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	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
Or8Way	Or(in0,in1,...,in7)	Or8Way.tst	Or8Way.cmp
Mux4Way16	16-bit/4-way mux	Mux4Way16.tst	Mux4Way16.cmp
Mux8Way16	16-bit/8-way mux	Mux8Way16.tst	Mux8Way16.cmp
DMux4Way	4-way demultiplexor	DMux4Way.tst	DMux4Way.cmp
DMux8Way	8-way demultiplexor	DMux8Way.tst	DMux8Way.cmp

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

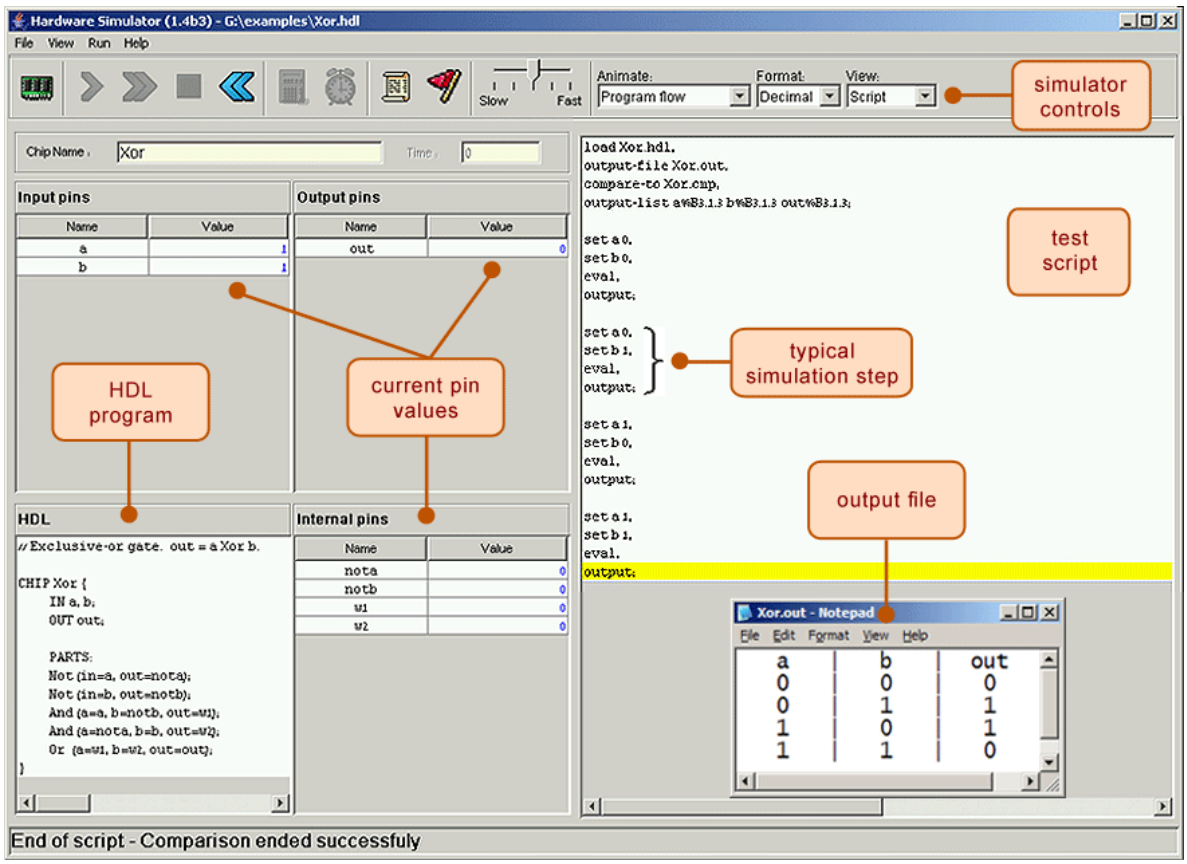
Prerequisite: If you haven't done it yet, download the Nand2Tetris Software Suite from the Software section of this website to your computer. Read Chapter 1 and Appendix 2 (not including A2.4), and go through parts I-II-III of the Hardware Simulator, before starting to work on this project.

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:


Hardware Simulator (1.4b3) - G:\examples\Xor.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Decimal
View:
Script
simulator controls
Chip Name:
Xor
Time:
0
Input pins
Name
Value
a
1
b
1
Output pins
Name
Value
out
0
HDL program
current pin values
test script
typical simulation step
output file
HDL
// 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
Name
Value
nota
0
notb
0
w1
0
w2
0
load Xor.hdl,
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;
Xor.out - Notepad
File Edit Format View Help
a b out
0 0 0
0 1 1
1 0 1
1 1 0
End of script - Comparison ended successfuly