Hog: handling HDL repositories on git

N. Aranzabal (ESRF, Grenoble) on behalf of the Hog group

13 June 2024 - 1st FPGA Developers Forum (FDF)

Hog: handling HDL repositories on git

- Facilitates HDL projects development among multiple collaborators
- A set of TCL scripts (<1.2 MB) plus a methodology
- Integrates with HDL IDEs to tackle advanced Git features
- “Zero effort” strategy to maintain HDL projects in Git or even to develop them locally

WHAT IS HOG?

GIT SUBMODULE

Update when you want. Different versions for different projects

BINARY TRACEABILITY

Git SHA and version ID embedded into firmware registers

TCL/SHELL

No extra requirements only your chosen IDE (Vivado, Quartus, Libero, ISE)

PLACE & ROUTE REPRODUCIBILITY

Absolute control of HDL files, constraints and IDE settings

CONTINUOUS INTEGRATION

Automatic firmware validation and verification, plus tagging and releasing

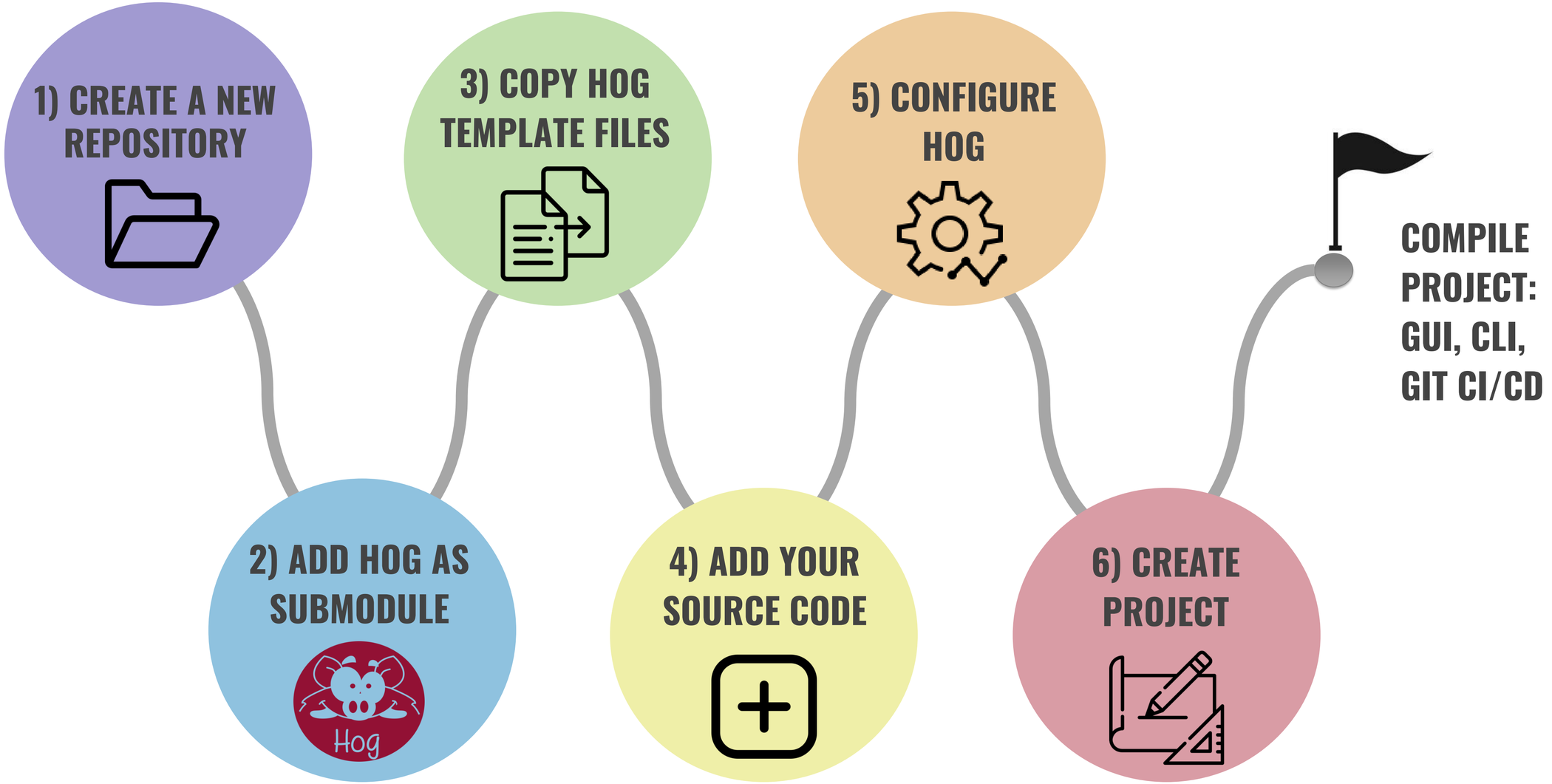
What do you need to work with Hog

A PC

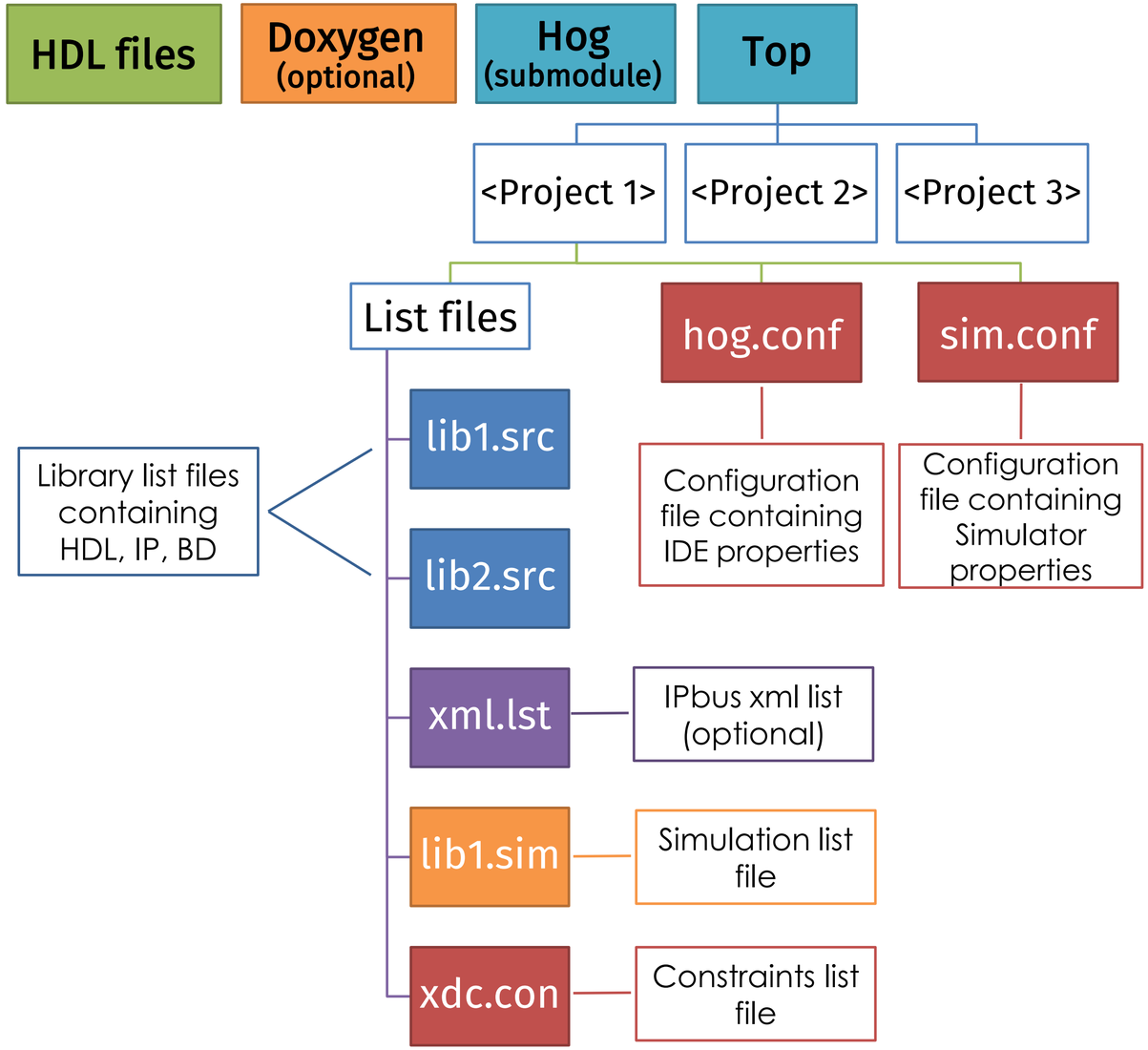
Git

The IDE of your choice

USING HOG: A PRACTICAL POINT OF VIEW



HOG-HANDLED REPOSITORY



TOP FOLDER

Hog projects dependencies and configuration. Each project subfolder corresponds to a single design and contains the necessary files to create the project

LIST FILES

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

HOG & SIM CONF

Project configuration: FPGA device, synthesis and implementation strategies… Simulation configuration: compilation strategy, syntax, pass string…

HDL SOURCES

HDL sources can be stored anywhere in the repository

LIST FILES

A list file contains a list of the files to be included in a project in text format

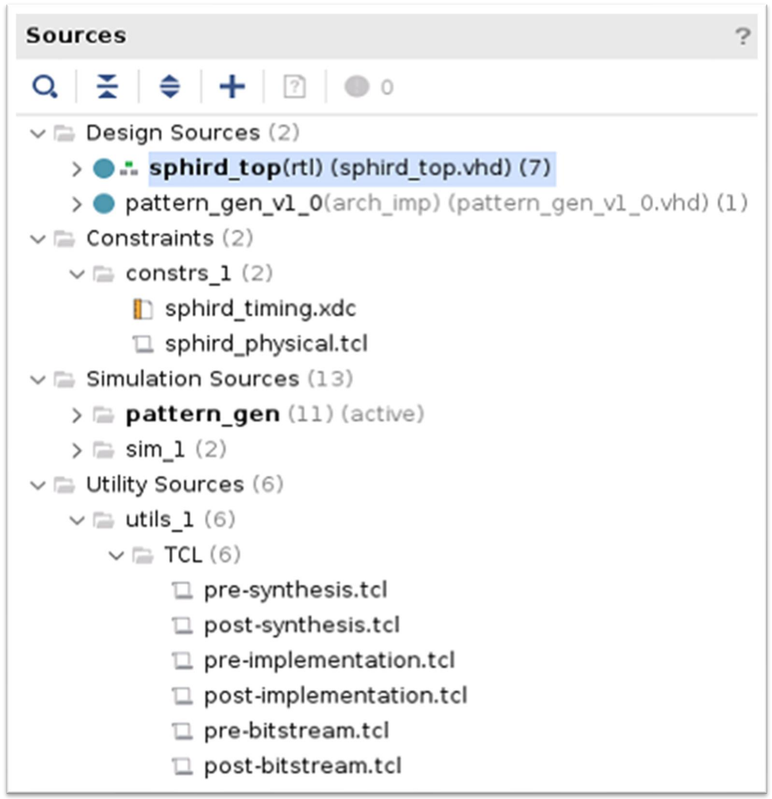
- **List file extension defines the type of content**
  - *.src for source files (HDL, IP, block design)
  - *.sim for simulation files
  - *.con for constraint files
  - *.ext for external proprietary libraries
- **List files are handled recursively: a list file can include another list file**
- **Hog will create a VHDL library for each list file**
  - E.g. if you create a file called lib1.src and place it into Top/project1/list/, when you create project1, the files listed in lib1.src will be included in Vivado in a library called lib1
- **If not interested in using libraries, developers can create just one list file**

AN EXAMPLE OF SOURCES AND LIBRARIES IN VIVADO

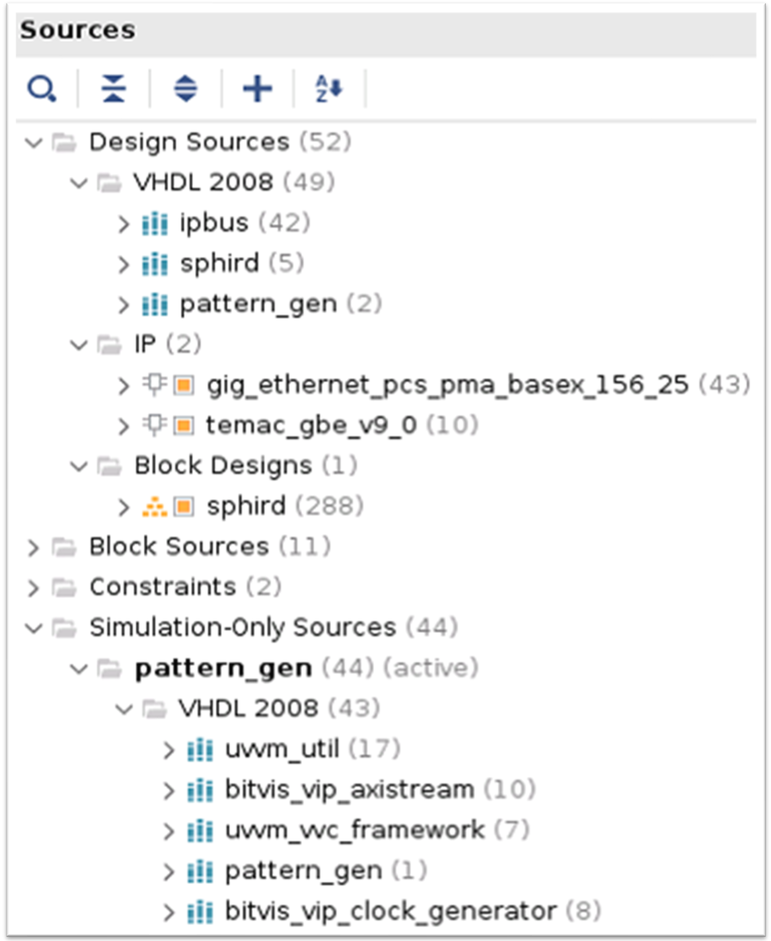
Top directory file tree

```
.
├── repo.conf
├── sim_lib
│   ├── bitvis_vip_axistream.sim
│   ├── bitvis_vip_clock_generator.sim
│   ├── esrf_vip_absenc.sim
│   ├── uvvm_util.sim
│   └── uvvm_vvc_framework.sim
└── sphird
    ├── hog.conf
    ├── list
    │   ├── bd.src
    │   ├── ipbus.src
    │   ├── ip.src
    │   ├── pattern_gen.sim
    │   ├── pattern_gen.src
    │   ├── sphird.con
    │   ├── sphird.src
    │   └── xml.lst
    └── sim.conf
```

Design & Simulation sources



Libraries



IP & BD sources

HOG.CONF EXAMPLE

```
#·vivado·2022.1

[parameters]
MAX_THREADS=1

[main]
IP_REPO_PATHS=hdl/custom_ip
PART=xczu4cg-fbvb900-1-e
SIMULATOR_LANGUAGE=VHDL
SOURCE_MGMT_MODE=All
TARGET_SIMULATOR=Questa

[impl_1]
STEPS.PHYS_OPT_DESIGN.IS_ENABLED=1
STEPS.WRITE_BITSTREAM.ARGS.BIN_FILE=1

[hog]
ALLOW_FAIL_ON_CONF=False
ALLOW_FAIL_ON_GIT=False
ALLOW_FAIL_ON_LIST=False
```

IDE to use

PROJECT PARAMETERS

HOG PARAMETERS

EMBEDDED SCRIPTS

Embedded scripts as part of Hog workflow:

Pre-synthesis → Post-implementation → Pre-bitstream → Post-bitstream

1) Pre-synthesis

- Check repository status (Critical Warning if not clean)
- Calculate versions & SHAs and feed them as generics
- Produce *version.txt* containing all versions & SHAs
- Checks YML file (optional)
- Checks IPbus address maps (optional)
- Generate and copy IPbus XMLs (optional)

```
## sphird Version Table

| **File set** | **Commit SHA** | **Version** |
| --- | --- | --- |
| Global | 7fd5d94 | 0.0.3 |
| Constraints | 7fd5d94 | 0.0.3 |
| IPbus XML | de64975 | 0.0.2 |
| Top Directory | 9c3f430 | 0.0.3 |
| Hog | e712587 | 7.31.1 |
| **Lib:** others | f304d45 | 0.0.3 |
| **Lib:** ip | d8c4892 | 0.0.2 |
| **Lib:** bd | d8c4892 | 0.0.2 |
| **Lib:** ipbus | 90536eb | 0.0.2 |
| **Lib:** sphird | d47a402 | 0.0.3 |
```

version.txt

EMBEDDED SCRIPTS

PRE-SYNTHESIS OUTPUT

Hog evaluates at pre-synthesis stage date, time, SHA and version for all project components: Repository, Constraints, Top, Hog submodules, libraries

```
286  ------------------------ PRE SYNTHESIS ------------------------
287  27/04/2021 at 11:19:57
288  Firmware date and time: '26042021', '00155921'
289  Project flavour: 1
290  Global SHA: fcc220c6, VER: 0.8.1
291  Constraints SHA: F218A30C, VER: 0.8.1
292  IPbus XML SHA: 91923485, VER: 0.8.0
293  Top SHA: FCC220C6, VER: 0.8.1
294  Hog SHA: C522A04, VER: 4.4.0
295  --- Libraries ---
296  ipbus_lib SHA: EAEDCE6B, VER: 0.8.0
297  infrastructure_lib SHA: 20310FED, VER: 0.8.1
298  --- External Libraries ---
299  ---------------------------------------------------------------
```

Hog uses commit time & date to guarantee reproducibility

VERSION AND SHA REGISTERS

The versions 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

EMBEDDED SCRIPTS

2) Post-implementation

- Copy all reports, log files and *version.txt* file
- Copy a timing *recap .txt* file, containing TNS and WNS (CI only)
  - This file name is *timing_ok.txt* if there is no violation and *timing_error.txt* if timing requirements are not met

3) Pre-bitstream

- E.g. to embed git SHA in binary file (in case the file gets renamed)

4) Post-bitstream

- Copy *.bit* and *.bin* files to a bin folder and rename them with git describe
- Copy IPbus *.xml* files (replacing place holders with git SHA and version)

Custom TCL scripts can be added at any stage of the workflow in Top/<MY_PROJECT>:
pre-creation.tcl, post-creation.tcl, pre-synthesis.tcl, post-synthesis.tcl, pre-implementation, post-implementation, pre-bitstream.tcl, post-bitstream.tcl

BINARY REPRODUCIBILITY

Two Vivado binary files produced with Linux on two different machines:

- ***.bin*** files are exactly the same
  - *If you run diff, you get nothing*
- ***.bit*** files differ if you run diff
  - *The difference is only a timestamp in the header of the file, the rest is exactly the same:*

```
0000000 f00f0900 f00ff00f 0000f00f 40006101
0000020 5f706f74 78656665 6f72705f 73736563
0000040 433b726f 52504d4f 3d535345 45555254
0000060 6573553b 3d444972 32434346 36433032
0000100 7265563b 6e6f6973 3230323d 00322e30
0000120 370f0062 35357876 66667430 32393167
0000140 00630037 3230320b 34302f31 0036322f
0000160 32090064 38343a31 0031353a a9250165
0000200 ffffff38 ffffffff ffffffff ffffffff
0000220 ffffffff ffffffff ffffffff ffffffff
*
0000640 000000ff 002211bb ffffff44 ffffffff
0000660 5599aaff 00002066 e0033000 00000001
0000700 8000300c 00000001 00002012 20023000
+ +--1084683 lines: 0000720 df175001 00023080 00020001 80
~
```

```
0000000 f00f0900 f00ff00f 0000f00f 40006101
0000020 5f706f74 78656665 6f72705f 73736563
0000040 433b726f 52504d4f 3d535345 45555254
0000060 6573553b 3d444972 32434346 36433032
0000100 7265563b 6e6f6973 3230323d 00322e30
0000120 370f0062 35357876 66667430 32393167
0000140 00630037 3230320b 34302f31 0037322f
0000160 31090064 39323a35 0039333a a9250165
0000200 ffffff38 ffffffff ffffffff ffffffff
0000220 ffffffff ffffffff ffffffff ffffffff
*
0000640 000000ff 002211bb ffffff44 ffffffff
0000660 5599aaff 00002066 e0033000 00000001
0000700 8000300c 00000001 00002012 20023000
+ +--1084683 lines: 0000720 df175001 00023080 00020001 8
~
```

AVOID DUPLICATING CODE WITH HOG

MULTIPLE DESIGNS SHARING THE SAME TOP HDL FILE

E.g. Different FPGA running the same design

HOG FLAVOUR

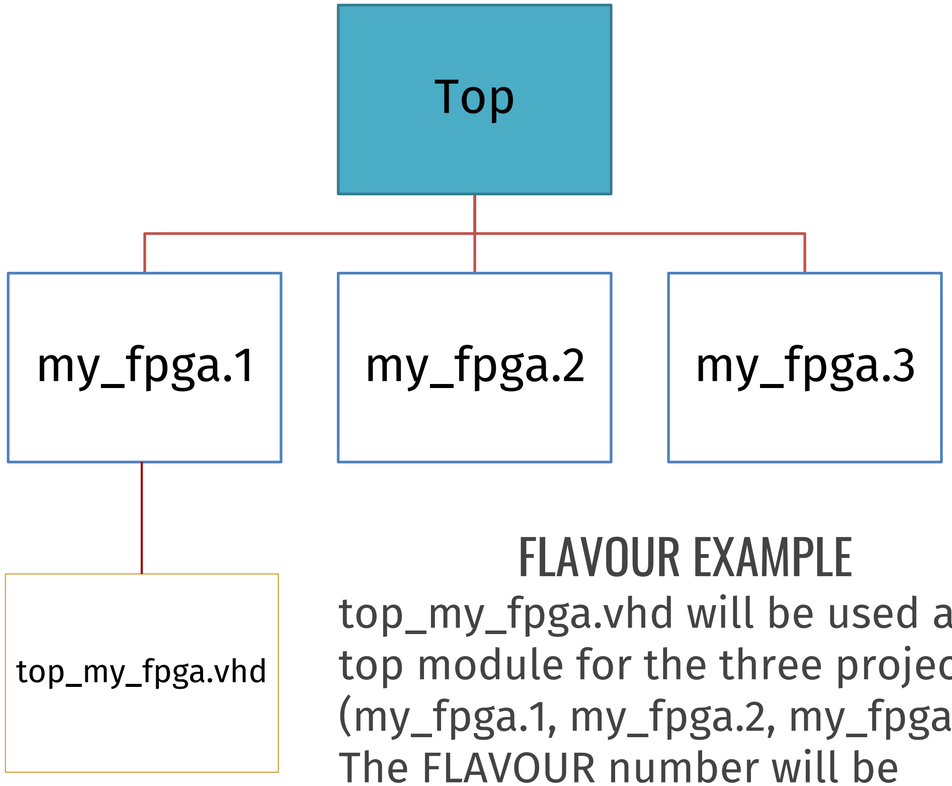
Integer number parsed as a generic to the top HDL file. Flavour is extracted from project folder name, if it ends with a numeric extension (e.g. Top/my_fpga.1)

RECURSIVE LIST FILES

List file can include other list files, which can be then included in multiple projects

USER GENERICS

Users can define generics in hog.conf file to be parsed to the project top module. This can be used in conditional statements in the code, to differentiate between projects



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

USING HOG WITH VIVADO

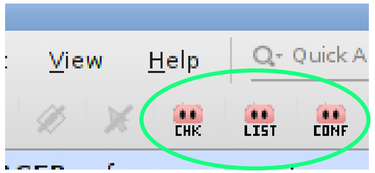
CREATE PROJECT
Use the Do script with CREATE option to create Vivado project

USE THE GUI
Developing can be done using Vivado GUI in project mode

USE THE SHELL SCRIPTS
Run Hog workflow in batch mode

INTEGRATED HOG SCRIPTS
Running at pre-synthesis, pre-implementation, post-implementation and post-bitstream stages. Embed git SHA and version, and create reports

ADD NEW FILES / CHANGE 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

A SIMULATION EXAMPLE: VIVADO, UVVM AND QUESTASIM

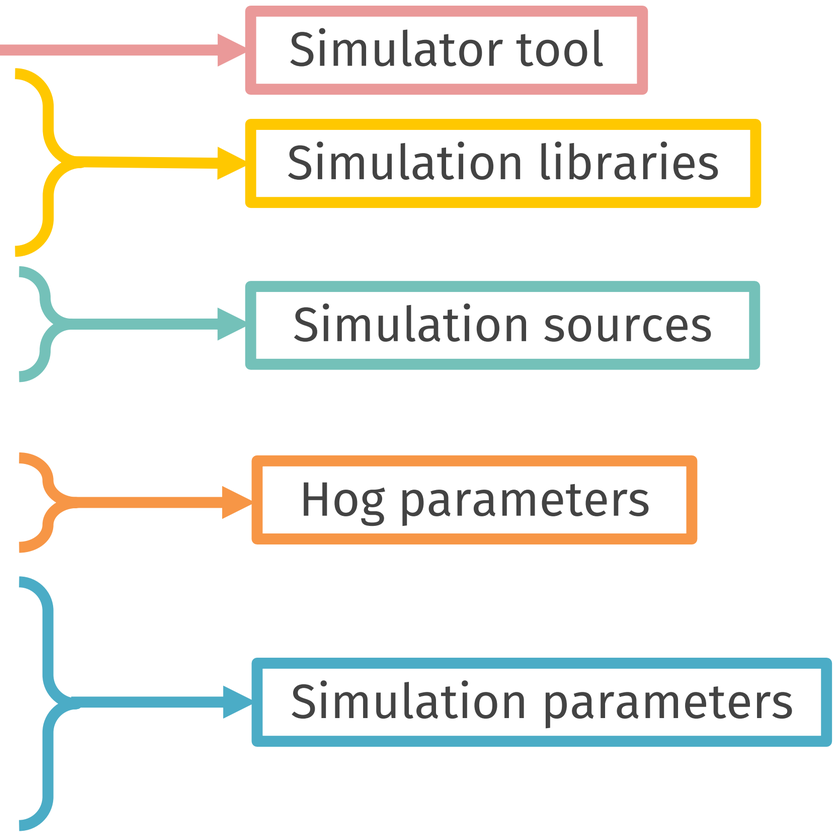
1. **Add UVVM as a submodule** (e.g. in <my_repository>/sim/uvvm)
2. **Create simulation list files with UVVM libraries**
   - E.g. create Top/<my_project>/list/uvvm_util.sim and add the paths to the files in uvvm/uvvm_util/src
3. **Create main simulation list file and simulation configuration file**

Top/<my_project>/list/<my_simulation>.sim

```
#·Simulator·questa
Top/<my_project>/list/uvvm_vvc_framework.sim·lib=uvvm_vvc_framework
Top/<my_project>/list/uvvm_util.sim·lib=uvvm_util
Top/<my_project>/list/bitvis_vip_clock_generator.sim·lib=bitvis_vip_clock_generator
Top/<my_project>/list/bitvis_vip_sbi.sim·lib=bitvis_vip_sbi

sim/my_sim/src/my_sim_vvc_th.vhd
sim/my_sim/src/my_sim_tb.vhd
```

Top/<my_project>/sim.conf

```
[hog]
HOG_SIMPASS_STR=">>·Simulation·SUCCESS:"

[my_sim]
ACTIVE=1
INCREMENTAL=0
QUESTA.COMPILE.VHDL_SYNTAX=2008
QUESTA.SIMULATE.CUSTOM_WAVE_DO=../../../../../../sim/my_sim/questa/wave.do
QUESTA.SIMULATE.RUNTIME=full
TOP=my_sim_tb
```



SETTING UP HOG CI/CD

STATIC GITLAB CI/CD

Include the hog.yml in your .gitlab-ci.yml file. Write few lines for each project, different CI/CD 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 GITLAB CI/CD

Include the hog-dynamic.yml in your .gitlab-ci.yml. The CI/CD 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'
```

GITHUB ACTIONS

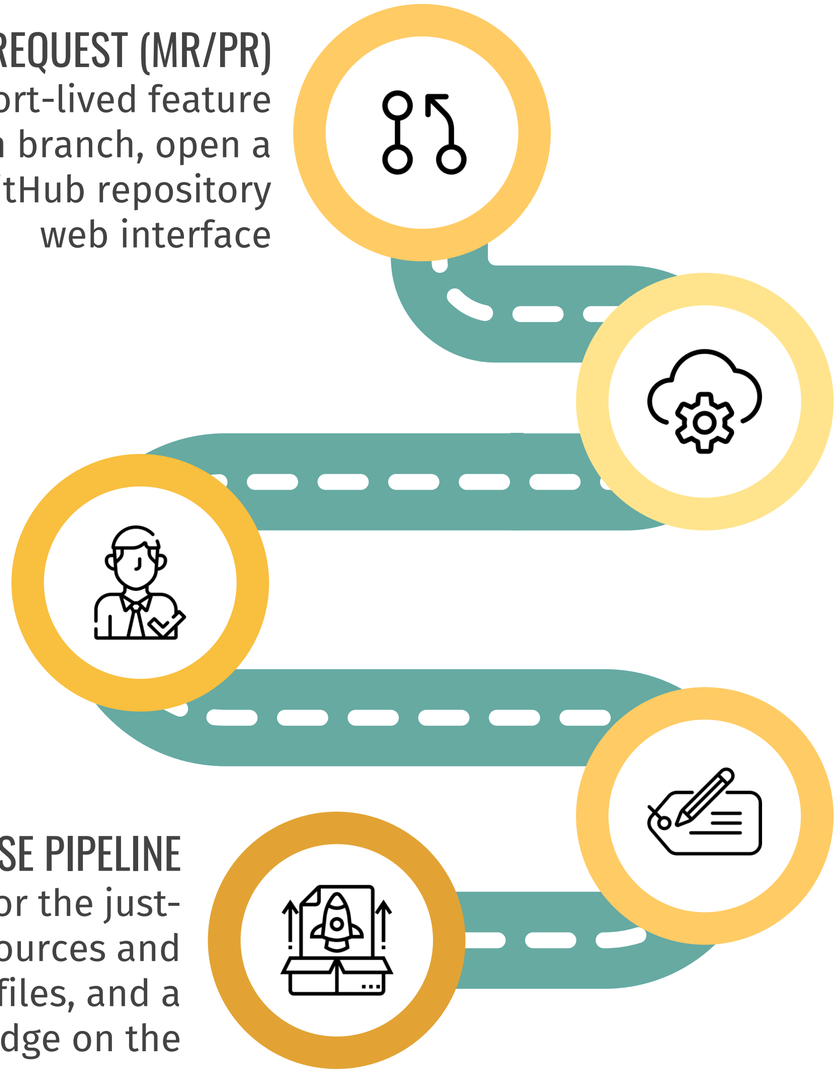
Include the Hog-pull.yml in your .github/workflow YAML configuration, and declare the projects to build and required configuration

```yaml
name: Deploy

on:
  pull_request:
    branches: [master, main]

jobs:
  hog-workflow:
    uses: hog-CERN/Hog/.github/workflows/Hog-pull.yml@Hog2023.1
    secrets:
      SUBMODULE_CONTENT_PULL_KEY: ${{ secrets.SUBMODULE_CONTENT_PULL_KEY }}
      HOG_PUSH_TOKEN: ${{ secrets.HOG_PUSH_TOKEN }}
      HOG_EMAIL: ${{ secrets.HOG_EMAIL}}
      HOG_USER: ${{ secrets.HOG_USER}}
      EOS_USER: ${{ secrets.EOS_USER }}
      EOS_PASSWORD: ${{ secrets.EOS_PASSWORD }}
      HOG_PATH: ${{ secrets.HOG_PATH }}
      HOG_XIL_LICENSE: ${{ secrets.HOG_XIL_LICENSE }}
      HOG_IP_PATH: ${{ secrets.HOG_IP_PATH }}
      HOG_TARGET_BRANCH: ${{ secrets.HOG_TARGET_BRANCH }}
      HOG_DEVELOP_BRANCH: ${{ secrets.HOG_DEVELOP_BRANCH }}
    with:
      BUILD_PROJECTS: >-
        ["example"]
      SIM_PROJECTS: >-
        ["example"]
```

HOG CI/CD WORKFLOW

1. OPEN A MERGE/PULL REQUEST (MR/PR)

Developments are done on short-lived feature branches. To push changes to main branch, open a merge/pull request on GitLab/GitHub repository web interface



2. MERGE/PULL REQUEST PIPELINE

Runs on private machines with the installed IDE. Runs the P&R workflow and the simulations for the specified projects

3. MERGE/PULL REQUEST IS READY

The repository maintainer reviews the changes and, if the MR/PR request pipeline was successful, he/she merges the feature branch into the release branch

4. TAG PIPELINE

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

5. RELEASE PIPELINE

Creates automatically the release for the just-produced tag, including version, resources and timing tables, generated binary files, and a changelog. Optionally, also creates a badge on the repository home page

GITLAB RELEASE EXAMPLE

Official version: v0.0.2

› Assets 4

Evidence collection

v0.0.2-evidences-875.json ••• 1ebb6939

Collected 1 week ago

Repository info

- Merge request number: 1
- Branch name: 1-base-project-with-ipbus

MR Description

Closes #1

Changelog

- Add IPBus:
  - IPBus interface in SFP1 (SFP to RJ45 adaptor)
  - Add IPBus ipbus_ctrlreg_v
  - Add IPBus ram_pattern_generator
  - Add IPBus ipbus_ported_dpram (1kword)
  - Add python script to read/write registers
  - Add python script to test ram (by direct read/v
- Clock generator, MGT MUX, and SFP1 required by
- Add Hog
- Project using Vivado 2023.1
- Definition of timing constraints using clock groups

sphird-v0.0.2 | LUTs: 19.10% FFs: 12.83% BRAM: 33.59% URAM: 0.00% DSPs: 0.41% | timing OK

sphird Implementation Utilization report

Site Type	Used	Fixed	Prohibited	Available	Util%
CLB LUTs	16779	0	0	87840	19.10
CLB Registers	22531	2	0	175680	12.83
Block RAM Tile	43	0	0	128	33.59
URAM	0	0	0	48	0.00
DSPs	3	0	0	728	0.41
Bonded IOB	84	84	0	204	41.18

sphird Timing summary

Parameter	value (ns)
WNS:	0.223746
TNS:	0.000000
WHS:	0.012434
THS:	0.000000

Time requirements are met.

Release binaries and reports automatically stored in disk (optional)

sphird Version Table

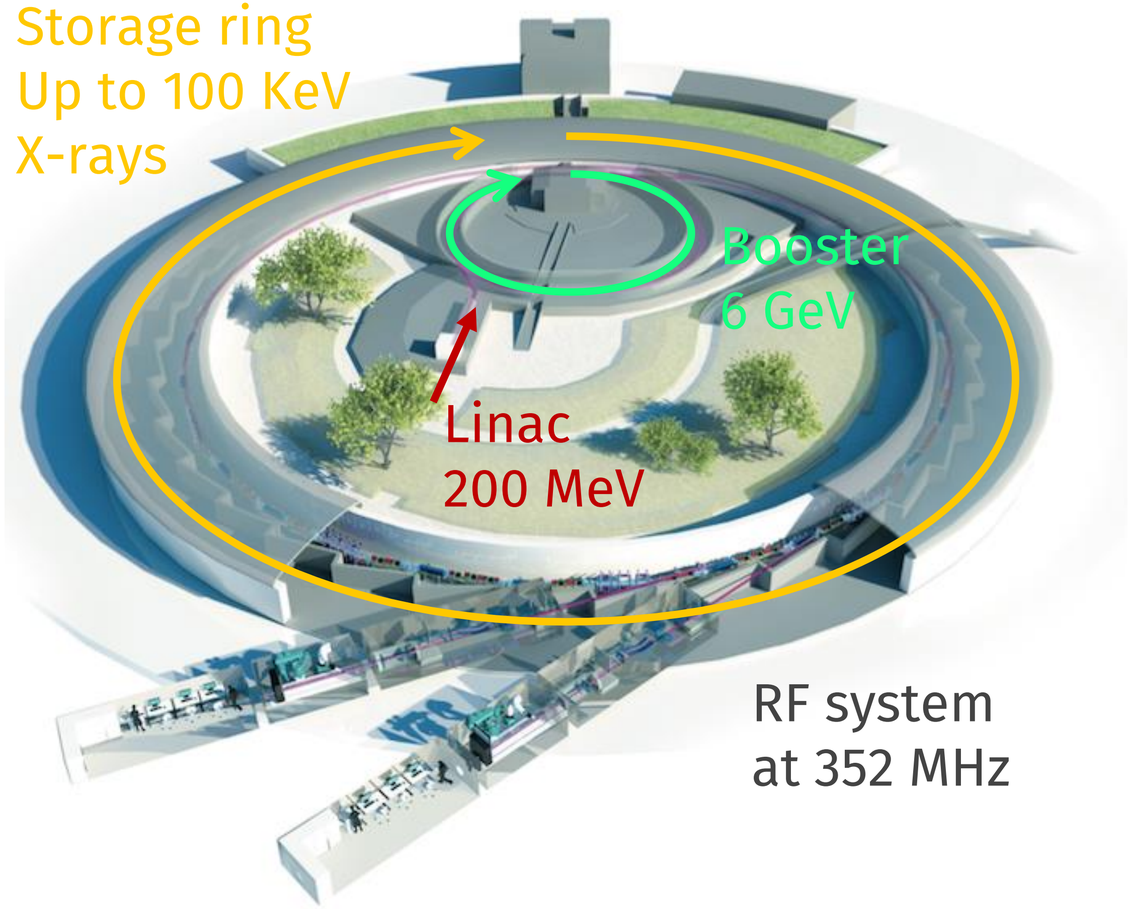
File set	Commit SHA	Version
Global	de649755	0.0.2
Constraints	90536eb6	0.0.2
IPbus XML	de649755	0.0.2
Top Directory	280d12ef	0.0.2
Hog	b1f9a84	7.17.0
Lib: others	6ec62e72	0.0.2
Lib: ip	d8c4892f	0.0.2
Lib: bd	d8c4892f	0.0.2
Lib: ipbus	90536eb6	0.0.2
Lib: sphird	90536eb6	0.0.2

Downloads

In case multipart archives are created (e.g. .z01, .z02, etc.),

- sphird.zip

HOG AT THE EUROPEAN SYNCHROTRON (ESRF)



844 m of circumference
43 beamlines

Detector & Electronics group

- Support, development and production in the area of X-ray detectors and electronics systems for data acquisition, control and instrumentation

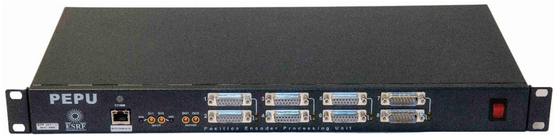
FPGA development methodology for new projects

- Moving from SVN to GitLab
- Continue with AMD as FPGA vendor
- ZYNQ SoC device (get rid of external processors)
- Enclustra SoM for low and mid-end projects
- AMD Versal for high-end projects
- Hog to handle HDL repositories

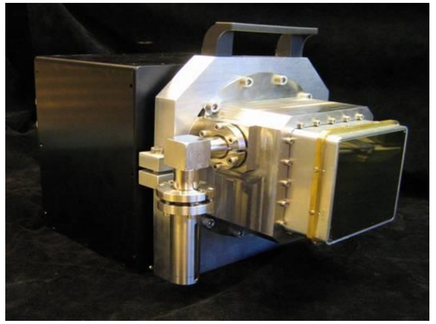
HOG AT THE EUROPEAN SYNCHROTRON (ESRF)

Several projects already ramping up with Hog

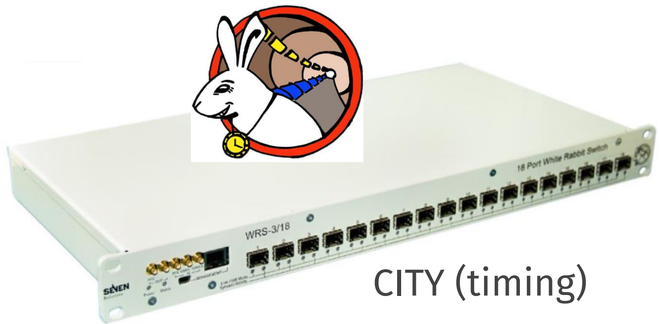
- Collaborations among other synchrotrons and institutes
- Vivado projects with IP (.xci) and block design (.bd)
- Most projects in VHDL, but also Verilog or System Verilog
- Planning to handle projects using HLS
- UVVM and QuestaSim for simulation, but collaborators might use different tools
- Hog-CI with Docker running in high performance servers



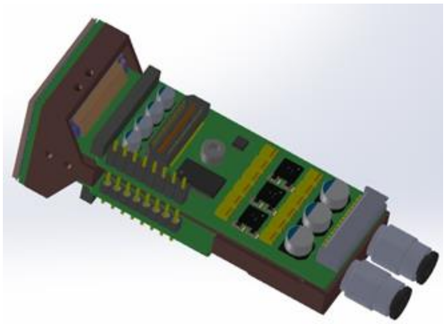
PEPU (Positioning Encoder Processing Unit)



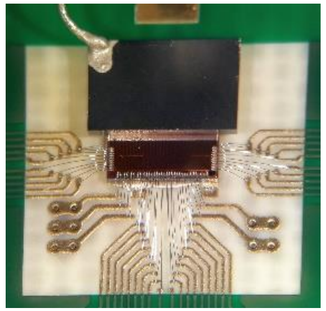
sCMOS camera (detector)



CITY (timing)

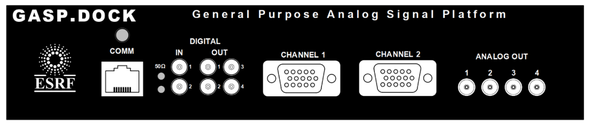


SMARTPIX (detector)



SPHIRD (detector)

GASP (General-purpose Analog Signal Platform)



Summary and Conclusions

Hog is available at gitlab.com/hog-cern/hog

- First commit in November 2017
- Released twice a year under Apache 2 license
- Latest release Hog2024.1, released in February
- Currently 7 contributors
- Experimental features available in the develop branch
- Used by several academic and industrial projects, including: ATLAS, CMS Phase-I and Phase-II upgrades, ESRF, GAPS, FOOT, NASDAQ, NOKIA

Documentation: hog.readthedocs.io
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/Do CREATE fmc0
> vivado ./Projects/fmc0/fmc0.xpr
```

Thank you!

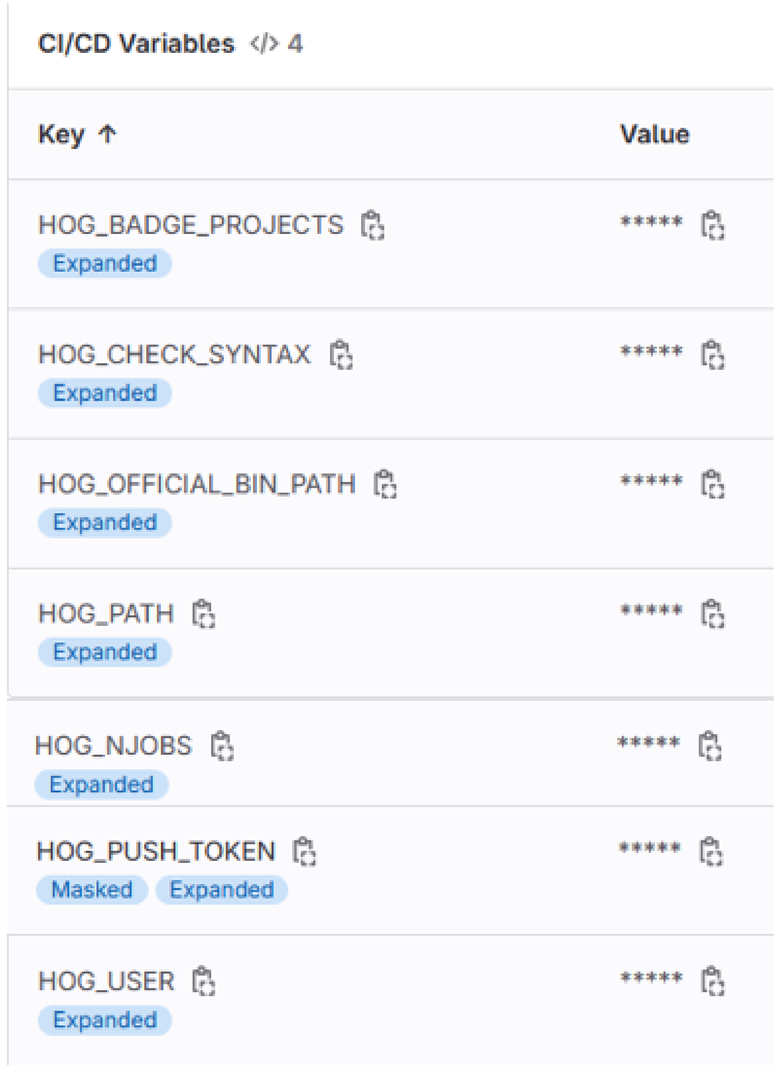
N. Aranzabal (ESRF, Grenoble) on behalf of the Hog group

13 June 2024 - 1st FPGA Developers Forum (FDF)

CUSTOMISING HOG CI

Hog CI will work with default configuration, but can be customized for each project

- You can add **custom jobs** that run before and after Hog jobs
- Configuration via **variables** from GitLab/GitHub web interface
- Additional optional features include:
  - Automatic GitLab/GitHub **releases**
  - **Archive releases** to EOS website or custom paths
  - Automatic **changelog** parsed from git commit messages (use FEATURE: keyword)
  - Automatic **syntax check** before running synthesis
  - Run CI **only for projects that were modified** wrt last official version



USING HOG WITH QUARTUS

CREATE PROJECT

Use the Do script with CREATE option to create Quartus project

USE THE GUI

Developing can be done using Quartus GUI in project mode

USE THE SHELL SCRIPTS

Run Hog workflow in batch mode

INTEGRATED HOG SCRIPTS

Running at pre-synthesis, pre-implementation, post-implementation and post-bitstream stages. Embed git SHA and version, and create reports

! ADD NEW FILES / CHANGE SETTINGS !

New files shall be added to list files and settings to the hog.conf. For Quartus this process is not automatised

! VERSIONING !

At **pre-flow** 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