Hog (HDL on git): a collaborative HDL management tool

Francesco Gonnella[1]

School of Physics and Astronomy, University of Birmingham, Edgbaston B15 2TT, Birmingham, UK

E-mail: `francesco.gonnella@cern.ch`

Abstract. Coordinating firmware development among many international collaborators is becoming a very widespread problem in high-energy physics. Guaranteeing firmware synthesis reproducibility and assuring traceability of binary files is paramount. We devised Hog (HDL on git), a set of Tcl scripts (no external tool or library is needed) that tackles these issues and is deeply integrated with FPGA IDEs (Xilinx Vivado Design Suite/ISE PlanAhead, Intel Quartus Prime). Hog assures absolute control of HDL source files, constraint files, Vivado/Quartus settings and guarantees traceability by automatically embedding the git commit SHA and a numeric version into the binary file which is automatically renamed to reflect its version. Hog allows the IDE GUI to be used normally, so developers can get quickly up to speed: clone the repository, run the Hog script, and work in your IDE. Hog works on Windows and Linux, supports IPbus, Sigasi and provides pre-made yml files to set up a working CI on Gitlab with no additional effort.

1. Introduction

Hog (HDL on git) [1, 2] scripts and Hog's methodology aim to simplify HDL project maintenance and versioning with git [3].

HDL IDEs (Integrated Development Environments), such as Xilinx Vivado[4] or ISE[5] (PlanAhead[6]), and Intel Quartus[7], can be used normally through their graphical interfaces. Hog can also be used entirely from the command line, leaving the choice to the individual developer.

No additional software is required aside from what is needed for HDL synthesis/implementation and simulation.

Hog maintains absolute control of source files, constraints, and IDE settings to assure reproducibility. Before starting the workflow, Hog automatically checks if any modifications were made to the project (added/removed files, modified files, project properties, etc.) with respect to what is committed to the Git repository. If that is the case it issues a Critical Warning and marks the binary file as "dirty".

Otherwise, Hog automatically embeds the version number and the git SHA[2] into the binary file to assure traceability. Files which were built as dirty are also named accordingly with a "-dirty" suffix.

[1] On behalf of the Hog team: N.V. Biesuz, A. Camplani, D. Cieri, N. Giangiacomi, F. Gonnella, A. Peck
[2] Secure Hash Algorithm, a checksum identifier that git uses to unambiguously identify a commit in a repository.

Hog is used in several high-energy physics projects inside and outside CERN, such as the ATLAS and CMS Phase-I and Phase-II upgrades, EMPHATIC, GAPS, and FOOT.

Being fully integrated in the IDE, Hog is designed to work with any Vivado/PlanAhead/Quartus version. However, it was tested on PlanAhead 14.7, Vivado 2016.4, 2017.1, 2017.3, 2018.1, 2018.2, 2018.3, 2019.1, 2019.2, 2020.1, 2020.2 and Quartus 19.1, 19.2.

Hog is described in detail in [2]. The next sections describe the main improvements that were made in version 2021.2 and the plans for future releases.

2. New features of Hog2021.2

2.1. Hog configuration file

In Hog previous releases, a Tcl script with the same name of the project was run to create each project. That was located in 'Top/⟨project⟩/' and named '⟨project⟩.tcl'.

This project file contained all the Vivado/Quartus/ISE settings and in the last lines sourced a Hog script called 'create_project.tcl' which would actually perform all the project creation tasks: read the list files, create the project, add the sources files, etc.

From Hog2021.2 another option is available: a configuration file called 'hog.conf'. This file follows the '.conf' (or '.ini' for Windows users) syntax and it is more readable and maintainable than a Tcl script. It is located in 'Top/⟨project⟩/' and it is always called 'hog.conf' independent of the project name.

The first line of the 'hog.conf' file is a comment (staring with #) that specifies what IDE is used for the project: Vivado, Quartus, PlanAhead, etc.

The simplest Hog project configuration file only contains the PART parameter specifying what FPGA is to be used in the project.

```
#vivado
[main]
PART = xc7a35tcpg236-1
```

In general this file may contains several sections for synthesis, implementation, and other kind of properties:

```
#vivado
[main]
PART = xc7vx550tffg1927-2

[synth_1]
STEPS.SYNTH_DESIGN.ARGS.RETIMING = true

[impl_1]
steps.write_bitstream.args.bin_file = 1
STEPS.OPT_DESIGN.ARGS.DIRECTIVE = Default
STRATEGY = "Performance_Retiming"
```

The idea behind this new feature is to reduce the overhead for developers by eliminating the need to learn Tcl syntax. To keep the build system flexible and powerful, project-specific pre-creation and post-creation Tcl scripts can still be used, allowing users to perform any additional tasks required at the time of project creation.

2.2. Sub directories in the Top directory

In case a repository contains numerous projects, it can be very useful to arrange them in sub groups.

To facilitate this, Hog now supports an arbitrary structure of sub-folders in the Top directory.

Projects are identified by the directory containing the 'hog.conf' file and the 'list' sub-directory containing the list files.

2.3. Hog GUI buttons

Three custom buttons can be added to the Vivado GUI by running a Hog script:

- CHK: Checks that the project files and properties match what is defined in the Top directory.
- LIST: Recreates all the list files of your projects to match the current project.
- CONF: Recreates the 'Top/[group name]/⟨project name⟩/hog.conf' file to match the current project.

The CHK button can be used to double check if the current project matches with the content of the list and conf files, i.e. what it is going to be committed to the repository.

If new files were added to the project using Vivado GUI, the LIST button can be used to update Hog's list files. Modified list files still need to be committed to git manually.

Unfortunately, in case of nested list files, i.e. list files which include other list files, Hog will not be able to recreate a hierarchical structure and will flatten it.

If any non-default property was set in the project, the CONF button can be used to update the 'hog.conf' file. The use of a configuration file rather than a Tcl script was instrumental to achieve this automatisation.

3. Plans for Hog2022.1

Hog is normally released twice a year in January and June. Next release is scheduled for January 2022 and will include some support for System on Chips (SoCs), at least for Vivado suite.

We also aim to support custom IP repositories (which are already supported by Hog) in a more robust way.

These features are gaining more and more importance as SoCs and custom IP produced with High Level Synthesis (HLS) are becoming widespread in high-energy physics community.

4. Conclusions

Hog is available at `gitlab.cern.ch/hog/Hog` and documented at `cern.ch/hog`. It is well integrated with Xilinx and Intel IDEs, requiring minimal overhead work for developers. It guarantees synthesis and P&R reproducibility and binary file traceability by embedding git-SHA and a numerical version into the firmware.

Hog can be used to allow virtually zero code duplication using the appropriate methodology.

A Hog tutorial was held at CERN (on Zoom only) on the 15^{th} of June 2021. It was very successful, with more than 80 participants. A Recording of the tutorial is available here: `https://www.youtube.com/watch?v=dDcPoeEGVdQ`

Currently Hog is used by several firmware projects within the High-Energy Physics community.

References

[1] F. Gonnella, *A collaborative HDL management tool for ATLAS L1Calo upgrades* PoS *TWEPP2018* (2019), 142
[2] N.V. Biesuz et al. *Hog (HDL on git): a collaborative management tool to handle git-based HDL repository* 2021 *JINST* **16** T04006
[3] `http://git-scm.com/`
[4] `www.xilinx.com/products/design-tools/vivado.html`
[5] `www.xilinx.com/products/design-tools/ise-design-suite.html`
[6] `https://www.xilinx.com/products/design-tools/planahead.html`
[7] `www.intel.com/content/www/us/en/software/programmable/quartus-prime/`