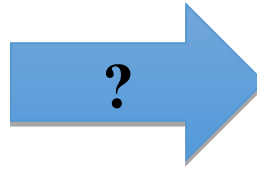
By Noam Nisan and Shimon Schocken

MIT Press

Building a logic gate



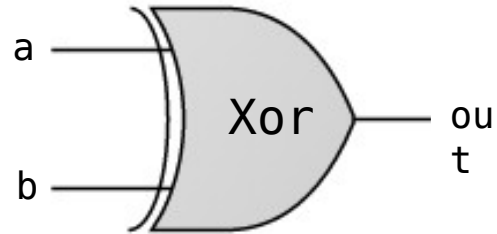
outputs 1 if one, and only one, of its inputs, is 1.



The Process:

- Design the gate architecture
- Specify the architecture in HDL
- Test the chip in a hardware simulator
- Optimize the design
- Realize the optimized design in silicon.

Design: from requirements to interface



outputs 1 if one, and only one, of its inputs, is 1.

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

Requirement:

Build a gate that delivers this functionality

```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b))  
*/
```

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

PARTS:

```
// Implementation missing
```

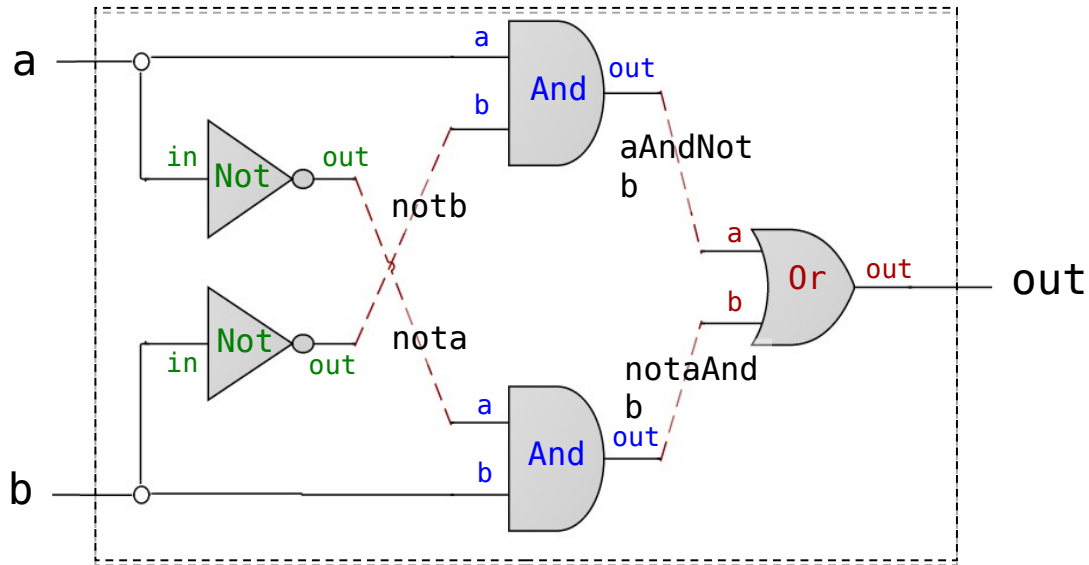
```
}
```

Gate interface

Expressed as an HDL stub file



Design: from gate diagram to HDL



```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b) */
```

interface

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

Other Xor
implementations
are possible!

implementation

```
  PARTS:
```

```
    Not (in=a, out=nota);
```

```
    Not (in=b, out=notb);
```

```
    And (a=a, b=notb, out=aAndNotb);
```

```
    And (a=nota, b=b, out=notaAndb);
```

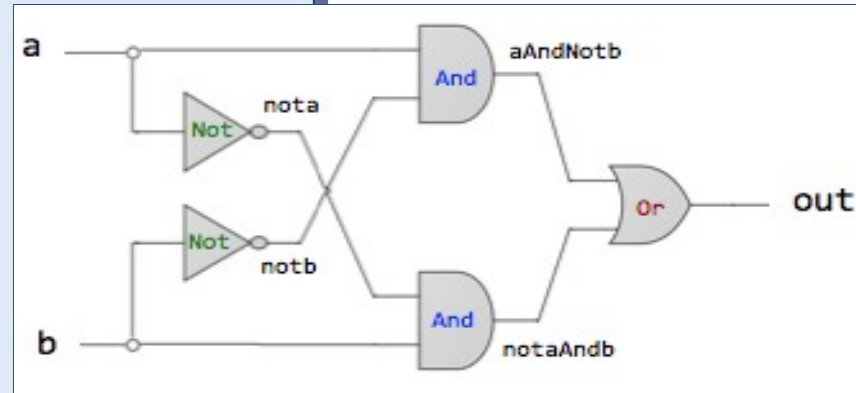
```
    Or (a=aAndNotb, b=notaAndb, out=out);
```

```
}
```

HDL: some comments

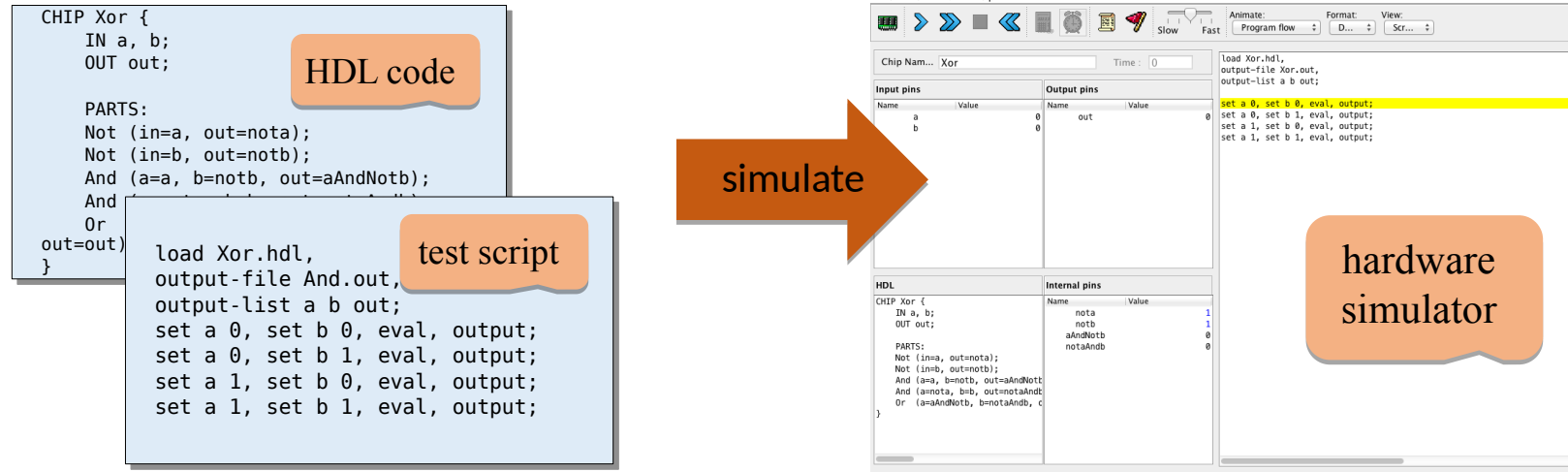
```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b) */
```

```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=aAndNotb);  
    And (a=nota, b=b, out=notaAndb);  
    Or (a=aAndNotb, b=notaAndb, out=out);  
}
```



- HDL is a functional / declarative language
- The order of HDL statements is insignificant
- Before using a chip part, you must know its interface. For example:
 Not(in= ,out=), And(a= ,b= ,out=), Or(a= ,b= ,out=)
- Connection patterns like chipName(**a=a**,...) and chipName(...,**out=out**)

Hardware simulation in a nutshell



Simulation options:

- Interactive
- Script-based
-

Interactive simulation

Hardware Simulator (2.5) - /Users/shimonschocken/Desktop/nand2tetris/week 1/Xor.hdl

File View Run Help

Chip Name: Xor

1. manipulate input pins

Input pins	
Name	Value
a	0
b	1

2. evaluate the chip logic

Output pins	
Name	Value
out	1

3. inspect output pins

```
HDL
CHIP Xor {
  IN a, b;
  OUT out;

  PARTS:
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=aAndNotb);
    And (a=nota, b=b, out=notaAndb);
    Or (a=aAndNotb, b=notaAndb, out=out);
}
```

HDL code

Internal pins	
Name	Value
nota	1
notb	0
aAndNotb	0
notaAndb	1

4. inspect internal pins

Interactive simulation



Script-based simulation

Xor.hdl

```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=aAndNotb);  
    And (a=nota, b=b, out=notaAndb);  
    Or (a=aAndNotb, b=notaAndb, out=out);  
}
```

tested chip

Xor.tst

```
load Xor.hdl;  
set a 0, set b 0, eval;  
set a 0, set b 1, eval;  
set a 1, set b 0, eval;  
set a 1, set b 1, eval;
```

test script = series of
commands to the simulator

Benefits:

- “Automatic” testing
- Replicable testing

Script-based simulation, with an output file

Xor.hdl

```
CHIP Xor {  
  IN a, b;  
  OUT out;  
  
  PARTS:  
    Not (in=a, out=nota);  
    Not (in=b, out=notb);  
    And (a=a, b=notb, out=aAndNotb);  
    And (a=nota, b=b, out=notaAndb);  
    Or (a=aAndNotb, b=notaAndb, out=out);  
}
```

tested chip

Xor.tst

```
load Xor.hdl,  
output-file Xor.out,  
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;
```

test
script

Xor.out

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

Output File, created
by the test script as a
side-effect of the
simulation process

The logic of a typical test script

- Initialize:
 - Load an HDL file
 - Create an empty output file
 - List the names of the pins whose values will be written to the output file
- Repeat:
 - set – eval - output

Hardware construction projects

- The players (first approximation):
 - System architects
 - Developers
- The system architect decides which chips are needed
- For each chip, the architect creates
 - A chip API
 - A test script
 - A compare file
- Given these resources, developers can build the chips.

Arrays of Bits

- Sometimes we wish to manipulate an array of bits as one group
- It's convenient to think about such a group of bits as a single entity, sometime termed “bus”
- HDLs usually provide notation and means for handling buses

Example: adding 16-bit integers

```
/*  
 * Adds two 16-bit inputs.  
 */  
CHIP Add16 {  
    IN a[16], b[16];  
    OUT out[16];  
  
    PARTS:  
    ...  
}
```



```
/*  
 * Adds three 16-bit inputs.  
 */  
CHIP Add3Way16 {
```

Working with **individual** bits within buses

```
/*  
 * 4-way And: Ands 4 bits.  
 */  
CHIP And4Way {
```

Working with **individual** bits *within buses*

```
/*  
 * Bit-wise And of two 4-bit inputs  
 */  
CHIP And4 {
```


Sub-buses

Buses can be composed from (and decomposed into) sub-buses

```
...  
IN lsb[8], msb[8], ...  
...
```

Sub-buses

Buses can be composed from (and decomposed into) sub-buses

```
...  
IN lsb[8], msb[8], ...  
...  
Add16(a[0..7]=lsb, a[8..15]=msb, b=..., out=...);  
Add16(..., out[0..3]=t1, out[4..15]=t2);
```

Some syntactic choices of our HDL

- buses are indexed right to left: if `foo` is a 16-bit bus, then `foo[0]` is the right-most bit, and `foo[15]` is the left-most bit
- overlaps of sub-buses are allowed in output buses of parts
- width of internal pin buses is deduced automatically
- The `false` and `true` constants may be used as buses of any width.

Chapter 1: Boolean logic



Boolean logic



Boolean function synthesis



Hardware description language



Hardware simulation



Multi-bit buses



Project 1 overview

Project 1

Given: Nand

Goal: Build the following gates:

Elementary logic gates

- Not
- And
- Or
- Xor
- Mux
- DMux

16-bit variants

- Not16
- And16
- Or16
- Mux16

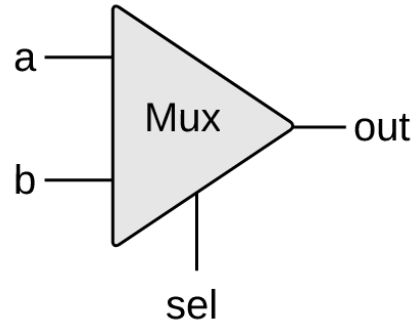
Multi-way variants

- Or8Way
- Mux4Way16
- Mux8Way16
- DMux4Way
- DMux8Way

Why these 15 particular gates?

- Commonly used gates
- Comprise all the elementary logic gates needed to build our computer.

Multiplexor



```
if (sel==0)
    out=a
else
    out=b
```

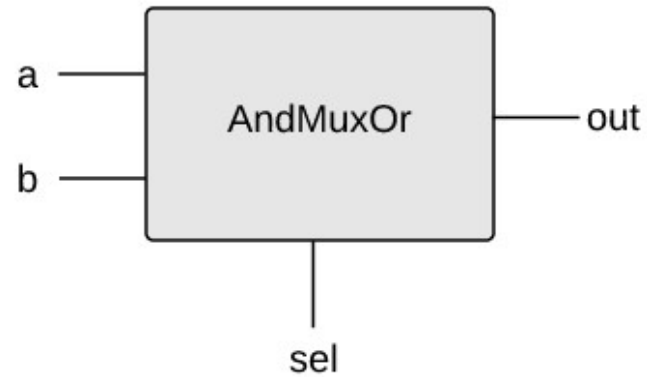
a	b	sel	out
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

sel	out
0	a
1	b

abbreviated
truth table

- A 2-way multiplexor enables selecting, and outputting, one of two possible inputs
- Widely used in:
 - Digital design
 - Communications networks

Example: using mux logic to build a programmable gate

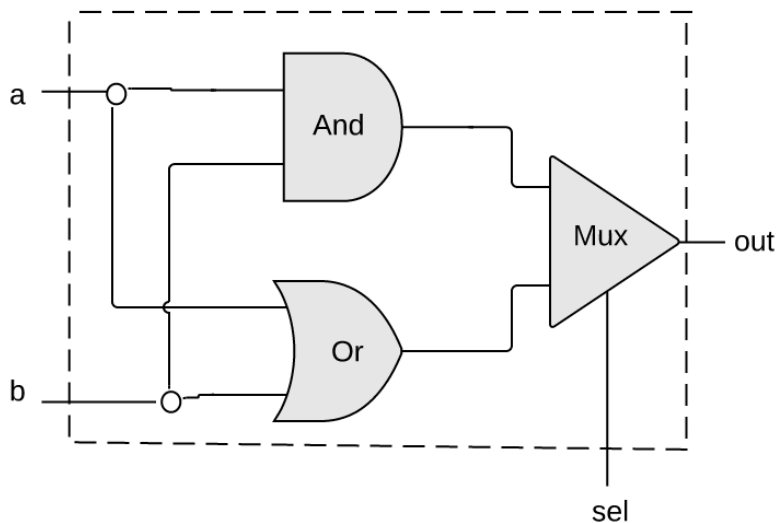


```
if (sel==0)
  out = (a And b)
else
  out = (a Or b)
```

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

When $sel==0$
the gate acts like
an And gate

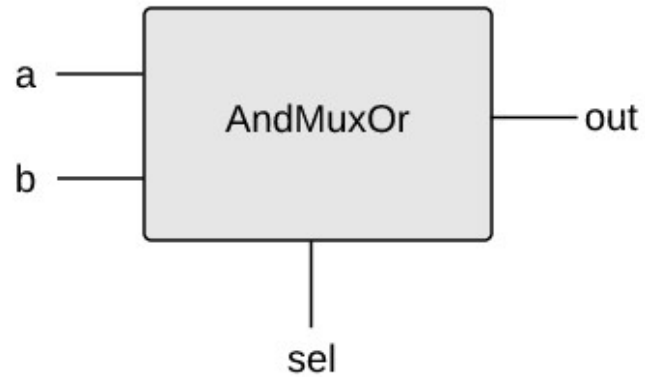
When $sel==1$
the gate acts like
an Or gate



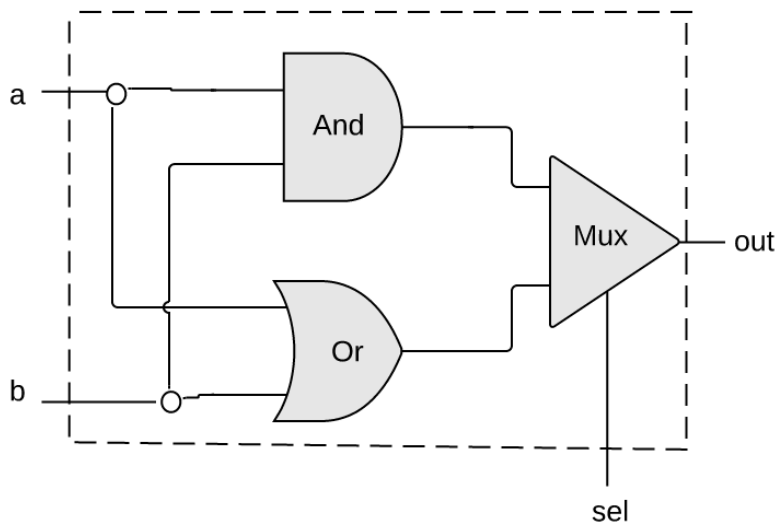
Mux.hdl

```
CHIP AndMuxOr {
  IN a, b, sel;
  OUT out;
```

Example: using mux logic to build a programmable gate



```
if (sel==0)
    out = (a And b)
else
    out = (a Or b)
```



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

When $sel==0$
the gate acts like
an And gate

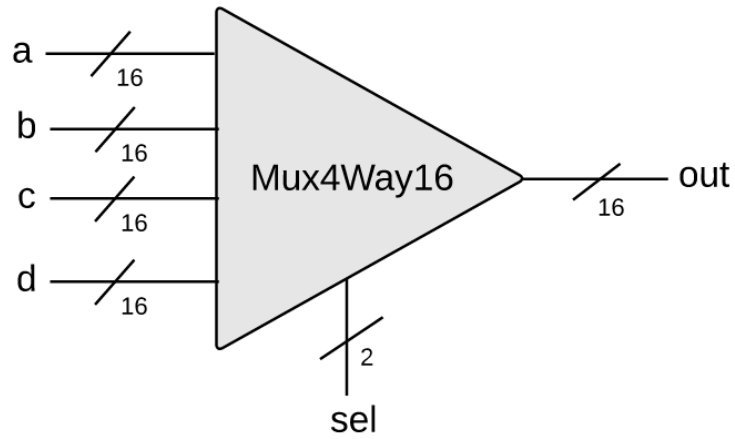
When $sel==1$
the gate acts like
an Or gate

Mux.hdl

```
CHIP AndMuxOr {
    IN a, b, sel;
    OUT out;

    PARTS:
        And (a=a, b=b, out=andOut);
        Or  (a=a, b=b, out=orOut);
        Mux (a=andOut, b=orOut, sel=sel,
            out=out);
}
```

16-bit, 4-way multiplexor



sel[1] sel[0]	out
0 0	a
0 1	b
1 0	c
1 1	d

Mux4Way16.hdl

```
CHIP Mux4Way16 {  
    IN a[16], b[16], c[16], d[16],  
        sel[2];  
    OUT out[16];  
  
    PARTS:  
    // Put your code here:  
}
```

Implementation tip:

Can be built from several Mux16 gates

Project 1 Resources

From NAND to Tetris
Building a Modern Computer From First Principles

www.nand2tetris.org



Home
Prerequisites
Syllabus

Course

Book
Software
Terms
Papers
Talks
Cool Stuff
About
Team
Q&A

Project 1: Elementary Logic Gates

Background

A typical computer architecture is based on a set of elementary logic gates like And, Or, Mux, etc., as well as their bit-wise versions And16, Or16, Mux16, etc. (assuming a 16-bit machine). This project engages you in the construction of a typical set of basic logic gates. These gates form the elementary building blocks from which more complex chips will be later constructed.

Objective

Build all the logic gates described in Chapter 1 (see list below), yielding a basic chip-set. The only building blocks that you can use in this project are primitive Nand gates and the composite gates that you will gradually build on top of them.


Chips

Chip (HDL)	Description	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
DMux	DMux gate	DMux.tst	DMux.cmp
Not16	16-bit Not	Not16.tst	Not16.cmp
And16	16-bit And	And16.tst	And16.cmp
Or16	16-bit Or	Or16.tst	Or16.cmp
Mux16	16-bit multiplexor	Mux16.tst	Mux16.cmp

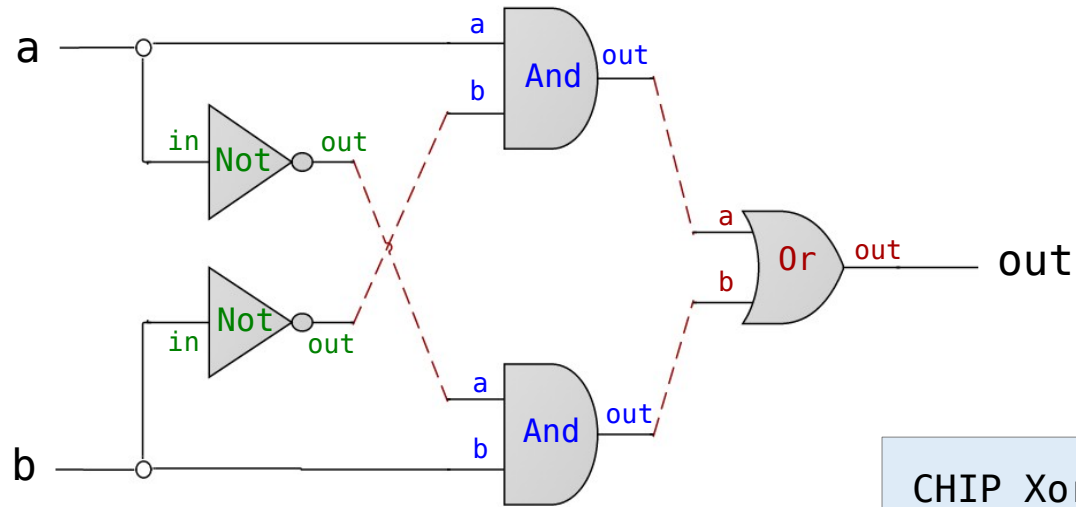
All the necessary project 1 files
are available in:
[nand2tetris/projects/](#)

01

More resources

- Text editor (for writing your HDL files)
 - HDL Survival Guide
 - Hardware Simulator Tutorial
 - nand2tetris Q&A forum
- 
- All available in: www.nand2tetris.org

Hack chipset API



When deciding to use some chip-parts, how do I know the names of their input and output pins?

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

PARTS:

```
Not (in= , out=);
```

```
Not (in= , out=);
```

```
And (a= , b= , out=);
```

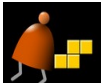
```
And (a= , b=b , out=);
```

```
Or (a= , b= , out=);
```

```
}
```


Hack chipset API

```
Add16 (a= ,b= ,out= );
ALU (x= ,y= ,zx= ,nx= ,zy= ,ny= ,f= ,no= ,out= ,zr= ,ng= );
And16 (a= ,b= ,out= );
And (a= ,b= ,out= );
Aregister (in= ,load= ,out= );
Bit (in= ,load= ,out= );
CPU
(inM= ,instruction= ,reset= ,outM=
DFF (in= ,out= );
DMux4Way (in= ,sel= ,a= ,b= ,c= ,
DMux8Way (in= ,sel= ,a= ,b= ,c= ,
Dmux (in= ,sel= ,a= ,b= );
Dregister (in= ,load= ,out= );
FullAdder (a= ,b= ,c= ,sum= ,carr
HalfAdder (a= ,b= ,sum= , carry=
Incl6 (in= ,out= );
Keyboard (out= );
Memory (in= ,load= ,address= ,out
Mux16 (a= ,b= ,sel= ,out= );
Mux4Way16 (a= ,b= ,c= ,d= ,sel= ,
Mux8Way16 (a= ,b= ,c= ,d= ,e= ,f=
Mux8Way (a= ,b= ,c= ,d= ,e= ,f= ,g= ,h= ,sel= ,out=
);
Mux (a= ,b= ,sel= ,out= );
Nand (a= ,b= ,out= );
Not16 (in= ,out= );
Not (in= ,out= );
Or16 (a= ,b= ,out= );
Or8Way (in= ,out= );
Or (a= ,b= ,out= );
PC (in= ,load= ,inc= ,reset= ,out= );
PCLoadLogic (cinstr= ,j1= ,j2= ,j3= ,load= ,inc= );
RAM16K (in= ,load= ,address= ,out= );
RAM4K (in= ,load= ,address= ,out= );
RAM512 (in= ,load= ,address= ,out= );
RAM64 (in= ,load= ,address= ,out= );
RAM8 (in= ,load= ,address= ,out= );
Register (in= ,load= ,out= );
ROM32K (address= ,out= );
Screen (in= ,load= ,address= ,out= );
Xor (a= ,b= ,out= );
```




(see *HDL Survival Guide* @ www.nand2tetris.org)

Built-in chips

```
CHIP Foo {  
    IN ...;  
    OUT ...;  
  
    PARTS:  
    ...  
    Mux16(...)   
    ...  
}
```

Q: What happens if there is no Mux16.hdl file in the current directory?

Built-in chips

```
CHIP Foo {  
    IN ...;  
    OUT ...;  
  
    PARTS:  
    ...  
    Mux16(...)   
    ...  
}
```

Q: What happens if there is no `Mux16.hdl` file in the current directory?

A: The simulator invokes, and evaluates, the built-in version of `Mux16` (if such exists).

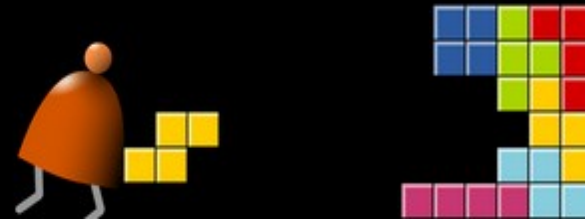
- The supplied simulator software features built-in chip implementations of all the chips in the Hack chip set
- If you don't implement some chips from the Hack chipset, you can still use them as chip-parts of other chips:
 - Just rename their given stub files to, say, `Mux16.hdl1`
 - This will cause the simulator to use the built-in chip implementation.

Best practice advice

- **Try to implement the chips in the given order**
- If you don't implement some chips, you can still use them as chip-parts in other chips (the built-in implementations will kick in)
- You can invent new, “helper chips”; however, this is not required: you can build any chip using previously-built chips only
- Strive to use as few chip-parts as possible.

Chapter 1: Boolean logic

- ✓ Boolean logic
- ✓ Boolean function synthesis
- ✓ Hardware description language
- ✓ Hardware simulation
- ✓ Multi-bit buses
- ✓ Project 1 overview



Chapter 1

Boolean Logic

These slides support chapter 1 of the book

The Elements of Computing Systems

By Noam Nisan and Shimon Schocken

MIT Press