MAX-PLANCK-INSTITUT
FÜR PHYSIK

Hog: handling HDL repositories on git

Davide Cieri (MPP Munich) on behalf of the Hog group

23. September 2021 - TWEPP2021

Handling HDL code on git

Hog

Coordinate firmware development among international collaborators

Guaranteeing firmware synthesis with Place and Route (P&R) reproducibility and assuring traceability of binary files is paramount

Hog (HDL on git) exploits advanced git features and integrates with HDL IDEs to tackle these issues

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

WHAT IS HOG?

TCL/SHELL
No extra requirements only your chosen IDE (Vivado, Quartus, ISE)

GIT SUBMODULE
Update when you want. Different versions for different projects

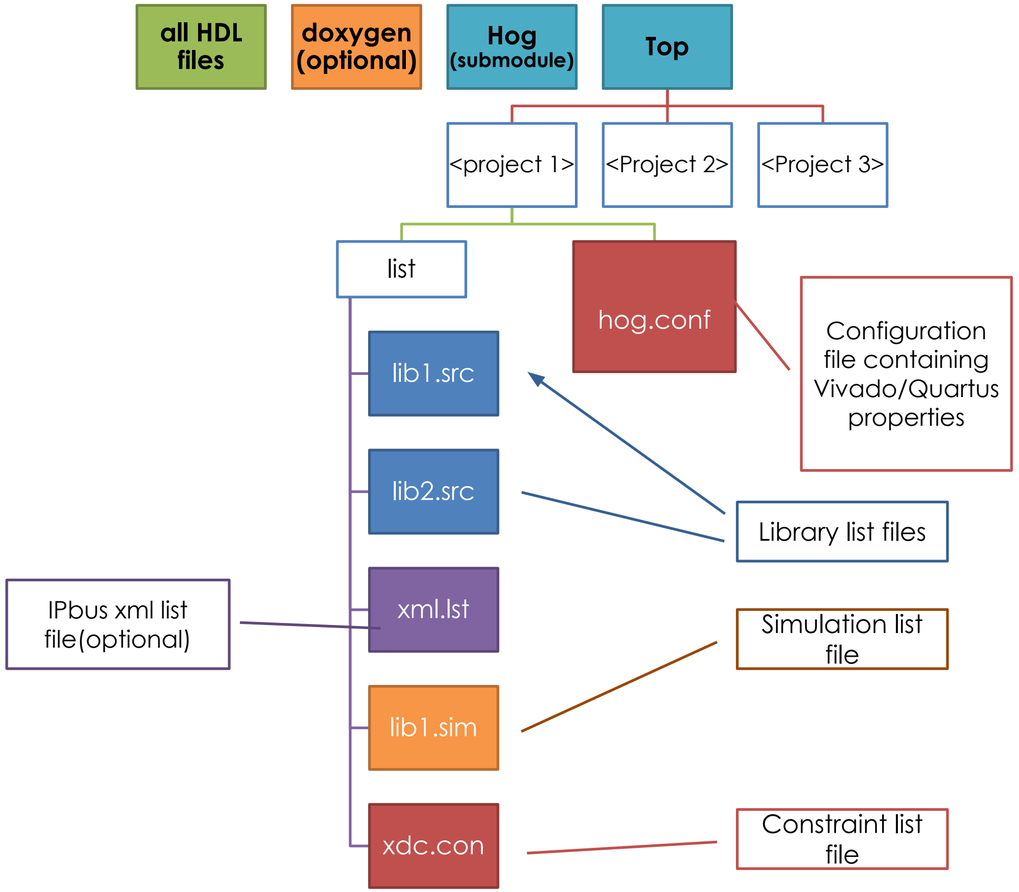
P&R REPRODUCIBILITY
Absolute control of HDL files, constraint files and IDE settings

BINARY TRACEABILITY
Git SHA and version numbers embedded into firmware registers

CONTINUOUS INTEGRATION
Building of firmware in Continuous Integration. Automatic tagging and releasing

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

HOG-HANDLED REPOSITORY



TOP FOLDER

The Top folder includes the Hog projects. Each project subfolder corresponds to a single design and contains the necessary files to create the project

LIST FILES

Plain text files, containing the list of files to be added to the project. Different list files for different file sets (sources, simulation, constraints, external files)

HOG.CONF

Project configuration file (toml), containing the instructions to configure the project properties (FPGA, synthesis and implementation strategies, etc..)

HDL SOURCES

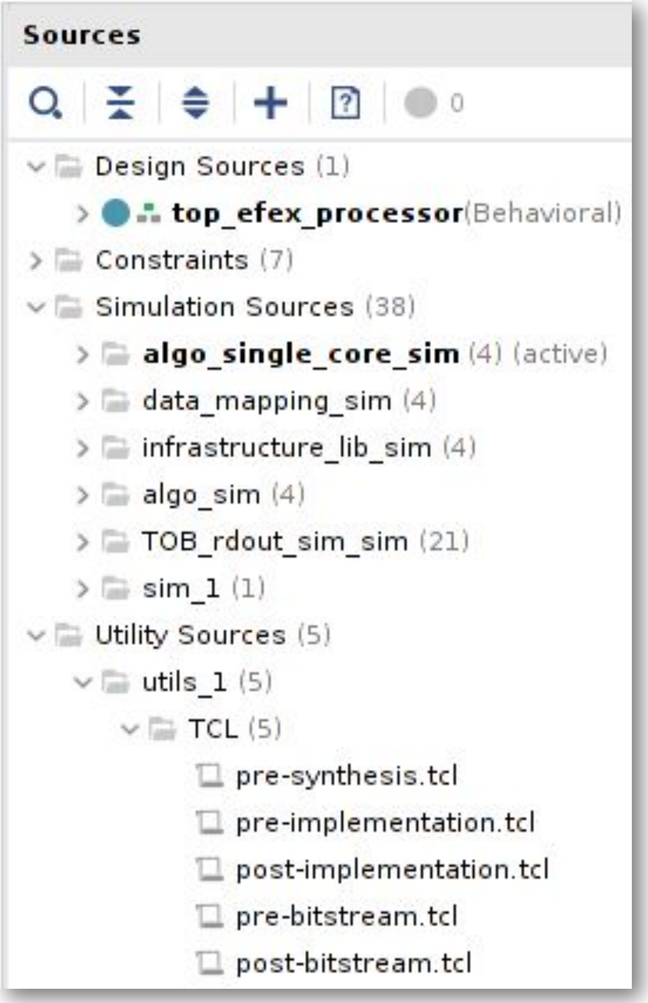
HDL sources can be stored anywhere in the repository. IP and BD files shall be contained in sub-folders with the same name of the file. Only the IP/BD file should be committed to the repository (.xci for Vivado)

AN EXAMPLE OF LIBRARIES IN VIVADO

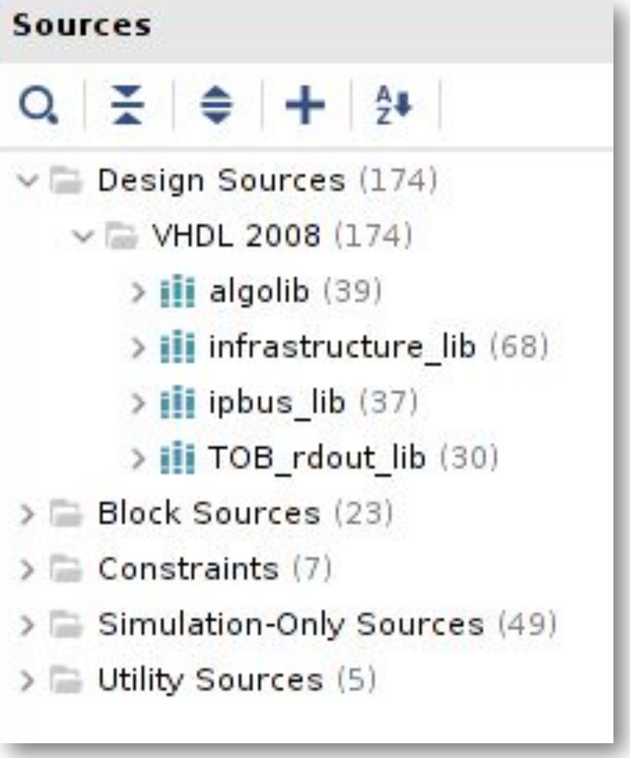
File tree

```
efex_processor.1
├── hog.conf
└── list
    ├── algolib.src
    ├── algo.sim
    ├── algo_single_core.sim
    ├── data_mapping.sim
    ├── infrastructure_lib.sim
    ├── infrastructure_lib.src
    ├── ipbus_lib.src
    ├── TOB_rdout_lib.src
    ├── TOB_rdout_sim.sim
    ├── XDC.con
    └── xml.lst
```

Simulation



Libraries



Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

USING HOG WITH VIVADO

CREATE THE PROJECT

Use the CreateProject.sh script to create the Vivado project

USE THE GUI

Developing can be done using the Vivado GUI in project mode

USE THE SHELL SCRIPTS

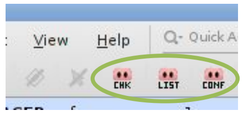
Run the workflow in batch mode

INTEGRATED HOG SCRIPTS

Running at pre-synthesis, pre-implementation, post-implementation and post-bitstream stage. Embed the git SHA and version, and write the reports

ADD NEW FILES / CHANGE THE SETTINGS

New files shall be added to list files and settings to the hog.conf. Users can do this manually and re-create the project, or update the Hog configuration files using the dedicated Hog buttons



VERSIONING

At pre-synthesis stage, Hog evaluates the design version from the git SHA in the vM.m.p format. Version values are calculated for each library in the project

COMMIT BEFORE RUNNING!

Uncommitted changes will generate a Critical Warning, and Hog will declare the repository as dirty, setting the design version to 0

AN EXAMPLE OF VERSIONS

PRE-SYNTHESIS OUTPUT

Hog evaluates at pre-synthesis stage date, time, SHAs and versions for all project components (Repository, Constraints, Top, Hog submodules, libraries)

VERSION AND SHA REGISTERS

The version and SHAs are parsed to the top file in the project as generic parameters, and can be used by the developer.
Version registers are formatted in hex as MM mm pppp

```
---------------------- PRE SYNTHESIS ----------------------
27/04/2021 at 11:19:57
Firmware date and time: '26042021', '00155921'
Project flavour: 1
Global SHA: fcc220c6, VER: 0.8.1
Constraints SHA: F218A30C, VER: 0.8.1
IPbus XML SHA: 91923485, VER: 0.8.0
Top SHA: FCC220C6, VER: 0.8.1
Hog SHA: C522A04, VER: 4.4.0
--- Libraries ---
ipbus_lib SHA: EAEDCE6B, VER: 0.8.0
infrastructure_lib SHA: 20310FED, VER: 0.8.1
--- External Libraries ---
-----------------------------------------------------------
```

Hog uses the commit time and date to guarantee reproducibility

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

AVOID CODE DUPLICATIONS WITH HOG "FLAVOURS"

MULTIPLE DESIGN SHARING SAME TOP HDL FILE

E.g. Different FPGA running the same design

HOG FLAVOUR

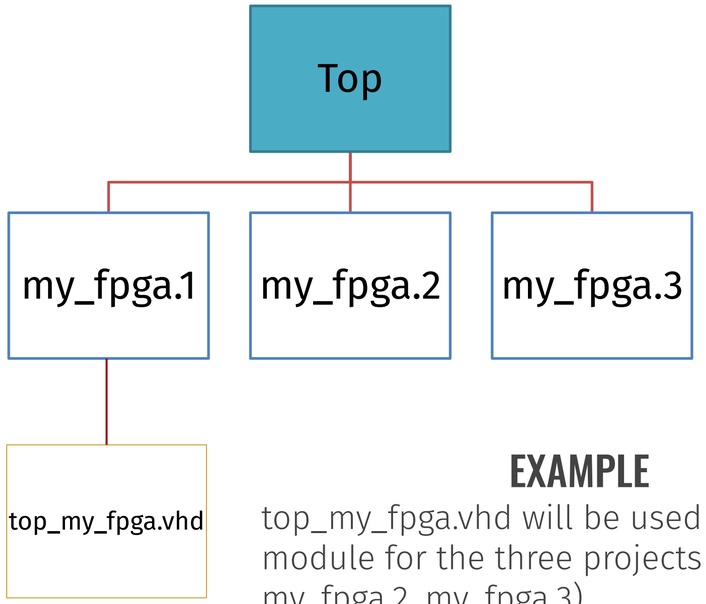
Integer number parsed as a generic to the top HDL file. Flavour is extracted from the project folder name, if it ends with a numeric extension (e.g. Top/my_fpga.1)

SHARED TOP HDL FILE

Top file and top module must be called just called top_<project_name> without the numeric extension (e.g. top_my_fpga.vhd)

FLAVOUR AS A GENERIC

Users can use the generic FLAVOUR variable in conditional-statement blocks to create different project as a function of the flavour



EXAMPLE

top_my_fpga.vhd will be used as a top module for the three projects (my_fpga.1, my_fpga.2, my_fpga.3).
The FLAVOUR number will be parsed to the top module and can be used to differentiate the designs

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

HOG CONTINUOUS INTEGRATION WORKFLOW

1. OPEN A MERGE REQUEST (MR)

Developments are done on short-lived feature branches. To push changes to main branch, open a merge request on the GitLab repository webpage

2. MERGE REQUEST PIPELINE

Runs on private runners with GitLab runner and Vivado installed. Runs the P&R workflow and the simulations for the specified projects

3. ACCEPT THE MERGE REQUEST

The repository librarian checks the changes and, if the merge request pipeline, was successful, he/she merges the feature branch into the main branch

4. MASTER PIPELINE

Runs on shared machines with docker, and automatically tags the repository. Special keyword can be used in the MR description to increase automatically the minor or major version numbers

5. TAG PIPELINE

Creates automatically the GitLab release for the just-produced tag, including the version and timing tables, the generated binary files, and a changelog, that can be filled using special keywords when committing

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

SET UP THE HOG CI

NORMAL CI

Include the hog.yml in your .gitlab-ci.yml file. Write few lines for each project, different CI jobs for simulation and P&R

```yaml
include:
    - project: 'hog/Hog'
      file: '/hog.yml'
      ref: 'v0.2.1'

#################### example ###########################
### Change 'example' with your project name

generate_project:example:
    extends: .generate_project
    variables:
      extends: .vars
      PROJECT_NAME: example
      HOG_ONLY_SYNTH: 0 # if 1 runs only the synthesis

simulate_project:example:
    extends: .simulate_project
    variables:
      extends: .vars
      PROJECT_NAME: example
```

DYNAMIC CI (EXPERIMENTAL)

Include the hog-dynamic.yml in your .gitlab-ci.yml. The CI configuration is created dynamically, and the merge-request pipeline is executed in a child-pipeline

```yaml
include:
    - project: 'hog/Hog'
      file: '/hog-dynamic.yml'
      ref: 'v0.2.1'
```

EXTRA CONFIGURATIONS

Hog-CI can be customised to tailor it to the requirements of any project.

Optional features include:

- Adding custom user jobs to the Hog-CI
- Automatic Gitlab releases
- Archive of binary files to EOS cloud storage
- Automatic generation of doxygen documentation
- Avoid building projects that have not been changed in the merge request

GITLAB RELEASE EXAMPLES

Official version: v0.1.1

› Assets 4

Evidence collection

v0.1.1-evidences-4644.json ••• 60bda14e

Collected 3 weeks ago

Repository info

- Merge request number: 7
- Branch name: feature-enlarge-rams

Changelog

- Increase input RAMs depth from 1024 to 65535

fmc0 version table

File set	Commit SHA	Version
Global	5b6ce4fb	0.1.1
Constraints	d01aaee3	0.0.8
IPbus XML	5b6ce4fb	0.1.1
Project Tcl	87e2e73a	0.0.8
Hog	48c98b7	3.8.0
Lib: fmc0	5b6ce4fb	0.1.1
Lib: ipbus	8e5214ea	0.1.0

fmc0 Timing summary

Parameter	value (ns)
WNS:	0.048110
TNS:	0.000000
WHS:	0.010556
THS:	0.000000

Time requirements are met.

Downloads

- fmc0.zip

Release note automatically generated by **Hog**.

a8f0e110 v0.1.1 Created 3 weeks ago by

Davide Cieri (Max-Planck-Institut für Physik) - davide.cieri@cern.ch

HOG VERSION TIMELINE

FIRST COMMIT (NOV 2017)
First version of Hog

HOG PRESENTED AT TWEPP2018
Link

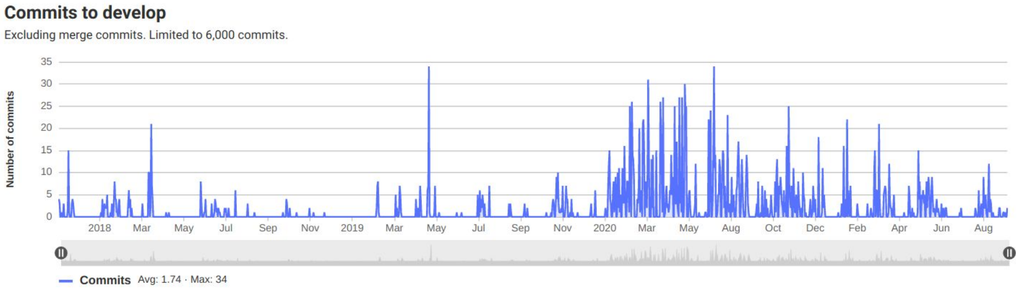
HOG v0.0.1 (DEC 2019)
First beta-version

HOG2020.1 (JAN 2020)
First official version

HOG2020.2 (JUN 2020)
Unified CI Stages
Hog dynamic CI
unified launcher LaunchWorkflow
Check project version before running the CI

HOG2021.1 (JAN 2021)
Custom user jobs in the CI
Sigasi support
Seperate building branches
Improve pipeline

HOG2021.2 (JUN 2021)
Hog.conf files
Group projects inside subfolders in Top
Custom Hog buttons to Vivado
Supports for all Xilinx-supported simulators



Hog 2022.1
January 2022

Hog

CONFIGURE SIMULATIONS WITH HOG.CONF
Each simulation sets can be configured using the dedicated section in the hog.conf file

TRACKING OF CUSTOM USER IP REPOSITORIES
A version number is assigned to each user IP repository in the project

IMPROVE QUARTUS SUPPORT
Adding timing report and version comparison script

IMPROVE GITLAB.COM SUPPORT
Fix release creation outside gitlab.cern.ch

Summary and Conclusions

Hog is available at gitlab.cern.ch/hog/Hog

- Active project, involving 5 developers
- Released twice a year under Apache 2 licence
- Next release Hog2022.1 in January 2022
- Experimental features are available in the develop branch
- Used by several projects, including: ATLAS and CMS Phase-I and Phase-II upgrades, GAPS, FOOT

Documentation: cern.ch/hog, support: hog-group@cern.ch, mailing list: hog-users@cern.ch

Hog Tutorial at CERN (YouTube link)

Do you want to try it?

```
> git clone --recursive https://gitlab.cern.ch/bham-dune/zcu102.git
> cd zcu102
> ./Hog/CreateProject.sh fmc0
> vivado ./Projects/fmc0/fmc0.xpr
```

MAX-PLANCK-INSTITUT
FÜR PHYSIK

The icons used in this presentation have been designed using resources from Flaticon.com

Thanks!

Davide Cieri (MPP Munich) on behalf of the Hog group

23. September 2021 - TWEPP2021