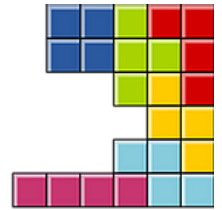
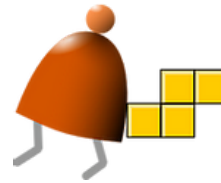


# From Nand to Tetris

## Building a Modern Computer From First Principles



Home
Projects
Book
Software
License
Papers
Cool Stuff
About
Team
Stay in Touch
Q&A

## Project 1: Boolean Logic

### 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 Name		Description	Test Scripts		Compare File	
Nand	<a href="#">Chip Name</a>	Nand gate (primitive)	<a href="#">Description</a>	<a href="#">Test Scripts</a>	<a href="#">Compare File</a>	
Not	<a href="#">Chip Name</a>	Not gate	<a href="#">Description</a>	<a href="#">Not.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Not.cmp</a> <a href="#">Compare File</a>
And	<a href="#">Chip Name</a>	And gate	<a href="#">Description</a>	<a href="#">And.tst</a>	<a href="#">Test Scripts</a>	<a href="#">And.cmp</a> <a href="#">Compare File</a>
Or	<a href="#">Chip Name</a>	Or gate	<a href="#">Description</a>	<a href="#">Or.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Or.cmp</a> <a href="#">Compare File</a>
Xor	<a href="#">Chip Name</a>	Xor gate	<a href="#">Description</a>	<a href="#">Xor.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Xor.cmp</a> <a href="#">Compare File</a>
Mux	<a href="#">Chip Name</a>	Mux gate	<a href="#">Description</a>	<a href="#">Mux.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Mux.cmp</a> <a href="#">Compare File</a>
DMux	<a href="#">Chip Name</a>	DMux gate	<a href="#">Description</a>	<a href="#">DMux.tst</a>	<a href="#">Test Scripts</a>	<a href="#">DMux.cmp</a> <a href="#">Compare File</a>
Not16	<a href="#">Chip Name</a>	16-bit Not	<a href="#">Description</a>	<a href="#">Not16.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Not16.cmp</a> <a href="#">Compare File</a>
And16	<a href="#">Chip Name</a>	16-bit And	<a href="#">Description</a>	<a href="#">And16.tst</a>	<a href="#">Test Scripts</a>	<a href="#">And16.cmp</a> <a href="#">Compare File</a>
Or16	<a href="#">Chip Name</a>	16-bit Or	<a href="#">Description</a>	<a href="#">Or16.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Or16.cmp</a> <a href="#">Compare File</a>
Mux16	<a href="#">Chip Name</a>	16-bit multiplexor	<a href="#">Description</a>	<a href="#">Mux16.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Mux16.cmp</a> <a href="#">Compare File</a>
Or8Way	<a href="#">Chip Name</a>	Or(in0,in1,...,in7)	<a href="#">Description</a>	<a href="#">Or8Way.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Or8Way.cmp</a> <a href="#">Compare File</a>
Mux4Way16	<a href="#">Chip Name</a>	16-bit/4-way mux	<a href="#">Description</a>	<a href="#">Mux4Way16.tst</a>	<a href="#">Test Scripts</a>	<a href="#">Mux4Way16.cmp</a> <a href="#">Compare File</a>

Mux8Way16	Chip Name	16-bit/8-way mux	Description	Mux8Way16.tst	Test Scripts	Mux8Way16.cmp	Compare File
DMux4Way	Chip Name	4-way demultiplexor	Description	DMux4Way.tst	Test Scripts	DMux4Way.cmp	Compare File
DMux8Way	Chip Name	8-way demultiplexor	Description	DMux8Way.tst	Test Scripts	DMux8Way.cmp	Compare File

## Contract

When loaded into the supplied Hardware Simulator, your chip design (modified .hdl program), tested on the supplied .tst script, should produce the outputs listed in the supplied .cmp file. If that is not the case, the simulator will let you know. This contract must be satisfied for each chip listed above, except for the Nand chip, which is considered primitive, and thus there is no need to implement it.

## Resources

See [Chapter 1](#), the [HDL Guide](#) (except for A2.4), and the [Hack Chip Set](#).

For each chip, we supply a skeletal .hdl file with a place holder for a missing implementation part. In addition, for each chip we supply a .tst script that instructs the hardware simulator how to test it, and a .cmp ("compare file") containing the correct output that this test should generate. Your job is to complete and test the supplied skeletal .hdl files.

If you've downloaded the Nand2Tetris Software Suite (from the Software section of this website), you will find the supplied hardware simulator and all the necessary project files in the nand2tetris/tools folder and in the nand2tetris/projects/01 folder, respectively. To get acquainted with the hardware simulator, see the Hardware Simulator Tutorial ([PPT](#), [PDF](#))

## Tips

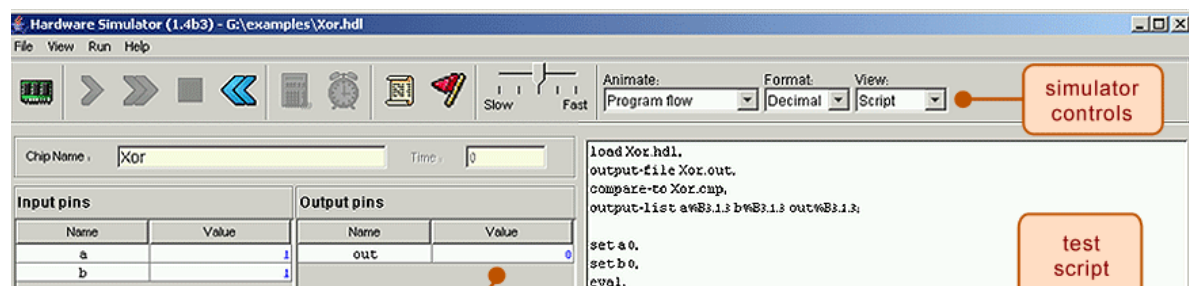
**Prerequisite:** If you haven't done it yet, download the Nand2Tetris Software Suite from the Software section of this website to your computer. Read Chapter 1 and Appendix 2 (not including A2.4), and go through parts I-II-III of the Hardware Simulator, before starting to work on this project.

**Built-in chips:** The Nand gate is considered primitive and thus there is no need to implement it: whenever a Nand chip-part is encountered in your HDL code, the simulator automatically invokes the built-in tools/builtInChips/Nand.hdl implementation. We recommend implementing all the other gates in this project in the order in which they appear in Chapter 1. However, note that the supplied hardware simulator features built-in implementations of all these chips. Therefore, you can use any one of these chips before implementing it: the simulator will automatically invoke their built-in versions.

For example, consider the supplied skeletal Mux.hdl program. Suppose that for one reason or another you did not complete the implementation of Mux, but you still want to use Mux chips as internal parts in other chip designs. You can easily do so, thanks to the following convention. If the simulator fails to find a Mux.hdl file in the current directory, it automatically invokes the built-in Mux implementation, which is part of the supplied simulator's environment. This built-in Mux implementation has the same interface and functionality as those of the Mux chip described in the book. Thus, if you want the simulator to ignore one or more of your chip implementations, rename the corresponding chipName.hdl file, or remove it from the directory. When you are ready to develop this chip in HDL, put the file chipName.hdl back in the directory, and proceed to edit it with your HDL code.

## Tools

All the chips mentioned projects 1-5 can be implemented and tested using the supplied hardware simulator. Here is a screen shot of testing a Xor.hdl chip implementation on the Hardware Simulator:



The screenshot displays a Verilog HDL simulation environment. It is divided into several sections:

- HDL program:** Contains the Verilog code for an Exclusive-or gate. The code includes a comment, a module definition, input/output declarations, and a parts section with logic gates.
- Internal pins:** A table showing the current values of internal signals.
- Simulation steps:** A list of commands like 'set a 0', 'set b 1', 'eval', and 'output:'.
- Output file:** A Notepad window showing the simulation results in a table.

Annotations with orange boxes and arrows point to these sections:

- HDL program** points to the Verilog code.
- current pin values** points to the 'Internal pins' table.
- typical simulation step** points to a group of simulation commands.
- output file** points to the 'Xor.out - Notepad' window.

**HDL code:**

```
// Exclusive-or gate. out = a Xor 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=w1);  
    And (a=nota, b=b, out=w2);  
    Or (a=w1, b=w2, out=out);  
}
```

**Internal pins table:**

Name	Value
nota	0
notb	0
w1	0
w2	0

**Simulation steps:**

```
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 (Xor.out - Notepad):**

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

End of script - Comparison ended successfully