This manual was extracted directly from http://www.ohwr.org/projects/hdl-make/wiki, the original project on which this one was based.

Quick start¶

It is not really needed to read the whole documentation. If you just want to use it, go to **Quick start** tutorial.

A description of a sample project can be found **here**.

Introduction to Hdlmake

Hdlmake generates multi-purpose makefiles for FPGA projects. It enables local and remote synthesis, fetching modules' dependencies from repositories, creating Quartus/ISE project files. All of this can be done with a makefile command or with Hdlmake directly. It supports modularity, scalability, use of revision control systems and code reuse. Hdlmake is free, open and distributed with GPL. All you need to download is a single file (hdlmake).

Rationale

Maintaining VHDL projects either for Altera or for Xilinx is a source of many problems. When using Modelsim for simulating VHDL designs there is no tool for dependency generation. The developer has to manually generate the dependencies between files and modules, as well as compilation order. This is laborious and consumes developer's time.

Synthesis of large projects is a time and resource-consuming process. It makes the edit-compile-test cycle unreasonably long and makes it harder to introduce trivial modifications to the hardware. Apart from this, much of the system resources like memory or CPU time are consumed and synthesis makes running other application harder. The solution for this problem is to set up a dedicated server solely for the purpose of synthesis. This allows to delegate this demanding task to a fast machine, while the developer's computer remains still useful.

The aim of the project is to offer multiple functionalities whose aim is to make the VHDL coder's life easier. Nevertheless, its principal goal is to provide means for performing simulation and synthesis for VHDL projects.

Python

The program is written in Python and is checked under Python 2.7. There were some issues with compatibility in Python 3.1 but they have been resolved. The best efforts have been made to ensure compatibility between versions but there is no guarantee of correct behavior in the newest versions of Python.

Required environment

Hdlmake is basically written for use in the Linux environment. It developed under Ubuntu 10.04 and all features are tested on *nix systems. Nevertheless, I was informed by Adreas Bergmann (thanks!) that there is no mistery in running Hdlmake under Windows. All you have to do is to install Cygwin, zip (Cygwin version) and add path to FPGA tools to the path environment variable.

For most of the options, Python is the only thing that is required to make it work.
In order to perform remote synthesis you should equip your system with a ssh client and rsync.

Way of use

The basis for Hdlmake are configuration files called ``manifests". Those files have form of Python scripts and therefore must obey its syntax. They should also have Python extension (.py). As Hdlmake supports synthesis and simulation, they are used for both testbenches and top modules. The major goal of the manifests is to describe the structure of the project:

- for parent modules; what modules it is made of? where to take them from? where to store them locally?
- what are the local files, that should be used in synthesis or simulation,
- what is the target library for a module (in a VHDL sense),
- what is the name of the project,
- which ISE version should be used,
- which qsh (Quartus Shell) version should be used, etc.

Whenever you want to run Hdlmake for any reason, you should prepare a manifest.py file containing all the needed variables. Description of available variables can be found in section \ref{subsec:vars}. For a quick overview you can also use the built-in user help.
Hdlmake makes use also of run arguments. In general, their purpose is to indicate the action that should be taken by the script. Some of them are also used for specifying additional parameters that are not enclosed in manifests. All supported actions are listed in an appropriate section.

Features

Hdlmake consists of two major components which are described below.

File-based project description

Hdlmake defines a way to describe projects that consist of smaller modules. Many times developers keep these modules together in a single repository, even though it is cleaner to store them separately. When some of these modules are reused across several projects, it can cause a mess. Hdlmake offers easy and convenient way to specify where to find necessary modules. There is no need to store them permanently on the local hard disk.
It is of course up to a developer how a project is divided into smaller parts. All files can be stored in one folder and kept in one repository, but this has been found to be a bit inconvenient. You should use a structure that you feels comfortable with.

Helper script for project-related operations

When your project is properly described with manifest files, then you can take advantage of written scripts. You can write makefiles for simulation and do synthesis on other machines over a network. You can also automatically fetch the needed modules recursively and and get rid of them whenever you want.

Basic run scenarios

In order to use Hdlmake developer has to put all python files together in one directory. hdlmake must have execution rights. Next it is necessary to change the current directory to this one that contains a manifest. From there hdlmake must be run with one or more arguments.

Makefile preparation (no option)

Makefile preparation is one of the most basic features that Hdlmake can be used for. It relieves a developer of creating makefiles by hand.
When compiling VHDL files for simulation, one must ensure their correct order. In every project there dependecies between

all project files, that can be expressed in a form of dependency tree - one file makes use of data from an other file (by e.g. including it). A file may not be compiled until all files it is dependent on are compiled, otherwise the compilation process will fail.

In order to use Hdlmake, it must be run from the directory containing a manifest. It is assumed that all needed modules were previously fetched. When run, Hdlmake starts with reading the top module and recursively collecting modules that are parts of the current design. You can specify modules that are stored locally, in a git or an SVN repository.

When the list of used modules is ready, Hdlmake checks if any of those modules has in its manifest a list of files that should be used in the testbench. If not, all of the module's files are added to the list of files. In the next step, hdlmake scans all of the listed files and tries to figure out dependencies between them by analyzing each file's content. They are next written down as a makefile in the testbench's directory.

Fetching submodules for a top module (option -f or make fetch)

Hdlmake offers a short way of describing project structures. It is assumed that a projects can consist of modules, that are stored in different places (locally or in a repository). Each of these modules can be based on other modules.
Hdlmake can fetch all of them and store them in specified places.
For each module one can specify a target catalog with manifest variable fetchto. Its value must be a name (existing or not) of a folder. The folder may be located anywhere in the filesystem. It must be though a relative path (Hdlmake supports only relative paths).

Synthesizing projects remotely (option -r or make remote)

Another valuable feature of Hdlmake is the ability to perform VHDL synthesis on a remote machine instead of running it on your desktop. Both Altera's qsh and Xilinx' ISE-command-line-tools are suppored. In comparison with local synthesis, remote synthesis has several adventages:

- Developer can run synthesis on a fast machine, instead on his slow machine
- Developer is not forced to install new software in case his computer is not equipped with a particular version of the software
- Synthesis is a resource-consuming process. When someone runs it on a remote machine, the development machine remains usable
- A common synthesis server allows to unify synthesis projects. Usually, when developing a project in a team, all team members have different version of software tools. These tools are complex and their versions are not always fully compatible with each other. When collaborating developers share the same machine for synthesis they can always be sure that the synthesis will run correctly.

Synthesis server and username for logging in on the synthesis server can be specified with run the parameters:--synth-server and --synth-user accordingly or can be edited manually in the makefile. If run options are given, the makefile uses environmental variables HDLMAKE_USER and HDLMAKE_SERVER. If they are not set in the environment, an error is raised.

Synthesizing projects locally (option -l or make local)

It is also possible to perform synthesis on the local machine. For this purpose server and username are not necessary.

Additional features

Verbose mode (option -v)

For debugging purpose and having a more detailed view over what the program is doing you can run Hdlmake in verbose mode. In this mode most of the intermediate data structures are printed

Manifest variables description

The description can be found on the **manifest variables description page**

Run arguments description

The description can be found on the **run arguments summary page**

Examples

If you didn't do so, check out the **Sample project** description.