Hog (HDL on Git):

Biesuz, N.v.; Cieri, D.; Gonnella, F.; Loustau de linares, G.; Peck, A.

DOI:
10.1016/j.nima.2023.168016

Nuclear Inst. and Methods in Physics Research, A 1049 (2023) 168016

Contents lists available at ScienceDirect

Nuclear Inst. and Methods in Physics Research, A

journal homepage: www.elsevier.com/locate/nima

Hog (HDL on Git): An easy system to handle HDL on a git-based repository

N.V. Biesuz [b], D. Cieri [c], F. Gonnella [a,*], G. Loustau De Linares [e], A. Peck [d]

[a] The University of Birmingham, B295TT, Edgbaston, Birmingham, United Kingdom
[b] INFN sezione di Ferrara, Polo Scientifico e Tecnologico - Edificio C, Via Saragat 1, 44122 Ferrara, Italy
[c] Experimentelle Teilchenphysik, Max–Planck-Institut für Physik, Föhringer Ring 6 80805, Munich, Germany
[d] Electronics Department, Boston University, 590 Commonwealth Avenue, Boston, MA 02215, USA
[e] University of Massachusetts Amherst, Amherst, MA 01003, USA

ARTICLE INFO

Keywords:
Git
HDL
VHDL
Verilog
Tcl

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 (cern.ch/hog), a set of Tcl and Shell scripts that tackles these issues and is deeply integrated with HDL IDEs, such as Xilinx Vivado Design Suite and ISE PlanAhead or Intel Quartus Prime, and all major simulation tools, like Siemens ModelSim or Aldec Riviera Pro.

Git is a very powerful tool and has been chosen as standard by several research institutions, including CERN. Hog perfectly integrates with git to assure an absolute control of HDL source files, constraint files, IDE and simulation settings. It guarantees traceability by automatically embedding the git commit SHA and a numeric version into the binary file, also automatically renamed.

Hog does not rely on any external tool apart from the HDL IDE and git, so it is extremely compatible and does not require any installation. Developers can get quickly up to speed: clone the repository, run the Hog script, work normally with the IDE.

The learning curve to use Hog for the users is minimal. Once the HDL project is created, developers can work on it either using the IDE graphical interface, or with the provided Shell scripts to run the workflow.

Hog works on Windows and Linux, supports IPbus, Sigasi and provides pre-made YAML files to set up a working Continuous Integration on GitLab (Hog-CI) with no additional effort, which runs the HDL implementation for the desired projects. Other features of Hog-CI are the automatic creation of tags and GitLab releases with timing and utilisation reports.

Currently, Hog is successfully used by several firmware projects within the High-Energy Physics community, e.g. in the ATLAS and CMS Phase-II upgrades.

1. Introduction

Hog (HDL on git) [1,2] is a set of Tcl and Bash scripts, designed to help maintaining HDL projects on git [3] repositories. Hog is completely integrated into the most popular HDL IDEs,[1] such as Xilinx Vivado [4] and ISE [5] (PlanAhead [6]) and Intel Quartus [7], allowing to use the software normally via their graphical interface, or in batch mode thanks to the provided Hog scripts. Hog supports all the external simulators supported by Xilinx, such as Siemens Modelsim [8] and Questasim [9] or Aldec Riviera [10].

2. Working principles

2.1. No installation required

The Hog repository must be added as a submodule to the main git HDL repository. For this reason, no installation is required: Hog does not rely on external libraries or software that would not already be needed for your HDL development. Cloning the git repository (and the submodules) is enough to be ready to synthesise, place and route and produce the binary file or start developing.

Moreover, different versions of Hog can be used in different repositories, so that the users do not need to update Hog unless they choose to do so, saving useless overhead work.

On the other hand, having multiple Hog copies on a machine is not a problem because the Hog module's size is less than 1 MB.

2.2. Guarantee reproducibility and traceability

Hog's main goal is to guarantee the Place&Route reproducibility and the traceability of the produced binary files. To achieve this, Hog has total control of the IDE settings and of all the project files (HDL sources, constraint files, etc.). All these files are listed in dedicated text files,

* Corresponding author.
E-mail address: francesco.gonnella@cern.ch (F. Gonnella).

[1] Integrated Development Environments.

https://doi.org/10.1016/j.nima.2023.168016
Received 31 July 2022; Received in revised form 26 December 2022; Accepted 27 December 2022
Available online 11 January 2023

called list files, with an extension that identifies the file set type to be included in the project, e.g. design sources (`.src`), simulation sources (*.sim*), constraint file (`.con`). Hog exploits the git SHA[2] to extract the version number of the project from specially formatted git tags, that are automatically created by Hog-CI as explained in Section 3. The SHA and the version are then embedded into the binary files by means of generics/parameters.

3. Hog's continuous integration

Hog provides YAML files to operate the Continuous Integration workflow on GitLab CI/CD platforms, using shared docker runners and private machines, with an installation of gitlab-runner and the required IDE and simulation software. Hog-CI builds and simulates the projects and automatically tags new versions in the repository. Optionally GitLab releases can be automatically created.

4. Using Hog locally

Hog-handled repositories require only the Hog submodule and a *Top* folder, which contains the files use by Hog create the HDL projects: configuration files (see Section 4.1) and list files (see Section 4.2). One repository can contain multiple HDL projects, each of them corresponding to a sub-directory of the Top folder, that we refer to as top project-directory. Each of these directories must include a configuration file named `hog.conf` and a `list` sub-directory, containing the list files. Every time a binary file is produced, Hog automatically checks that no uncommitted modifications were done to the project. This certifies that the binary file correspond to a certain commit in the repository and guarantees reproducibility and traceability.

4.1. Project configuration files

The `hog.conf` project configuration file is used by Hog to create the project and define the properties for the IDE and the project. It uses the TOML formatting language syntax [11], with five defined sections.

- *Main section*: containing the project-related settings, such as the target FPGA or the device family;
- *Synthesis and Implementation sections*: containing the settings to configure the synthesis and implementation strategies;
- *Parameter section (Xilinx only)*: including volatile parameters that must be set before launching each run;
- *Hog section*: specifies Hog directives valid for the project.

Custom Tcl scripts can be executed before and/or after the creation of the project by adding them in the top project-directory and naming them `pre-creation.tcl` and `post-creation.tcl`.

An optional simulation configuration file, called `sim.conf`, can be used to specify simulation properties.

4.2. List files

HDL files, IPs and constraint files that need to be added to the HDL project, must be listed in dedicated text files, called *list files*. These are stored inside a sub-directory `list` of the top project-directory. Different extensions are used, depending on the type of files to be included in the project, i.e. `.src` for design sources or `.sim` for simulation sources. List files can be handled recursively, meaning that developers can include a list file inside another, thus reducing duplication. Hog creates a different VHDL library for each `.src` list file.

4.3. HDL sources and Intellectual Properties (IPs)

For the IPs, particular care must be taken to avoid that Vivado/ Quartus-generated files are committed to the repository. Each IP file (`.xci` for Vivado) must be stored in a separate sub-folder and only the IP file must be committed to the repository. The generated files shall be listed in the `.gitignore` file to tell git to ignore them.

5. Summary and conclusions

Hog is available at `gitlab.com/hog-cern/Hog` and documented at `cern.ch/hog`. Official mirror-repository are provided on gitlab.cern.ch, baltig.infn.it, github.com

Hog is released each year in January and June. The latest stable version is the Hog2022.2.

Hog integrates with Xilinx and Intel IDEs, requiring minimal overhead work for developers with no extra installations.

Hog guarantees synthesis and P&R reproducibility and binary file traceability by embedding git-SHA and a numerical version into the firmware.

Hog can help to allow zero code duplication using the appropriate methodology. A template to set up the GitLab Continuous Integration is provided with Hog, allowing the developers to run the P&R workflow and simulations on private runner machines.

A tutorial was held at CERN on the 15th of June 2021. It was very successful, with more than 80 participants. A recording of the tutorial is available at [12].

Hog is currently used by several firmware projects within the High-Energy Physics community, including ATLAS and CMS Phase-II upgrades.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] N.V. Biesuz, et al., Hog (HDL on git): a collaborative management tool to handle git-based HDL repository, JINST 16 (2021) 04006.
[2] Hog Documentation, http://cern.ch/hog.
[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] www.xilinx.com/products/design-tools/planahead.html.
[7] www.intel.com/content/www/us/en/software/programmable/quartus-prime/.
[8] eda.sw.siemens.com/en-US/ic/modelsim/.
[9] eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/.
[10] www.aldec.com/en/products/functional_verification/riviera-pro.
[11] https://toml.io/en/.
[12] Hog Tutorial, https://bit.ly/hog-tutorial.

[2] Secure Hash Algorithm, a checksum identifier that git uses to unambiguously identify a commit in a repository.