A Deeper Look at GHDL

Yehowshua Immanuel

January 17, 2020

Introduction

GDHL is a Free and Open Source(FOSS) VHDL simulator and synthesizer(for FPGAs and VLSI CAD etc.) frontend. I help maintain the GHDL port for Macs, so let me know if you have any trouble.

GHDL works in primarily in two steps, analysis and elaboration which are described below. In addition, the pure VHDL specification is rather bare, so it is necessary to pass libraries flags such as `ieee=synopsys` to let GHDL know you want to use constructs such as `std_logic`.

Design Analysis

This usually involve something along the lines of `$ghdl -a -fexplicit --ieee=synopsys *.vhdl` where `*` is the bash shell globbing operator. `*.vhdl` expands to all the VHDL files in a directory. You can replace `*.vhdl` with the files you explicitly wish ghdl to analyze.

Analysis basically converts the VHDL into a cycle accurate ADA model. These models are then built into an object file per VHDL file.

Design Elaboration

Elaborating with GHDL usually takes the form of `ghdl -e -fexplicit --ieee=synopsys top_entity_name`. Elaboration resolves the path to all the included standards and links all the generated object files together. The result of elaboration is an executable binary that simulates the VHDL.

Soft intro to Makefiles

Correctly maintaining dependencies between the design analysis and elaboration stages can get a little hairy. Fortunately, GHDL provides an internal mechanism for keeping these dependencies straight. This particular mechanism makes use of a software call `make`.

Software is often built in stages. Suppose you had five files that went through the **compile** $\rightarrow$ **object** stage and **object** $\rightarrow$ **linked-executable stage**. It is possible to write a single monolithic script that rebuilds every file at stage every time you wish to rebuild the final executable. But if you only changed one file, this is an inefficient way to rebuild the executable. You should only rebuild the objects that depend on the files you changed, and then relink the new object and old objects together to build the executable. Makefiles allow you to build a dependency graph via rules that state the dependencies of generated files at various stages.

Makefiles are executed with `$make` which consumes a makefile(usually named `makefile` or `Makefile` with no file extension), generates, and finally executes the dependency graph, only executing graph nodes with children that are younger than the parents.

Using the Provisioned Makefile

The makefile provided for this assignment should work for simulating most of the VHDL designs you may encounter both within and without this class. In the future, you will probably only need to change the `$SRC` and `$TOP_ENTITY` variables in the makefile which are currently set to:

```
SRC = MIPS-SingleCycle.vhdl MIPS-Assignment-1tb.vhdl
TOP_ENTITY = MIPS_tb
```

The provisioned makefile makes good use of GHDL's internal makefile generation capabilities, and adds the auxiliary abilities of both `make clean` and exporting both the waveforms and object files into a single directory.

Additional Resources

- GHDL Help
- GTKWave
- Xming Download
- VHDL Help