HDL Survival Guide

by Mark Armbrust

This guide is designed to help you understand and write HDL programs in the context of Nand2Tetris courses. It was written in order to answer recurring issues that came up in many posts in the Q&A forum of the nand2tetris web site. These posts were made by people who worked on the course's hardware projects and got stuck for one reason or another.

Terminology

Throughout this document, we use the terms "HDL file", "HDL program", and "HDL implementation" interchangeably. Likewise, we use the terms "build a chip", "construct a chip", and "implement a chip" interchangeably. The term "HDL file stub" refers to a file that contains the HDL definition of a chip interface. That is, a stub file contains the chip name and the names of all the chip's input and output pins, without the chip's implementation, also known as the "PARTS" section of the HDL program. A stub file also contains the chip's documentation -- a succinct description of what the chip is supposed to do.

Project Files

If you've downloaded the nand2tetris software, then there are now 13 directories (folders) on your computer, named `projects/01` ..., `projects/13`. Each directory contains all the files necessary to complete the respective project. The projects themselves are described in the **Course** page of the nand2tetris web site. Projects 1-5 focus on building the hardware platform of the Hack computer. Each project walks you through the construction of a certain subset of the Hack chip-set. The directory (folder) that accompanies each project contains stub HDL files for all the chips you need to build, as well as all the test scripts required to test your HDL implementations .The "only" thing that you have to do is extend the supplied stub HDL files into working chip implementations, i.e. chips that run in the supplied hardware simulator and successfully pass all the tests listed in the supplied test scripts.

Other useful resources to help you complete the projects are the lecture slides given in the "Lecture" column of the **Course** page, as well as the relevant book chapters.

Chip Implementation Order

Each project directory contains a set of HDL stub files, one for each chip that you have to build. It's important to understand that all the supplied HDL files contain no implementations (building these implementations is what the project is all about). For example, consider the construction of a Xor chip. If your `Xor.hdl` program will include, say, And and Or chip-parts, and you have not yet implemented the And and Or chips, your tests will fail even if your Xor implementation is correct.

Note however that if the project directory included no `And.hdl` and `Or.hdl` files at all, your `Xor.hdl` program will work properly. This sounds surprising, so here is the explanation. The hardware simulator, which is a Java program, features working Java implementations of all the chips necessary to build the Hack computer. When the simulator has to execute the logic of some chip-part, say And, taken from the Hack chip-set, it looks for an `And.hdl` file in the current project directory. At this point there are three possibilities:

- No HDL file is found. In this case, the Java implementation of the chip kicks in and "covers" for the

missing HDL implementation.
- A stub HDL file is found. The simulator tries to execute it. But since a stub file contains no implementation, the execution fails.
- A "normal" HDL file is found. The simulator executes it, reporting errors to the best of its ability.

Therefore, to avoid chip order implementation troubles, you can do one of two things. First, you can implement your chips in the order presented in the book and in the project descriptions. Since the chips are presented "bottom-up", from basic chips to more complex ones, you will encounter no chip order implementation troubles.

A recommended alternative is to create a subdirectory called "stubs" and move all the supplied HDL stub files into that directory. You can then move them into your working directory as needed.

Note that the `.hdl` file that you are working on and its associated `.tst` file must be in the same directory. If you will look at the header code of the supplied test script, you will see that it includes a command that loads the HDL file that it is supposed to test from the same working directory. Thus, if you load your HDL file into the simulator and then load the test script from a different directory, the test script will load and test a different HDL file. Proper usage is to load into the simulator either an `.hdl` file or a `.tst` file, not both.

HDL syntax and the meaning of `"a=a"`

HDL statements are used to describe how chip-parts are connected to each other, as well as to the input and output pins of the constructed chip. The syntax of these statements can be confusing, especially when pin names like `in`, `out`, `a` and `b` are used both by the chip-parts as well as by the constructed chip. This leads to statements like `And(a=a, b=b, out=r)`. This particular statement instructs to (i) connect the `a` and `b` input pins of the And chip-part to the `a` and `b` input pins of the constructed chip, (ii) create an internal pin ("wire") named `r`, and (iii) connect the output pin of the And chip-part to `r`.

Here is a simple rule that helps sort things out: the symbol on the left hand side of each "=" symbol is always the name of a pin in a chip-part, and the symbol on the right hand side is always the name of a pin or a wire in the constructed chip (the chip that you are building).

The following figure shows a diagram and HDL code of a Xor chip. The diagram uses color coding to highlight which symbols "belong" to which chips. (Note that there are several different ways to implement Xor; this is just one of them.)