UNIVERSITY OF BIRMINGHAM

Hog, HDL on git

Francesco Gonnella - University of Birmingham

On behalf of the **Hog group**

Hog Tutorial – 15th June 2021

Rationale

Why and how do we use git for a HDL repository?

- Guarantee HDL synthesis with P&R **reproducibility**
  - Absolute control of **HDL files**, **constraint** files, Vivado/Quartus/ISE **settings**
  - Produced binary files are **the same*** **(Fixed in vivado2020.2!)**
    - Quartus and ISE's behaviour is under investigation
- Assure **traceability** of binary files:
  - **Embed Git SHA** into firmware registers automatically
  - **Version number** evaluated automatically and **embedded in firmware**
- Help developers to:
  - Use **Vivado/Quartus/ISE normally**
  - Use **git normally** (not through external scripts)
  - Do not add **overhead work** (no downloads, no installations)
    - **No additional requirements:** no python, no libraries, no scripts to source
    - Have **Vivado/Quartus/ISE** in your PATH? That's enough!
    - Have **vsim** configured and in your PATH? It also simulates

*Are binary files really the same?

- 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  00363322f
0000160  32090064  38343a31  003135 3a  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  00373322f
0000160  31090064  39323a35  003933 3a  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
~
```

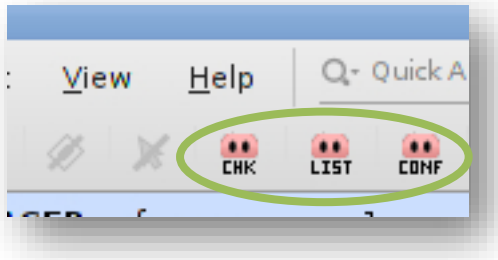
("vimdiff" on "od –t x4" output)

What is Hog?

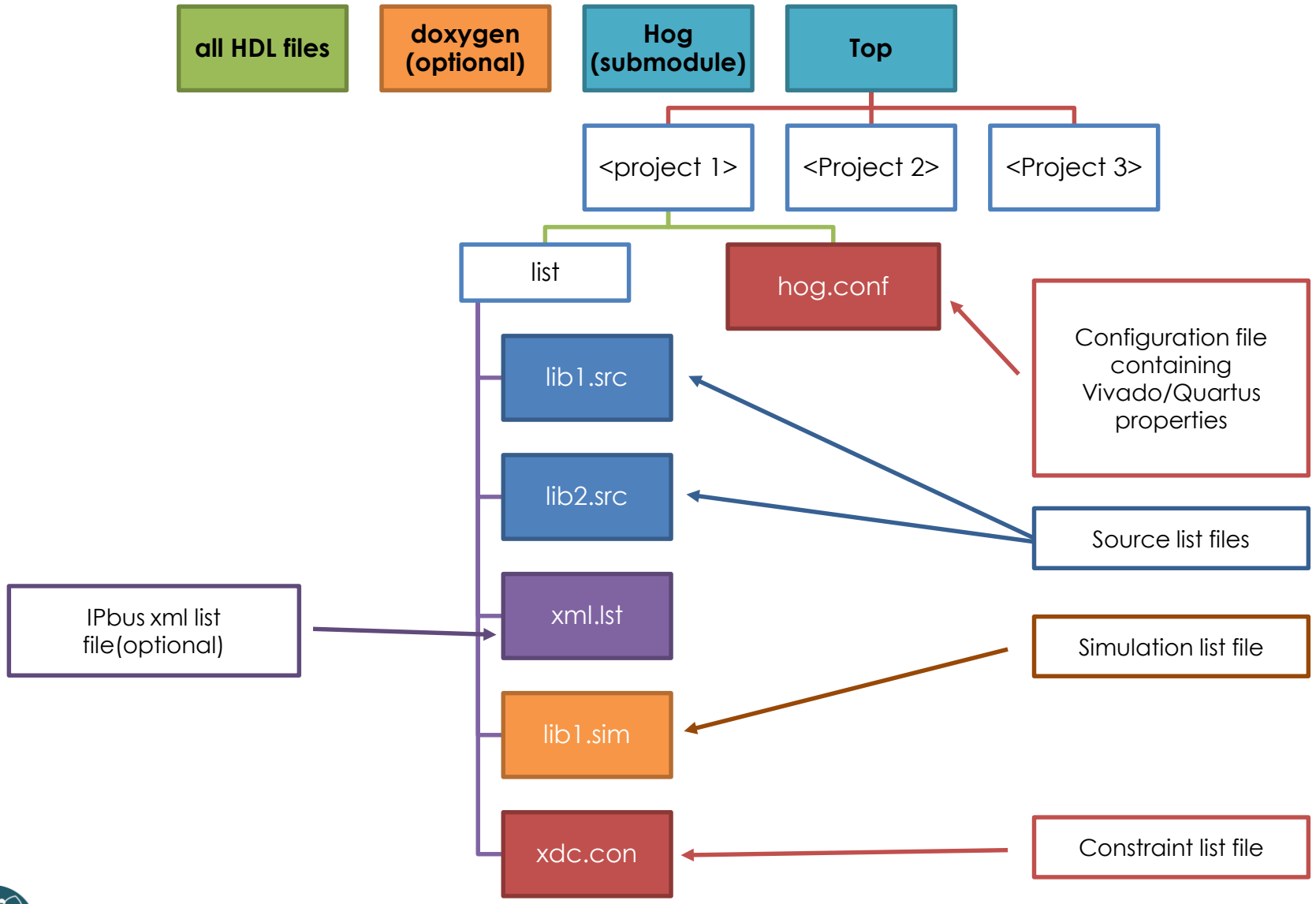
- **Hog** (HDL on git) is a set of **Tcl scripts** (less than 1MB) plus a **methodology**
  - All the functions used are available in **Vivado/Quartus Tcl shells**
  - **Hog** must be included to the repository as a **submodule**
    - Update it **when you want**
    - **different Hog versions** can be used for different projects
- Special folder **Top** containing **list files** (<repository>/**Top**/<project name>/**list**/)
  - Text file for Vivado/Quartus properties (<repository>/**Top**/<project name>/**hog.conf**)
  - HDL **source files** can be stored **anywhere** in the repository, even **submodules**
- **Create project** script (Creates Xilinx/Intel projects from list files)
- Scripts **integrated in Xilinx/Intel** flow guarantee **traceability and reproducibility**:
  - **Pre-synthesis** (check repository against project, feed HDL generics, …)
  - **Post-implementation** (embed git SHA into binary file)
  - **Post write bitstream** (rename and copy files in bin directory, …)
- git flow: **short-lived feature branches, no new-commit** on merge (preserve SHA)
- For the developers, **no additional downloads** or installations are required:
  1. **Clone** the repository (and update the submodules)
  2. **Launch** Hog script
  3. **Start** developing firmware in Vivado

Use Vivado/Quartus normally

- Developing can be done using Quartus or Vivado **GUI** in **project mode**
  - We also provide **optional shell scripts** to run the workflow in batch mode
- Developers start the synthesis and implementation by **clicking** the Vivado buttons normally
- This will trigger the Hog scripts for **pre-synthesis**, pre-implementation, post-implementation and **post-bitstream**
- Hog scripts are **embedded** into the project automatically when the project is created (via CreateProject.sh script)
- Use **Vivado in project-mode normally**, with few exceptions:
  - Do not add new files to the project, but **add file names to the list files** and **re-create the project**
    - Alternatively you can add the files in the GUI and then update the list files using the **list** "**Hog button**"
  - Add Vivado/Quartus **properties** to **hog.conf** file or use the **conf** "**Hog button**"
  - Create **out-of-context IPs** and store files (**.xci, .ip**) into the repository
  - Connect a set of **generics/parameters** with versions and git SHA in the top file
  - A third button can be used to **check** if the **hog.conf** file and **the list files** are up to date



Hog and Top directories

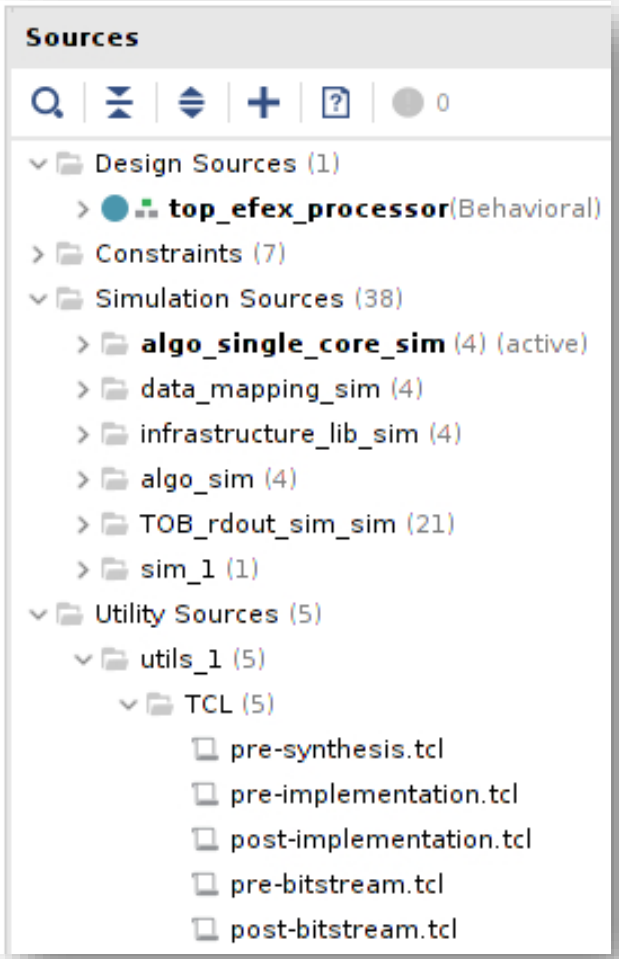


An example of libraries in Vivado

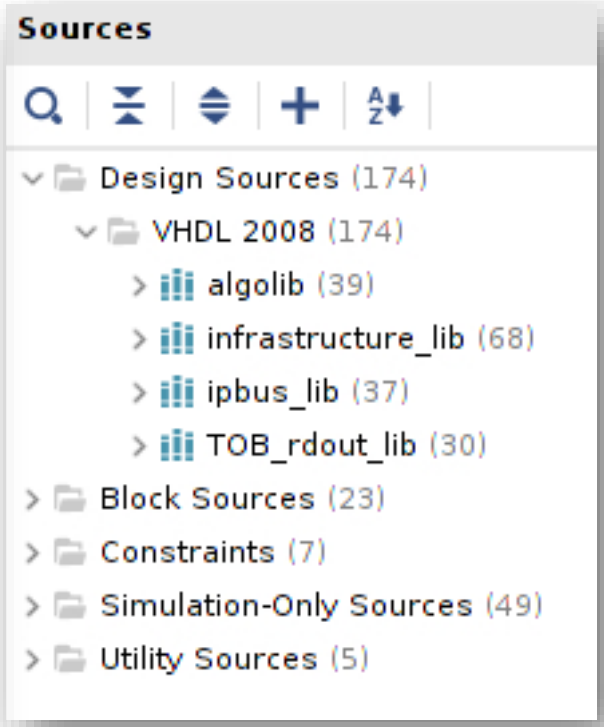
File tree

```
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 sets



Libraries



An example of versions

- This is the output of **Hog CI**
- Date, time, SHAs and versions are evaluated
- Version register are formatted in hex as **MMmmpppp**

```
------------------------ PRE SYNTHESIS ------------------------
08/06/2021 at 18:01:26
Firmware date and time: '08062021', '00165403'
Project flavour: 1
Global SHA: 38a94e7e, VER: 0.8.9
Constraints SHA: B22879A2, VER: 0.8.3
IPbus XML SHA: F48954BF, VER: 0.8.8
Top SHA: 9290074D, VER: 0.8.6
Hog SHA: 189A2AF, VER: 4.16.0
--- Libraries ---
algolib SHA: DAEC54A5, VER: 0.8.8
ipbus_lib SHA: B1F44220, VER: 0.8.5
infrastructure_lib SHA: 46E9CB6C, VER: 0.8.7
TOB_rdout_lib SHA: 10209E86, VER: 0.8.6
--- External Libraries ---
---------------------------------------------------------------
```

Hog uses the commit time and date to guarantee reproducibility

Project "Flavour"

- **Code duplication** should be reduced to the **minimum**, possibly zero
- In case multiple **designs share most of the code** (e.g. different chips on the same board), it is difficult to keep the duplication to zero
  - The most tricky thing is to use the **same top HDL** file in different projects
- To address this problem Hog gives the possibility to specify a project flavour: an **integer number** to distinguish projects sharing the same top HDL file
- If you want to use the Hog "flavour", just add a numeric extension (e.g. .1, .2, .10, .0) to the project folder name: e.g. ***Top/my_fpga.1***
- In this case the top file and the top module must be called ***top_my_fpga*** and not top_my_fpga.1
  - Normally Hog requires the top file and the top module be called top_<project name> and the top file top_<project name>.vhdl (or .v)
- The integer number is passed to the top module as a **generic** called **FLAVOUR**
- The developers can use it in **if generate** or in functions to create different projects as a function of the flavour

Commit before building!

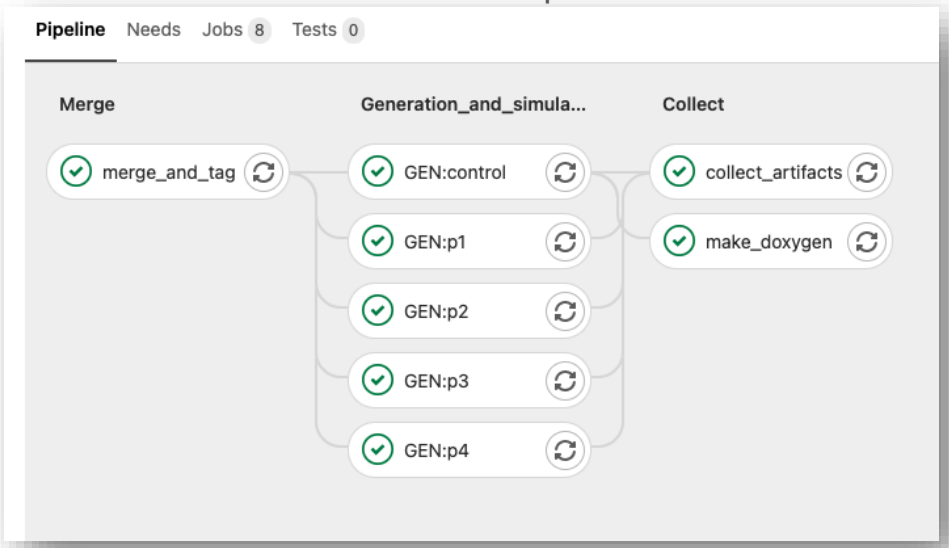
You should **always** commit (even if it's just a test)… but what if you don't?

- What if you have **uncommitted modifications**?
  - If your repository is "dirty" when you launch the synthesis, Hog's **pre-synthesis** script will generate a Critical warning and **produce diff files**
  - The version in the binary file will be set to **zero**
- What if I **add a file** from the **GUI**?
  - The **pre-synthesis** script compares the files in the project with the files in the list files and give a Critical Warning if they don't match
- What if I **change a property** in the Vivado project from the GUI?
  - Again, the **pre-synthesis** script will find out…
- Mh… What if I remove Hog's **pre-synthesis** script from Vivado?
  - Well, really? The versions and SHAs wont' be fed to parameters/generics

However, you can **add files using the GUI** or modify properties and then click the **Hog buttons** to automatically modify the **list files** or the **hog.conf**

Hog's Continuous Integration

- Use **private runners** on any machine with a Gitlab **runner** installed
  - Need Vivado and Questasim installations not available on public runners
- Triggered by **Merge Request**
- Launch Vivado/Quartus in **batch mode**
  - Use Hog **launcher scripts**
- **Automatically extracts Version from Git tag vM.m.p**
- increases **m, M** or **p** according to keywords contained in MR description



- Collects binary files, reports, bitfiles, ipbus XMLs **from all projects** as artefacts
- When branch is **merged:**
  - Tag official version (optionally run Doxygen)
  - Makes a **Gitlab Release** containing archives of all produced files
  - (optionally) store files on CERN **EOS cloud storage (web accessible)**

How to set it up?

- Create a **simple .gitlab-ci.yml** file
  - Example from Hog templates
- **Include** the Hog yml files
- Write **few lines** for each project
  - Alternatively the **dynamic CI** can be used, where the job are created dynamically
- This will **enable Hog-CI** on a machine with a Gitlab Runner and Vivado/Quartus
- **Instructions** and **scripts** to setup the runners can be found at: gitlab.cern.ch/hog/vm-setup

Normal CI

```
15 include:
16     - project: 'hog/Hog'
17       file: '/hog.yml'
18       ref: 'v0.2.1'
19
20 ################### example ##########################
21 ### Change 'example' with your project name
22
23 generate_project:example:
24     extends: .generate_project
25     variables:
26       extends: .vars
27       PROJECT_NAME: example
28       HOG_ONLY_SYNTH: 0 # if 1 runs only the synthesis
29
30 simulate_project:example:
31     extends: .simulate_project
32     variables:
33       extends: .vars
34       PROJECT_NAME: example
```

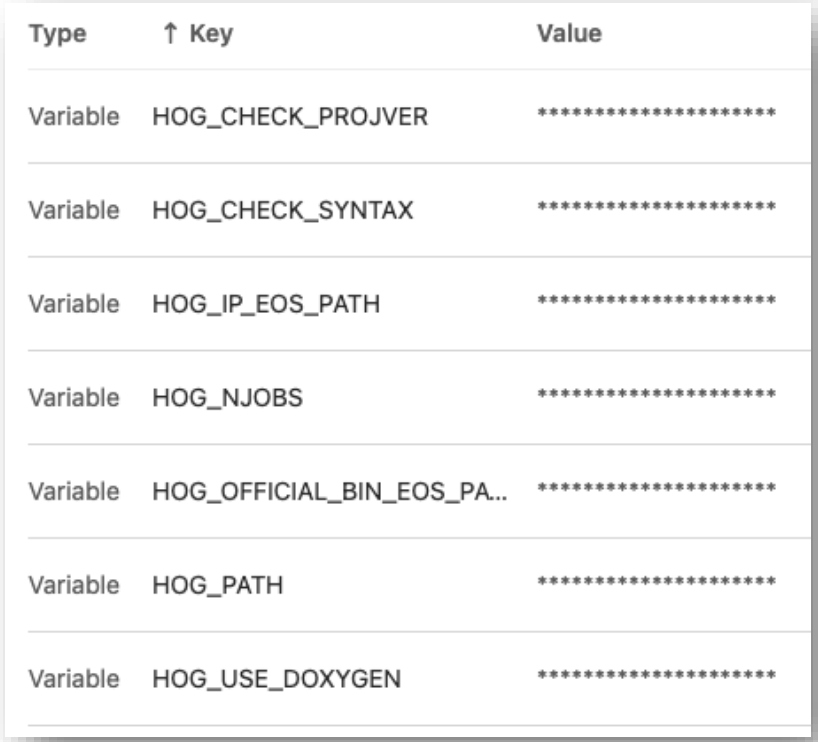
Dynamic CI

```
15 include:
16     - project: 'hog/Hog'
17       file: '/hog-dynamic.yml'
18       ref: 'v0.2.1'
```

Customising Hog Gitlab CI

Hog CI will work with **default configuration**, **but** you might want to **tailor it to your project**

- You can add **custom jobs** that run before and after Hog jobs
- Configure via **variables** in Gitlab website
- Additional optional features include:
  - Automatic Gitlab **releases**
  - **Archive of binary files** to EOS website
  - Automatic **changelog** parsed from git messages (use **FEATURE:** keyword)
  - Automatic **syntax check** before running synthesis
  - Run CI **only for projects that were modified** wrt last official version
  - Run **Doxygen** (use **DOXYGEN_ONLY**, to avoid running the workflow)



Type	↑ Key	Value
Variable	HOG_CHECK_PROJVER	**********
Variable	HOG_CHECK_SYNTAX	**********
Variable	HOG_IP_EOS_PATH	**********
Variable	HOG_NJOBS	**********
Variable	HOG_OFFICIAL_BIN_EOS_PA...	**********
Variable	HOG_PATH	**********
Variable	HOG_USE_DOXYGEN	**********

Gitlab releases

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	
Lib: fmc0	5b6ce4fb	
Lib: ipbus	8e5214ea	

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**.

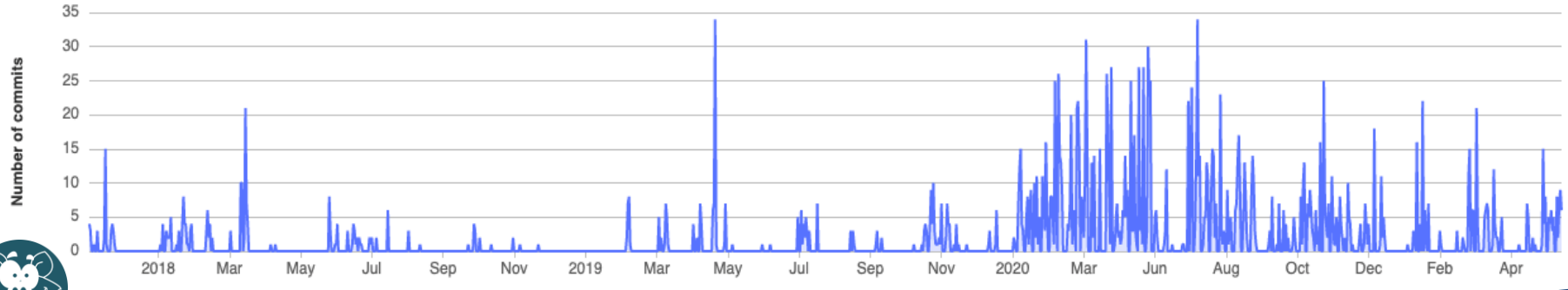
a8f0e110 v0.1.1 Created 3 weeks ago by

Conclusions

- Hog is available at gitlab.cern.ch/hog/Hog
  - Active project, involving **5 developers**
  - Released **twice a year** (usually January and June) under **Apache 2** licence
  - **Experimental features** are available in the **develop** branch
  - Last release **Hog2021.2** (8 June 2021)
- **Documentation**: cern.ch/hog, **support:** hog-group@cern.ch
- By-product of using Tcl scripts only: **Hog works on Windows!** (with git bash)
- Hog is **used by several projects**, including: **ATLAS** and **CMS** Phase-I and Phase-II upgrades, GAPS, FOOT

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
```



UNIVERSITY OF BIRMINGHAM

Thanks for your attention

Francesco Gonnella

Other Hog's features

- **Simulation** sets run by the **CI**
- **Automatic checking of IPbus XMLs** against VHDL address maps
- Automatic **IPbus** VHDL address map **generation locally**
  - Using official IPbus python script (requires installation)
- Automatic **documentation** with Doxygen
- **Customisable main branch**, default "master"
- Support: **Questasim**, **Modelsim**, **Vivado Simulator**, **Riviera**
- Hog creates **Sigasi** project csv files

Integrated Hog scripts

- Pre **synthesis**
  - Check **repository status** (Critical Warning if not clean)
  - Calculate **versions and SHAs** and feed them as generic
  - Checks yml file (optional)
  - Checks ipbus address maps (optional)
  - Produce a **version.txt** file, containing all the versions and SHA for all the libraries
  - Copy ipbus XMLs (optional)
- Post **write bitstream**
  - Copy the bit and bin file to a **bin folder** in renaming it with **git describe**
  - Copy the **ipbus xml files** (replacing place holders with git SHA and version)
  - Copy all report files and log files
  - Copy **version.txt** file
  - Copy a timing recap .txt file, containing TNS and WNS (CI only)
    - This file's name is **timing_ok.txt** if there is no violation and **timing_error.txt** if timing requirements are not met
- **Pre-** and **post-implementation**, **pre-bitsream** are also used
  - E.g. to embed the **git SHA** in the **binary file** (in case the file gets renamed)

List files, submodules: libraries

- The files included in a project are specified inside **text files** (called list files) located in a specific directory in the repository
  - There are **source** list files (.src), **simulation** list files (.sim) and **constraint** list files (.con)
  - Another feature was added to handle external proprietary libraries (.ext)
  - List files are **handled recursively**: a list file can include another list file. Very useful in combination with the project *flavour* feature
- Every project corresponds to a directory that can contain **multiple list files**, for each one **Hog** will create a **VHDL library**
  - 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 using **third-party HDL** in a project (e.g. ipbus) or if part of the code is shared among many Git repositories, it is convenient to use a Git **submodule**
  - Submodules are a Git feature allowing to include independent repositories into a repository
  - They can be very handy but are a bit more complicated to use, and should not be used if not necessary
- Hog **supports Git submodules** everywhere in the repository and containing every kind of files: IP, HDL,. constraints
- If not interested in using libraries, developers can create **just one list file**

Version and Git SHA

- In order to guarantee binary file **traceability**, **Hog** embeds the repository Git-SHA/Version into firmware registers
  - To easily compare different firmware versions, Hog calculates the **version of each firmware library** and feeds them as generic/parameters
- Git can evaluate SHAs **independently for any given subset of files:**
  - Given 2 SHAs, it is impossible to tell which is more recent without looking at the repository
  - For this reason **a numeric version** M.m.p is evaluated to indicate how recent each library is, as explained in the following slides
- **M, m** and **p** are **automatically** extracted **from Git tags** and **fed to firmware** via *VHDL* generics at synthesis time
- Releases tags are of the form: **v<M>.<m>.<p>**
- Official versions are **automatically tagged** (by the Hog-CI), not to rely on developers to increase numbers manually in VHDL files

Handling IPs

- Projects contain **Intellectual Properties** (IP)
  - FIFO, RAM, MGT, etc.
- Each made of **multiple files** (VHDL, Verilog) contained in a directory
- Only the main file is committed to the repository: **xci** (for Xilinx)
  - These are **text file** that can be handled by git
- **All the other files are generated** by Vivado at synthesis/simulation time and must be ignored by Git
  - A template tells the developer how to properly set the **.gitignore**
- For **Intel (Altera) IPs** the same logic applies
  - QSYS, IP, and QIP files are handled
- Optionally, the CI can **speed up IP synthesis** by archiving them on EOS and **copying them over** rather than synthesising them again

Automatic version number M.m.p

- Hog CI extracts the values of **M, m, p from Git tags**
  - Hog then feeds these to the firmware registers via VHDL generics at synthesis time
- In order to do this, it must **know the new firmware version a priori,** before starting the synthesis, and before accepting the merge request
- **How to do this?**
  - Extract the **current version number from the most recent tag** describing the current commit and:
    - Increases p by default
    - To increase **M** or **m** a **"candidate" tag** must be created (**b**x**-v**M.m.p)
    - x is the Git merge request number (to avoid duplicated tags)
    - These tags are automatically created by Hog-CI