The PoC-Library Documentation

Release 1.1.3

Patrick Lehmann, Thomas B. Preusser, Martin Zabel

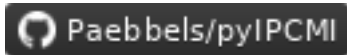**Oct 13, 2019**

Part I

Introduction

This Python package is maintained by Patrick Lehmann.|br| https://Paebbels.GitHub.io/

PoC - "Pile of Cores" provides implementations for often required hardware functions such as Arithmetic Units, Caches, Clock-Domain-Crossing Circuits, FIFOs, RAM wrappers, and I/O Controllers. The hardware modules are typically provided as VHDL or Verilog source code, so it can be easily re-used in a variety of hardware designs.

All hardware modules use a common set of VHDL packages to share new VHDL types, sub-programs and constants. Additionally, a set of simulation helper packages eases the writing of testbenches. Because PoC hosts a huge amount of IP cores, all cores are grouped into sub-namespaces to build a better hierachy.

Various simulation and synthesis tool chains are supported to interoperate with PoC. To generalize all supported free and commercial vendor tool chains, PoC is shipped with a Python based infrastructure to offer a command line based frontend.

News

See *Change Log* for latest updates.

Cite the PoC-Library

The PoC-Library hosted at GitHub.com. Please use the following biblatex entry to cite us:

```
# BibLaTex example entry
@online{poc,
  title={{PoC - Pile of Cores}},
  author={{Chair of VLSI Design, Diagnostics and Architecture}},
  organization={{Technische Universität Dresden}},
  year={2016},
  url={https://github.com/VLSI-EDA/PoC},
  urldate={2016-10-28},
}
```

CHAPTER 1

What is pyIPCMI?

PoC - "Pile of Cores" provides implementations for often required hardware functions such as Arithmetic Units, Caches, Clock-Domain-Crossing Circuits, FIFOs, RAM wrappers, and I/O Controllers. The hardware modules are typically provided as VHDL or Verilog source code, so it can be easily re-used in a variety of hardware designs.

All hardware modules use a common set of VHDL packages to share new VHDL types, sub-programs and constants. Additionally, a set of simulation helper packages eases the writing of testbenches. Because PoC hosts a huge amount of IP cores, all cores are grouped into sub-namespaces to build a better hierachy.

Various simulation and synthesis tool chains are supported to interoperate with PoC. To generalize all supported free and commercial vendor tool chains, PoC is shipped with a Python based Infrastruture to offer a command line based frontend.

The PoC-Library pursues the following five goals:

- independence in the platform, target, vendor and tool chain
- generic, efficient, resource sparing and fast implementations of IP cores
- optimized for several device architectures, if suitable
- supportive scripts to ease the IP core handling with all supported vendor tools on all listed operating systems
- ship all IP cores with testbenches for local and online verification

In detail the PoC-Library is:

- synthesizable for ASIC and FPGA devices, e.g. from Altera, Lattice, Xilinx, . . . ,
- supports a wide range of simulation and synthesis tool chains, and is
- executable on several host platforms: Darwin, Linux or Windows.

This is achieved by using generic HDL descriptions, which work with most synthesis and simulation tools mentioned above. If this is not the case, then PoC uses vendor or tool dependent work-arounds. These work-arounds can be different implementations switched by VHDL *generate* statements as well as different source files containing modified implementations.

One special feature of PoC is it, that the user has not to take care of such implementation switchings. PoC's IP cores decide on their own what's the *best* implementation for the chosen target platform. For this feature, PoC implements a configuration package, which accepts a well-known development board name or a target device

string. For example a FPGA device string is decoded into: vendor, device, generation, family, subtype, speed grade, pin count, etc. Out of these information, the PoC component can for example implement a vendor specific carry-chain description to speed up an algorithm or group computation units to effectively use 6-input LUTs.

1.1 What is the History of PoC?

In the past years, a lot of "IP cores" were developed at the chair of VLSI design[1] . This lose set of HDL designs was gathered in an old-fashioned CVS repository and grow over the years to a collection of basic HDL implementations like ALUs, FIFOs, UARTs or RAM controllers. For their final projects (bachelor, master, diploma thesis) students got access to PoC, so they could focus more on their main tasks than wasting time in developing and testing basic IP implementations from scratch. But the library was initially for internal and educational use only.

As a university chair for VLSI design, we have a wide range of different FPGA prototyping boards from various vendors and device families as well as generations. So most of the IP cores were developed for both major FPGA vendor platforms and their specific vendor tool chains. The main focus was to describe hardware in a more flexible and generic way, so that an IP core could be reused on multiple target platforms.

As the number of cores increased, the set of common functions and types increased too. In the end PoC is not only a collection of IP cores, its also shipped with a set of packages containing utility functions, new types and type conversions, which are used by most of the cores. This makes PoC a *library*, not only a *collection* of IPs.

As we started to search for ways to publish IP cores and maybe the whole PoC-Library, we found several platforms on the Internet, but none was very convincing. Some collective websites contained inactive projects, others were controlled by companies without the possibility to contribute and the majority was a long list of private projects with at most a handful of IP cores. Another disagreement were the used license types for these projects. We decided to use the Apache License, because it has no copyleft rule, a patent clause and allows commercial usage.

We transformed the old CVS repository into three Git repositories: An internal repository for the full set of IP cores (incl. classified code), a public one and a repository for examples, called PoC-Examples, both hosted on GitHub. PoC itself can be integrated into other HDL projects as a library directory or a Git submodule. The preferred usage is the submodule integration, which has the advantage of linked repository versions from hosting Git and the submodule Git. This is already exemplified by our PoC-Examples repository.

1.2 Which Tool Chains are supported?

The PoC-Library and its Python-based infrastructure currently supports the following free and commercial vendor tool chains:

- Synthesis Tool Chains:
    - **Altera Quartus** Tested with Quartus-II $\geq$ 13.0. Tested with Quartus Prime $\geq$ 15.1.
    - **Intel Quartus** Tested with Quartus Prime $\geq$ 16.1.
    - **Lattice Diamond** Tested with Diamond $\geq$ 3.6.
    - **Xilinx ISE** Only ISE 14.7 inclusive Core Generator 14.7 is supported.
    - **Xilinx PlanAhead** Only PlanAhead 14.7 is supported.
    - **Xilinx Vivado** Tested with Vivado $\geq$ 2015.4. Due to a limited VHDL language support compared to ISE 14.7, some PoC IP cores need special work arounds. See the synthesis documentation section for Vivado for more details.
- Simulation Tool Chains:

---

[1] The PoC-Library is published and maintained by the **Chair for VLSI Design, Diagnostics and Architecture** - Faculty of Computer Science, Technische Universität Dresden, Germany http://tu-dresden.de/inf/vlsi-eda

- **Aldec Active-HDL** Tested with Active-HDL (or Student-Edition) $\geq$ 10.3 Tested with Active-HDL Lattice Edition $\geq$ 10.2

- **Cocotb with Mentor QuestaSim backend** Tested with Mentor QuestaSim 10.4d

- **Mentor Graphics ModelSim** Tested with ModelSim PE (or Student Edition) $\geq$ 10.5c Tested with ModelSim SE $\geq$ 10.5c Tested with ModelSim Altera Edition 10.3d (or Starter Edition)

- **Mentor Graphics QuestaSim/ModelSim** Tested with Mentor QuestaSim $\geq$ 10.4d

- **Xilinx ISE Simulator** Tested with ISE Simulator (iSim) 14.7. The Python infrastructure supports isim, but PoC's simulation helper packages and testbenches rely on VHDL-2008 features, which are not supported by isim.

- **Xilinx Vivado Simulator** Tested with Vivado Simulator (xsim) $\geq$ 2016.3. The Python infrastructure supports xsim, but PoC's simulation helper packages and testbenches rely on VHDL-2008 features, which are not fully supported by xsim, yet.

- **GHDL + GTKWave** Tested with GHDL $\geq$ 0.34dev and GTKWave $\geq$ 3.3.70 Due to ungoing development and bugfixes, we encourage to use the newest GHDL version.

1.3 Why should I use PoC?

Here is a brief list of advantages:

- We explicitly use the wording *PoC-Library* rather then *collection*, because PoC's packages and IP cores build an ecosystem. Complex IP cores are build on-top of basic IP cores - they are no lose set of cores. The cores offer a clean interface and can be configured by many generic parameters.

- PoC is target independent: It's possible to switch the target device or even the device vendor without switching the IP core.

Todo: Use a well tested set of packages to ease the use of VHDL

Use a well tested set of simulation helpers

Run testbenches in various simulators.

Run synthesis tests in varous synthesis tools.

Compare hardware usage for different target platfroms.

Supports simulation with vendor primitive libraries, ships with script to pre-compile vendor libraries.

Vendor tools have bugs, check you IP cores when a new tool release is available, before changing code base

---

1.4 Who uses PoC?

PoC has a related Git repository called PoC-Examples on GitHub. This repository hosts a list of example and reference implementations of the PoC-Library. Additional to reading an IP cores documention and viewing its characteristic stimulus waveform in a simulation, it can helper to investigate an IP core usage example from that repository.

- The Q27 Project 27-Queens Puzzle: Massively Parellel Enumeration and Solution Counting

- Reconfigurable Cloud Computing Framework (RC2F) An FPGA computing framework for virtualization and cloud integration.

- PicoBlaze-Library The PicoBlaze-Library offers several PicoBlaze devices and code routines to extend a common PicoBlaze environment to a little System on a Chip (SoC or SoFPGA).

- PicoBlaze-Examples A SoFPGA reference implementation, based on the PoC-Library and the PicoBlaze-Library.

Quick Start Guide

This **Quick Start Guide** gives a fast and simple introduction into PoC. All topics can be found in the *Using PoC* section with much more details and examples.

Contents of this Page

- *Requirements and Dependencies*
- *Download*
- *Configuring PoC on a Local System*
- *Integration*
- *Run a Simulation*
- *Run a Synthesis*
- *Updating*

2.1 Requirements and Dependencies

The PoC-Library comes with some scripts to ease most of the common tasks, like running testbenches or generating IP cores. PoC uses Python 3 as a platform independent scripting environment. All Python scripts are wrapped in Bash or PowerShell scripts, to hide some platform specifics of Darwin, Linux or Windows. See *Requirements* for further details.

PoC requires:

- A *supported synthesis tool chain*, if you want to synthezise IP cores.
- A *supported simulator too chain*, if you want to simulate IP cores.
- The **Python 3** programming language and runtime, if you want to use PoC's infrastructure.
- A shell to execute shell scripts:
    - **Bash** on Linux and OS X

– **PowerShell** on Windows

PoC optionally requires:

- **Git** command line tools or
- **Git User Interface**, if you want to check out the latest 'master' or 'release' branch.

PoC depends on third part libraries:

- THIRD:Cocotb ⭘ A coroutine based cosimulation library for writing VHDL and Verilog testbenches in Python.
- THIRD:OSVVM ⭘ Open Source VHDL Verification Methodology.
- THIRD:UVVM ⭘ Universal VHDL Verification Methodology.
- THIRD:VUnit ⭘ An unit testing framework for VHDL.

All dependencies are available as GitHub repositories and are linked to PoC as Git submodules into the PoC-Root\lib directory. See *Third Party Libraries* for more details on these libraries.

2.2 Download

The PoC-Library can be downloaded as a zip-file (latest 'master' branch), cloned with `git clone` or embedded with `git submodule add` from GitHub. GitHub offers HTTPS and SSH as transfer protocols. See the *Download* page for further details. The installation directory is referred to as `PoCRoot`.

Protocol	Git Clone Command
HTTPS	`git clone --recursive https://github.com/VLSI-EDA/PoC.git PoC`
SSH	`git clone --recursive ssh://git@github.com:VLSI-EDA/PoC.git PoC`

2.3 Configuring PoC on a Local System

To explore PoC's full potential, it's required to configure some paths and synthesis or simulation tool chains. The following commands start a guided configuration process. Please follow the instructions on screen. It's possible to relaunch the process at any time, for example to register new tools or to update tool versions. See *Configuration* for more details. Run the following command line instructions to configure PoC on your local system:

```
cd PoCRoot
.\poc.ps1 configure
```

Use the keyboard buttons: to accept, to decline, to skip/pass a step and to accept a default value displayed in brackets.

2.4 Integration

The PoC-Library is meant to be integrated into other HDL projects. Therefore it's recommended to create a library folder and add the PoC-Library as a Git submodule. After the repository linking is done, some short configuration steps are required to setup paths, tool chains and the target platform. The following command line instructions show a short example on how to integrate PoC.

1. Adding the Library as a Git submodule

The following command line instructions will create the folder `lib\PoC\` and clone the PoC-Library as a Git submodule into that folder. `ProjectRoot` is the directory of the hosting Git. A detailed list of steps can be found at Integration.

```
cd ProjectRoot
mkdir lib | cd
git submodule add https://github.com:VLSI-EDA/PoC.git PoC
cd PoC
git remote rename origin github
cd ..\..
git add .gitmodules lib\PoC
git commit -m "Added new git submodule PoC in 'lib\PoC' (PoC-Library)."
```

2. Configuring PoC

The PoC-Library should be configured to explore its full potential. See Configuration for more details. The following command lines will start the configuration process:

```
cd ProjectRoot
.\lib\PoC\poc.ps1 configure
```

3. Creating PoC's `my_config.vhdl` and `my_project.vhdl` Files

The PoC-Library needs two VHDL files for its configuration. These files are used to determine the most suitable implementation depending on the provided target information. Copy the following two template files into your project's source folder. Rename these files to *.vhdl and configure the VHDL constants in the files:

```
cd ProjectRoot
cp lib\PoC\src\common\my_config.vhdl.template src\common\my_config.vhdl
cp lib\PoC\src\common\my_project.vhdl.template src\common\my_project.vhdl
```

my_config.vhdl defines two global constants, which need to be adjusted:

```
constant MY_BOARD            : string := "CHANGE THIS"; -- e.g. Custom, ML505,
→KC705, Atlys
constant MY_DEVICE           : string := "CHANGE THIS"; -- e.g. None, XC5VLX50T-
→1FF1136, EP2SGX90FF1508C3
```

my_project.vhdl also defines two global constants, which need to be adjusted:

```
constant MY_PROJECT_DIR    : string := "CHANGE THIS"; -- e.g. d:/vhdl/myproject/,
→ /home/me/projects/myproject/"
constant MY_OPERATING_SYSTEM : string := "CHANGE THIS"; -- e.g. WINDOWS, LINUX
```

Further informations are provided at Creating my_config/my_project.vhdl.

4. Adding PoC's Common Packages to a Synthesis or Simulation Project

PoC is shipped with a set of common packages, which are used by most of its modules. These packages are stored in the `PoCRoot\src\common` directory. PoC also provides a VHDL context in `common.vhdl`, which can be used to reference all packages at once.

5. Adding PoC's Simulation Packages to a Simulation Project

Simulation projects additionally require PoC's simulation helper packages, which are located in the `PoCRoot\src\sim` directory. Because some VHDL version are incompatible among each other, PoC uses version suffixes like `*.v93.vhdl` or `*.v08.vhdl` in the file name to denote the supported VHDL version of a file.

6. Compiling Shipped IP Cores

Some IP Cores are shipped are pre-configured vendor IP Cores. If such IP cores shall be used in a HDL project, it's recommended to use PoC to create, compile and if needed patch these IP cores. See Synthesis for more details.

2.5 Run a Simulation

The following quick example uses the GHDL Simulator to analyze, elaborate and simulate a testbench for the module `arith_prng` (Pseudo Random Number Generator - PRNG). The VHDL file `arith_prng.vhdl` is located at `PoCRoot\src\arith` and virtually a member in the *PoC.arith* namespace. So the module can be identified by an unique name: `PoC.arith.prng`, which is passed to the frontend script.

Example:

```
cd PoCRoot
.\poc.ps1 ghdl PoC.arith.prng
```

The CLI command `ghdl` chooses *GHDL Simulator* as the simulator and passes the fully qualified PoC entity name `PoC.arith.prng` as a parameter to the tool. All required source file are gathered and compiled to an executable. Afterwards this executable is launched in CLI mode and its outputs are displayed in console:

Each testbench uses PoC's simulation helper packages to count asserts and to track active stimuli and checker processes. After a completed simulation run, an report is written to STDOUT or the simulator's console. Note the

line `SIMULATION RESULT = PASSED`. For each simulated PoC entity, a line in the overall report is created. It lists the runtime per testbench and the simulation status (`...  ERROR`, `FAILED`, `NO ASSERTS` or `PASSED`). See Simulation for more details.

2.6 Run a Synthesis

The following quick example uses the Xilinx Systesis Tool (XST) to synthesize a netlist for IP core `arith_prng` (Pseudo Random Number Generator - PRNG). The VHDL file `arith_prng.vhdl` is located at `PoCRoot\src\arith` and virtually a member in the *PoC.arith* namespace. So the module can be identified by an unique name: `PoC.arith.prng`, which is passed to the frontend script.

Example:

```
cd PoCRoot
.\poc.ps1 xst PoC.arith.prng --board=KC705
```

The CLI command `xst` chooses *Xilinx Synthesis Tool* as the synthesizer and passes the fully qualified PoC entity name `PoC.arith.prng` as a parameter to the tool. Additionally, the development board name is required to load the correct `my_config.vhdl` file. All required source file are gathered and synthesized to a netlist.

2.7 Updating

The PoC-Library can be updated by using `git fetch` and `git merge`.

```
cd PoCRoot
# update the local repository
git fetch --prune
# review the commit tree and messages, using the 'treea' alias
git treea
# if all changes are OK, do a fast-forward merge
git merge
```

See also:

Running one or more testbenches The installation can be checked by running one or more of PoC's testbenches.

Running one or more netlist generation flows The installation can also be checked by running one or more of PoC's synthesis flows.

Get Involved

A first step might be to use and explore PoC and it's infrastructure in an own project. Moreover, we encurage to read our online help which covers all aspects from quickstart example up to detailed IP core documentation. While using PoC, you might discover issues or missing feature. Please report them as *listed below*. If you have an interresting project, please send us feedback or get listed on our Who uses PoC? page.

If you are more familiar with PoC and it's components, you might start asking youself how components internally work. Please read our more advanced topics in the online help, read our inline source code comments or start a discussion on *Gitter* to ask us directly.

Now you should be very familiar with our work and you might be interessted in developing own components and contribute them to the main repository. See the *next section* for detailed instructions on the Git fork, commit, push and pull-request flow.

PoC ships some *third-party libraries*. If you are interessted in getting your library or components shipped as part of PoC or as a third-party components, please contact us.

3.1 Report a Bug

Please report issues of any kind in our Git provider's issue tracker. This allows us to categorize issues into groups and assign developers to them. You can track the issue's state and see how it's getting solved. All enhancements and feature requests are tracked on GitHub at GitHub Issues.

3.2 Feature Request

Please report missing features of any kind. We are allways looking forward to provide a full feature set. Please use our Git provider's issue tracker to report enhancements and feature requests, so you can track the request's status and implementation. All enhancements and feature requests are tracked on GitHub at GitHub Issues.

3.3 Talk to us on Gitter

You can chat with us on Gitter in our Giiter Room VLSI-EDA/PoC. You can use Gitter for free with your existing GitHub or Twitter account.

3.4 Contributers License Agreement

We require all contributers to sign a Contributor License Agreement (CLA). If you don't know whatfore a CLA is needed and how it prevents legal issues on both sides, read this short blog post. PoC uses the Apache Contributor License Agreement to match the Apache License 2.0.

So to get started, sign the Contributor License Agreement (CLA) at CLAHub.com. You can authenticate yourself with an existing GitHub account.

3.5 Contribute to PoC

Contibuting source code via Git is very easy. We don't provide direct write access to our repositories. Git offers the fork and pull-request philosophy, which means: You clone a repository, provide your changes in your own repository and notify us about outstanding changes via a pull-requests. We will then review your proposed changes and integrate them into our repository.

Steps 1 to 5 are done only once for setting up a forked repository.

3.5.1 1. Fork the PoC Repository

Git repositories can be cloned on a Git provider's server. This procedure is called *forking*. This allows Git providers to track the repository's network, check if repositories are related to each other and notify if pull-requests are available.

Fork our repository `VLSI-EDA/PoC` on GitHub into your or your's Git organisation's account. In the following the forked repository is referenced as `<username>/PoC`.

3.5.2 2. Clone the new Fork

Clone this new fork to your machine. See *Downloading via Git clone* for more details on how to clone PoC. If you have already cloned PoC, then you can setup the new fork as an additional *remote*. You should set `VLSI-EDA/PoC` as fetch target and the new fork `<username>/PoC` as push target.

Shell Commands for Cloning:

```
cd GitRoot
git clone --recursive "ssh://git@github.com:<username>/PoC.git" PoC
cd PoC
git remote rename origin github
git remote add upstream "ssh://git@github.com:VLSI-EDA/PoC.git"
git fetch --prune --tags
```

Shell Commands for Editing an existing Clone:

```
cd PoCRoot
git remote rename github upstream
git remote add github "ssh://git@github.com:<username>/PoC.git"
git fetch --prune --tags
```

These commands work for Git submodules too.

3.5.3 3. Checkout a Branch

Checkout the `master` or `release` branch and maybe stash outstanding changes.

```
cd PoCRoot
git checkout release
```

3.5.4 4. Setup PoC for Developers

Run PoC's *configuration routines* and setup the developer tools.

```
cd PoCRoot
.\PoC.ps1 configure git
```

3.5.5 5. Create your own `master` Branch

Each developer has his own `master` branch. So create one and check it out.

```
cd PoCRoot
git branch <username>/master
git checkout <username>/master
git push github <username>/master
```

If PoC's branches are moving forward, you can update your own master branch by merging changes into your branch.

3.5.6 6. Create your Feature Branch

Each new feature or bugfix is developed on a feature branch. Examples for branch names:

Branch name	Description
bugfix-utils	Fixes a bug in `utils.vhdl`.
docs-spelling	Fixes the documentation.
spi-controller	A new SPI controller implementation.

```
cd PoCRoot
git branch <username>/<feature>
git checkout <username>/<feature>
git push github <username>/<feature>
```

3.5.7 7. Commit and Push Changes

Commit your porposed changes onto your feature branch and push all changes to GitHub.

```
cd PoCRoot
# git add ....
git commit -m "Fixed a bug in function bounds() in utils.vhdl."
git push github <username>/<feature>
```

3.5.8 8. Create a Pull-Request

Go to your forked repository and klick on "Compare and Pull-Request" or go to our PoC repository and create a new pull request.

If this is your first Pull-Request, you need to sign our Contributers License Agreement (CLA).

3.5.9 9. Keep your `master` up-to-date

Todo: undocumented

3.6 Give us Feedback

Please send us feedback about the PoC documentation, our IP cores or your user story on how you use PoC.

3.7 List of Contributers

Contributor[1]	Contact E-Mail
Genßler, Paul	paul.genssler@tu-dresden.de
Köhler, Steffen	steffen.koehler@tu-dresden.de
Lehmann, Patrick[2]	patrick.lehmann@tu-dresden.de; paebbels@gmail.com
Preußer, Thomas B.[2]	thomas.preusser@tu-dresden.de; thomas.preusser@utexas.edu
Reichel, Peter	peter.reichel@eas.iis.fraunhofer.de; peter@peterreichel.info
Schirok, Jan	janschirok@gmx.net
Voß, Jens	jens.voss@mailbox.tu-dresden.de
Zabel, Martin[2]	martin.zabel@tu-dresden.de

---

[1] In alphabetical order.
[2] Maintainer.

Part II

Main Documentation

Using PoC

PoC can be used in several ways, if all *Requirements* are fulfilled. Chose one of the following integration kinds:

- **Stand-Alone IP Core Library:** Download PoC as archive file (*.zip) from GitHub as latest branch copy or as tagged release file. IP cores can be copied into one or more destination projects or the projects link to the selected IP core source files.

    Advantages:

    - Simple and fast setup, configuring PoC is optional.

    - Needs less disk space than a Git repository.

    - After a configuration, PoC's additional features: simulation, synthesis, etc. can be used.

    Disadvantages:

    - Manual updating via download and file overwrites.

    - Updated IP cores need to be copied again into the destination project.

    - Using different PoC versions in different projects is not possible.

    - No possibility to contribute bugfixes and extensions via Git pull requests.

    Next steps: 1. See *Downloads* for how to download a stand-alone version (*.zip-file) of the PoC-Library. 2. See *Configuration* for how to configure PoC on a local system.

- **Stand-Alone IP Core Library cloned from Git:** Download PoC via `git clone` from GitHub as latest branch copy. IP cores can be copied into one or more destination projects or the projects link to the selected IP core source files.

    Advantages:

    - Simple and fast setup, configuring PoC is optional.

    - Access to the newest commits on a branch: New IP cores, new features, bugfixes.

    - Fast and simple updates via `git pull`.

    - After a configuration, PoC's additional features: simulation, synthesis, etc. can be used.

    - Contribute bugfixes and extensions via Git pull requests.

    Disadvantages:

    - Updated IP cores need to be copied again into the destination project.

– Using different PoC versions in different projects is not possible

Next steps: 1. See *Downloads* for how to clone a stand-alone version of the PoC-Library. 2. See *Configuration* for how to configure PoC on a local system.

• **Embedded IP Core Library as Git Submodule:** Integrate PoC as a Git submodule into the destination projects Git repository.

Advantages:

– Simple and fast setup, configuring PoC is optional, but recommended.

– Access to the newest commits on a branch: New IP cores, new features, bugfixes.

– Fast and simple updates via `git pull`.

– After a configuration, PoC's additional features: simulation, synthesis, etc. can be used.

– Moreover, some PoC infrastructure features can be used in the hosting repository and project as well.

– Contribute bugfixes and extensions via Git pull requests.

– Version linking between hosting Git and PoC.

Next steps: 1. See *Integration* for how to integrate PoC as a Git submodule into an existing Git. 2. See *Configuration* for how to configure PoC on a local system.

4.1 Requirements

Contents of this Page

• *Common requirements:*
• *Linux specific requirements:*
    – *Optional Tools on Linux:*
• *Mac OS specific requirements:*
    – *Optional Tools on Mac OS:*
• *Windows specific requirements:*
    – *Optional Tools on Windows:*

The PoC-Library comes with some scripts to ease most of the common tasks, like running testbenches or generating IP cores. We choose to use Python 3 as a platform independent scripting environment. All Python scripts are wrapped in Bash or PowerShell scripts, to hide some platform specifics of Darwin, Linux or Windows.

4.1.1 Common requirements:

Programming Languages and Runtime Environments:

• Python 3 (≥ 3.5):
    – colorama
    – py-flags

All Python requirements are listed in requirements.txt and can be installed via: `sudo python3.5 -m pip install -r requirements.txt`

Synthesis tool chains:

• Altera Quartus II ≥ 13.0 or

- Altera Quartus Prime $\geq$ 15.1 or

- Intel Quartus Prime $\geq$ 16.1 or

- Lattice Diamond $\geq$ 3.6 or

- Xilinx ISE 14.7[1] or

- Xilinx Vivado $\geq$ 2016.3[2]

Simulation tool chains

- Aldec Active-HDL (or Student Edition) or

- Aldec Active-HDL Lattice Edition or

- Mentor Graphics ModelSim PE (or Student Edition) or

- Mentor Graphics ModelSim SE or

- Mentor Graphics ModelSim Altera Edition or

- Mentor Graphics QuestaSim or

- Xilinx ISE Simulator 14.7 or

- Xilinx Vivado Simulator $\geq$ 2016.3[3] or

- GHDL $\geq$ 0.34dev and GTKWave $\geq$ 3.3.70

4.1.2 Linux specific requirements:

Debian and Ubuntu specific:

- `bash` is configured as `/bin/sh` (read more) `dpkg-reconfigure dash`

Optional Tools on Linux:

Git The command line tools to manage Git repositories. It's possible to extend the shell prompt with Git information.

SmartGit A Git client to handle complex Git flows in a GUI.

Generic Colouriser (grc) $\geq$ 1.9 Colorizes outputs of foreign scripts and programs. GRC is hosted on GitHub The latest *.deb installation packages can be downloaded here.

4.1.3 Mac OS specific requirements:

Bash $\geq$ 4.3 Mac OS is shipped with Bash 3.2. Use Homebrew to install an up-to-date Bash `brew install bash`

coreutils Mac OS' `readlink` program has a different behavior than the Linux version. The `coreutils` package installs a GNU readlink clone called `greadlink`. `brew install coreutils`

---

[1] Xilinx discontinued ISE since Oct. 2013. The last release was 14.7.

[2] Due to numerous bugs in the Xilinx Vivado Synthesis (incl. 2016.1), PoC can offer only a restricted Vivado support. See PoC's `Vivado` branch for a set of workarounds. The list of issues is documented on the Known Issues page.

[3] Due to numerous bugs in the Xilinx Simulator (incl. 2016.1), PoC can offer only a restricted Vivado support. The list of issues is documented on the Known Issues page.

Optional Tools on Mac OS:

Git The command line tools to manage Git repositories. It's possible to extend the shell prompt with Git information.

SmartGit or SourceTree A Git client to handle complex Git flows in a GUI.

Generic Colouriser (grc) $\geq$ 1.9 Colorizes outputs of foreign scripts and programs. GRC is hosted on GitHub
```
brew install Grc
```

4.1.4 Windows specific requirements:

PowerShell

- **Allow local script execution** (read more) `PS> Set-ExecutionPolicy RemoteSigned`

- **PowerShell $\geq$ 5.0 (recommended)** PowerShell 5.0 is shipped since Windows 10. It is a part if the Windows Management Framework 5.0 (WMF). Windows 7 and 8/8.1 can be updated to WMF 5.0. The package does not include **PSReadLine**, which is included in the Windows 10 PowerShell environment. Install PSReadLine manually: `PS> Install-Module PSReadline`.

- **PowerShell 4.0** PowerShell is shipped with Windows since Vista. If the required version not already included in Windows, it can be downloaded from Microsoft.com: WMF 4.0

Optional Tools on Windows:

PowerShell $\geq$ 4.0

- **PSReadLine** replaces the command line editing experience in PowerShell for versions 3 and up.

- **PowerShell Community Extensions (PSCX) $\geq$ 3.2** The latest PSCX can be downloaded from PowerShellGallery `PS> Install-Module Pscx` Note: PSCX $\geq$ 3.2.1 is required for PowerShell $\geq$ 5.0.

Git (MSys-Git) The command line tools to manage Git repositories.

SmartGit or SourceTree A Git client to handle complex Git flows in a GUI.

posh-git PowerShell integration for Git `PS> Install-Module posh-git`

4.2 Downloading PoC

Contents of this Page

4.2.1 Downloading from GitHub

The PoC-Library can be downloaded as a zip-file from GitHub. See the following table, to choose your desired git branch.

Branch	Download Link
master	zip-file [ZIP]
release	zip-file [ZIP]

4.2.2 Downloading via `git clone`

The PoC-Library can be downloaded (cloned) with `git clone` from GitHub. GitHub offers the transfer protocols HTTPS and SSH. You should use SSH if you have a GitHub account and have already uploaded an OpenSSH public key to GitHub, otherwise use HTTPS if you have no account or you want to use login credentials.

The created folder `<GitRoot>\PoC` is used as `<PoCRoot>` in later instructions or on other pages in this documentation.

Protocol	GitHub Repository URL
HTTPS	https://github.com/VLSI-EDA/PoC.git
SSH	ssh://git@github.com:VLSI-EDA/PoC.git

On Linux

Command line instructions to clone the PoC-Library onto a Linux machine with HTTPS protocol:

```
cd GitRoot
git clone --recursive "https://github.com/VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
```

Command line instructions to clone the PoC-Library onto a Linux machine machine with SSH protocol:

```
cd GitRoot
git clone --recursive "ssh://git@github.com:VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
```

On OS X

Please see the Linux instructions.

On Windows

Note: All Windows command line instructions are intended for **Windows PowerShell**, if not marked otherwise. So executing the following instructions in Windows Command Prompt (**cmd.exe**) won't function or result in errors! See the *Requirements section* on where to download or update PowerShell.

Command line instructions to clone the PoC-Library onto a Windows machine with HTTPS protocol:

```
cd GitRoot
git clone --recursive "https://github.com/VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
```

Command line instructions to clone the PoC-Library onto a Windows machine with SSH protocol:

```
cd GitRoot
git clone --recursive "ssh://git@github.com:VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
```

Note: The option `--recursive` performs a recursive clone operation for all linked git submodules. An additional `git submodule init` and `git submodule update` call is not needed anymore.

4.2.3 Downloading via `git submodule add`

The PoC-Library is meant to be integrated into other HDL projects (preferably Git versioned projects). Therefore it's recommended to create a library folder and add the PoC-Library as a git submodule.

The following command line instructions will create a library folder :file:'lib' and clone PoC as a git submodule into the subfolder :file:'<ProjectRoot>libPoC'.

On Linux

Command line instructions to clone the PoC-Library onto a Linux machine with HTTPS protocol:

```
cd ProjectRoot
mkdir lib
git submodule add "https://github.com/VLSI-EDA/PoC.git" lib/PoC
cd lib/PoC
git remote rename origin github
cd ../..
git add .gitmodules lib/PoC
git commit -m "Added new git submodule PoC in 'lib/PoC' (PoC-Library)."
```

Command line instructions to clone the PoC-Library onto a Linux machine machine with SSH protocol:

```
cd ProjectRoot
mkdir lib
git submodule add "ssh://git@github.com:VLSI-EDA/PoC.git" lib/PoC
cd lib/PoC
git remote rename origin github
cd ../..
git add .gitmodules lib/PoC
git commit -m "Added new git submodule PoC in 'lib/PoC' (PoC-Library)."
```

On OS X

Please see the Linux instructions.

On Windows

Note: All Windows command line instructions are intended for **Windows PowerShell**, if not marked otherwise. So executing the following instructions in Windows Command Prompt (**cmd.exe**) won't function or result in errors! See the *Requirements section* on where to download or update PowerShell.

Command line instructions to clone the PoC-Library onto a Windows machine with HTTPS protocol:

```
cd <ProjectRoot>
mkdir lib | cd
git submodule add "https://github.com/VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
cd ..\..
git add .gitmodules lib\PoC
git commit -m "Added new git submodule PoC in 'lib\PoC' (PoC-Library)."
```

Command line instructions to clone the PoC-Library onto a Windows machine with SSH protocol:

```
cd <ProjectRoot>
mkdir lib | cd
git submodule add "ssh://git@github.com:VLSI-EDA/PoC.git" PoC
cd PoC
git remote rename origin github
cd ..\..
git add .gitmodules lib\PoC
git commit -m "Added new git submodule PoC in 'lib\PoC' (PoC-Library)."
```

4.3 Integrating PoC into Projects

Contents of this page

4.3.1 As a Git submodule

The following command line instructions will integrate PoC into a existing Git repository and register PoC as a Git submodule. Therefore a directory lib\PoC\ is created and the PoC-Library is cloned as a Git submodule into that directory.

On Linux

```
cd ProjectRoot
mkdir lib
cd lib
git submodule add https://github.com/VLSI-EDA/PoC.git PoC
cd PoC
git remote rename origin github
cd ../..
git add .gitmodules lib\PoC
git commit -m "Added new git submodule PoC in 'lib/PoC' (PoC-Library)."
```

On OS X

Please see the Linux instructions.

On Windows

Note: All Windows command line instructions are intended for **Windows PowerShell**, if not marked otherwise. So executing the following instructions in Windows Command Prompt (**cmd.exe**) won't function or result in errors! See the Requirements section on where to download or update PowerShell.

```
cd ProjectRoot
mkdir lib | cd
git submodule add https://github.com/VLSI-EDA/PoC.git PoC
cd PoC
git remote rename origin github
cd ..\..
git add .gitmodules lib\PoC
git commit -m "Added new git submodule PoC in 'lib\PoC' (PoC-Library)."
```

See also:

Configuring PoC on a Local System

Create PoC's VHDL Configuration Files

4.4 Configuring PoC's Infrastructure

To explore PoC's full potential, it's required to configure some paths and synthesis or simulation tool chains. It's possible to relaunch the process at any time, for example to register new tools or to update tool versions.

Contents of this page

- *Overview*
- *The PoC-Library*
- *Git*
- *Aldec*
    - *Active-HDL*
- *Altera*
    - *Quartus*
    - *ModelSim Altera Edition*
- *Lattice*
    - *Diamond*
    - *Active-HDL Lattice Edition*
- *Mentor Graphics*
    - *QuestaSim*
- *Xilinx*
    - *ISE*

- – *Vivado*
- *GHDL*
- *GTKWave*
- *Hook Files*

4.4.1 Overview

The setup process is started by invoking PoC's frontend script with the command `configure`. Please follow the instructions on screen. Use the keyboard buttons: to accept, to decline, to skip/pass a step and to accept a default value displayed in brackets.

Optionally, a vendor or tool chain name can be passed to the configuration process to launch only its configuration routines.

On Linux:

```
cd ProjectRoot
./lib/PoC/poc.sh configure
# with tool chain name
./lib/PoC/poc.sh configure Xilinx.Vivado
```

On OS X

Please see the Linux instructions.

On Windows

Note: All Windows command line instructions are intended for **Windows PowerShell**, if not marked otherwise. So executing the following instructions in Windows Command Prompt (**cmd.exe**) won't function or result in errors! See the Requirements section on where to download or update PowerShell.

```
cd ProjectRoot
.\lib\PoC\poc.ps1 configure
# with tool chain name
.\lib\PoC\poc.ps1 configure Xilinx.Vivado
```

Introduction screen:

```
PS D:\git\PoC> .\poc.ps1 configure
================================================================================
                        The PoC-Library - Service Tool
================================================================================
Explanation of abbreviations:
  Y - yes      P       - pass (jump to next question)
  N - no       Ctrl + C - abort (no changes are saved)
Upper case or value in '[...]' means default value
--------------------------------------------------------------------------------

Configuring PoC
  PoC version: v1.0.1 (found in git)
  Installation directory: D:\git\PoC (found in environment variable)
```

4.4.2 The PoC-Library

PoC itself has a fully automated configuration routine. It detects if PoC is under Git control. If so, it extracts the current version number from the latest Git tag. The installation directory is infered from `$PoCRootDirectory` setup by `PoC.ps1` or `poc.sh`.

```
Configuring PoC
  PoC version: v1.0.1 (found in git)
  Installation directory: D:\git\PoC (found in environment variable)
```

4.4.3 Git

Note: Setting up Git and Git developer settings, is an advanced feature recommended for all developers interrested in providing Git pull requests or patches.

```
Configuring Git
  Git installation directory [C:\Program Files\Git]:
  Install Git mechanisms for PoC developers? [y/N/p]: y
  Install Git filters? [Y/n/p]:
  Installing Git filters...
  Install Git hooks? [Y/n/p]:
  Installing Git hooks...
  Setting 'pre-commit' hook for PoC...
```

4.4.4 Aldec

Configure the installation directory for all Aldec tools.

```
Configuring Aldec
  Are Aldec products installed on your system? [Y/n/p]: Y
  Aldec installation directory [C:\Aldec]:
```

Active-HDL

```
Configuring Aldec Active-HDL
  Is Aldec Active-HDL installed on your system? [Y/n/p]: Y
  Aldec Active-HDL version [10.3]:
  Aldec Active-HDL installation directory [C:\Aldec\Active-HDL]: C:\Aldec\Active-
↪HDL-Student-Edition
```

4.4.5 Altera

Configure the installation directory for all Altera tools.

```
    Configuring Altera
Are Altera products installed on your system? [Y/n/p]: Y
Altera installation directory [C:\Altera]:
```

Quartus

```
Configuring Altera Quartus
  Is Altera Quartus-II or Quartus Prime installed on your system? [Y/n/p]: Y
  Altera Quartus version [15.1]: 16.0
  Altera Quartus installation directory [C:\Altera\16.0\quartus]:
```

ModelSim Altera Edition

```
Configuring ModelSim Altera Edition
  Is ModelSim Altera Edition installed on your system? [Y/n/p]: Y
  ModelSim Altera Edition installation directory [C:\Altera\15.0\modelsim_ae]:␣
→C:\Altera\16.0\modelsim_ase
```

4.4.6 Lattice

Configure the installation directory for all Lattice Semiconductor tools.

```
Configuring Lattice
  Are Lattice products installed on your system? [Y/n/p]: Y
  Lattice installation directory [D:\Lattice]:
```

Diamond

```
Configuring Lattice Diamond
  Is Lattice Diamond installed on your system? [Y/n/p]: >
  Lattice Diamond version [3.7]:
  Lattice Diamond installation directory [D:\Lattice\Diamond\3.7_x64]:
```

Active-HDL Lattice Edition

```
Configuring Active-HDL Lattice Edition
  Is Aldec Active-HDL installed on your system? [Y/n/p]: Y
  Active-HDL Lattice Edition version [10.2]:
  Active-HDL Lattice Edition installation directory [D:\Lattice\Diamond\3.7_
→x64\active-hdl]:
```

4.4.7 Mentor Graphics

Configure the installation directory for all mentor Graphics tools.

```
Configuring Mentor
  Are Mentor products installed on your system? [Y/n/p]: Y
  Mentor installation directory [C:\Mentor]:
```

QuestaSim

```
Configuring Mentor QuestaSim
  Is Mentor QuestaSim installed on your system? [Y/n/p]: Y
  Mentor QuestaSim version [10.4d]: 10.4c
  Mentor QuestaSim installation directory [C:\Mentor\QuestaSim\10.4c]:␣
→C:\Mentor\QuestaSim64\10.4c
```

4.4.8 Xilinx

Configure the installation directory for all Xilinx tools.

```
Configuring Xilinx
  Are Xilinx products installed on your system? [Y/n/p]: Y
  Xilinx installation directory [C:\Xilinx]:
```

ISE

If an Xilinx ISE environment is available and shall be configured in PoC, then answer the following questions:

```
Configuring Xilinx ISE
  Is Xilinx ISE installed on your system? [Y/n/p]: Y
  Xilinx ISE installation directory [C:\Xilinx\14.7\ISE_DS]:
```

Vivado

If an Xilinx ISE environment is available and shall be configured in PoC, then answer the following questions:

```
Configuring Xilinx Vivado
  Is Xilinx Vivado installed on your system? [Y/n/p]: Y
  Xilinx Vivado version [2016.2]:
  Xilinx Vivado installation directory [C:\Xilinx\Vivado\2016.2]:
```

4.4.9 GHDL

```
Configuring GHDL
  Is GHDL installed on your system? [Y/n/p]: Y
  GHDL installation directory [C:\Tools\GHDL\0.34dev]:
```

4.4.10 GTKWave

```
Configuring GTKWave
  Is GTKWave installed on your system? [Y/n/p]: Y
  GTKWave installation directory [C:\Tools\GTKWave\3.3.71]:
```

4.4.11 Hook Files

PoC's wrapper scripts can be customized through pre- and post-hook file. See Wrapper Script Hook Files for more details.

4.5 Creating my_config/my_project.vhdl

The PoC-Library needs two VHDL files for its configuration. These files are used to determine the most suitable implementation depending on the provided platform information. These files are also used to select appropiate work arounds.

4.5.1 Create my_config.vhdl

The **my_config.vhdl** file can easily be created from the template file `my_config.vhdl.template` provided by PoC in `PoCRoot\src\common`. (View source on GitHub.) Copy this file into the project's source directory and rename it to `my_config.vhdl`.

This file should be included in version control systems and shared with other systems. `my_config.vhdl` defines three global constants, which need to be adjusted:

```vhdl
constant MY_BOARD   : string  := "CHANGE THIS"; -- e.g. Custom, ML505, KC705, Atlys
constant MY_DEVICE  : string  := "CHANGE THIS"; -- e.g. None, XC5VLX50T-1FF1136,
→EP2SGX90FF1508C3
constant MY_VERBOSE : boolean := FALSE;         -- activate report statements in
→VHDL subprograms
```

The easiest way is to define a board name and set `MY_DEVICE` to `None`. So the device name is infered from the board information stored in `PoCRoot\src\common\config.vhdl`. If the requested board is not known to PoC or it's custom made, then set `MY_BOARD` to `Custom` and `MY_DEVICE` to the full FPGA device string.

Example 1: A "Stratix II GX Audio Video Development Kit" board:

```vhdl
constant MY_BOARD  : string := "S2GXAV";  -- Stratix II GX Audio Video Development
→Kit
constant MY_DEVICE : string := "None";    -- infer from MY_BOARD
```

Example 2: A custom made Spartan-6 LX45 board:

```vhdl
constant MY_BOARD  : string := "Custom";
constant MY_DEVICE : string := "XC6SLX45-3CSG324";
```

4.5.2 Create my_project.vhdl

The **my_project.vhdl** file can also be created from a template file `my_project.vhdl.template` provided by PoC in `PoCRoot\src\common`.

The file should to be copied into a projects source directory and renamed into `my_project.vhdl`. This file **must not** be included into version control systems – it's private to a computer. `my_project.vhdl` defines two global constants, which need to be adjusted:

```vhdl
constant MY_PROJECT_DIR      : string := "CHANGE THIS"; -- e.g. "d:/vhdl/myproject/
→", "/home/me/projects/myproject/"
constant MY_OPERATING_SYSTEM : string := "CHANGE THIS"; -- e.g. "WINDOWS", "LINUX"
```

Example 1: A Windows System:

```vhdl
constant MY_PROJECT_DIR      : string := "D:/git/GitHub/PoC/";
constant MY_OPERATING_SYSTEM : string := "WINDOWS";
```

Example 2: A Debian System:

```vhdl
constant MY_PROJECT_DIR      : string := "/home/paebbels/git/GitHub/PoC/";
constant MY_OPERATING_SYSTEM : string := "LINUX";
```

See also:

Running one or more testbenches The installation can be checked by running one or more of PoC's testbenches.

Running one or more netlist generation flows The installation can also be checked by running one or more of PoC's synthesis flows.

4.6 Adding IP Cores to a Project

4.6.1 Manually Addind IP Cores

Adding IP Cores to Altera Quartus

Todo: No documentation available.

Adding IP Cores to Lattice Diamond

Todo: No documentation available.

Adding IP Cores to Xilinx ISE

Todo: No documentation available.

Adding IP Cores to Xilinx Vivado

Todo: No documentation available.

4.7 Simulation

Contents of this Page

4.7.1 Overview

The Python Infrastructure shipped with the PoC-Library can launch manual, half-automated and fully automated testbenches. The testbench can be run in command line or GUI mode. If available, the used simulator is launched with pre-configured waveform files. This can be done by invoking one of PoC's frontend script:

- **poc.sh:** `poc.sh <common options> <simulator> <module> <simulator options>`
  Use this fronend script on Darwin, Linux and Unix platforms.

- **poc.ps1:** `poc.ps1 <common options> <simulator> <module> <simulator options>` Use this frontend script Windows platforms.

> **Attention:** All Windows command line instructions are intended for Windows PowerShell, if not marked otherwise. So executing the following instructions in Windows Command Prompt (`cmd.exe`) won't function or result in errors!

See also:

PoC Configuration See the Configuration page on how to configure PoC and your installed simulator tool chains. This is required to invoke the simulators.

Supported Simulators See the Intruction page for a list of supported simulators.

4.7.2 Quick Example

The following quick example uses the GHDL Simulator to analyze, elaborate and simulate a testbench for the module `arith_prng` (Pseudo Random Number Generator - PRNG). The VHDL file `arith_prng.vhdl` is located at `PoCRoot\src\arith` and virtually a member in the *PoC.arith* namespace. So the module can be identified by an unique name: `PoC.arith.prng`, which is passed to the frontend script.

Example 1:

```
cd PoCRoot
.\poc.ps1 ghdl PoC.arith.prng
```

The CLI command `ghdl` chooses *GHDL Simulator* as the simulator and passes the fully qualified PoC entity name `PoC.arith.prng` as a parameter to the tool. All required source file are gathered and compiled to an executable. Afterwards this executable is launched in CLI mode and it's outputs are displayed in console:

Each testbench uses PoC's simulation helper packages to count asserts and to track active stimuli and checker processes. After a completed simulation run, an report is written to STDOUT or the simulator's console. Note the line `SIMULATION RESULT = PASSED`. For each simulated PoC entity, a line in the overall report is created. It lists the runtime per testbench and the simulation status (`...` `ERROR`, `FAILED`, `NO ASSERTS` or `PASSED`).

Example 2:

Passing an additional option `--gui` to the service tool, opens the testbench in GUI-mode. If a waveform configuration file is present (e.g. a `*.gtkw` file for GTKWave), then it is preloaded into the simulator's waveform viewer.

```
cd PoCRoot
.\poc.ps1 ghdl PoC.arith.prng --gui
```

The opened waveform viewer and displayed waveform should look like this:
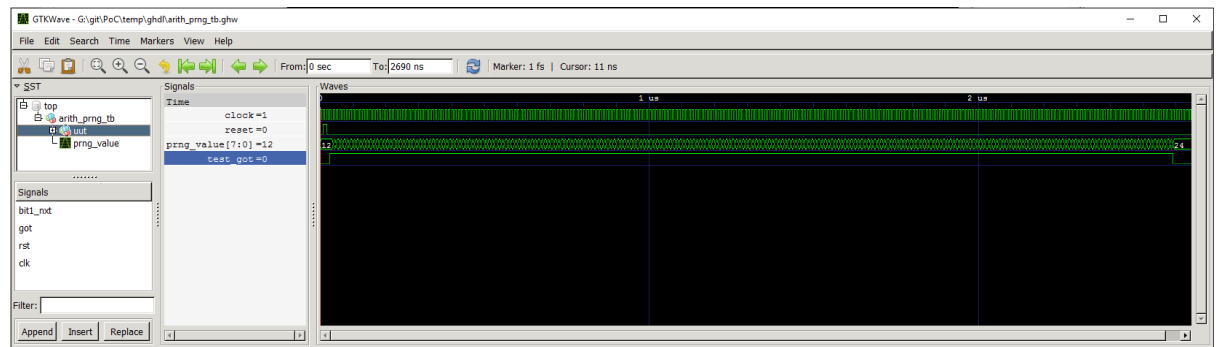
4.7.3 Vendor Specific Testbenches

PoC is shipped with a set of well known FPGA development boards. This set is extended by a list of generic boards, named after each supported FPGA vendor. These generic boards can be used in simulations to select a representative FPGA of a supported device vendor. If no board or device name is passed to a testbench run, the `GENERIC` board is chosen.

Board Name	Target Board	Target Device
GENERIC	GENERIC	GENERIC
Altera	DE4	Stratix-IV 230
Lattice	ECP5Versa	ECP5-45UM
Xilinx	KC705	Kintex-7 325T

A vendor specific testbench can be launched by passing either `--board=xxx` or `--device=yyy` as an additional parameter to the PoC scripts.

```
# Example 1 – A Lattice board
.\poc.ps1 ghdl PoC.arith.prng --board=Lattice
# Example 2 – A Altera Stratix IV board
.\poc.ps1 ghdl PoC.arith.prng --board=DE4
# Example 3 – A Xilinx Kintex-7 325T device
.\poc.ps1 ghdl PoC.arith.prng --device=XC7K325T-2FFG900
```

Note: Running vendor specific testbenches may require pre-compiled vendor libraries. Some simulators are shipped with diverse pre-compiled libraries, others include scripts or user guides to pre-compile them on the target system.

PoC is shipped with a set of pre-compile scripts to offer a unified interface and common storage for all supported vendor's pre-compile procedures. See Pre-Compiling Vendor Libraries.

4.7.4 Running a Single Testbench

A testbench run is supervised by PoC's `PoCRoot\py\PoC.py` service tool, which offers a consistent interface to all simulators. Unfortunately, every platform has it's specialties, so a wrapper script is needed as abstraction from the host's operating system. Depending on the choosen tool chain, the wrapper script will source or invoke the vendor tool's environment scripts to pre-load the needed environment variables, paths or license file settings.

The order of options to the frontend script is as following: `<common options> <simulator> <module> <simulator options>`

The frontend offers several common options:

Common Option		Description
-q	–quiet	Quiet-mode (print nothing)
-v	–verbose	Print more messages
-d	–debug	Debug mode (print everything)
	–dryrun	Run in dry-run mode

One of the following supported simulators can be choosen, if installed and configured in PoC:

Simulator	Description
asim	Active-HDL Simulator
cocotb	Cocotb simulation using QuestaSim Simulator
ghdl	GHDL Simulator
isim	Xilinx ISE Simulator
vsim	QuestaSim Simulator or ModelSim
xsim	Xilinx Vivado Simulator

A testbench run can be interrupted by sending a keyboard interrupt to Python. On most operating systems this is done by pressing `Ctrl + C`. If PoC runs multiple testbenches at once, all finished testbenches are reported with there testbench result. The aborted testbench will be listed as errored.

Aldec Active-HDL

The command to invoke a simulation using Active-HDL is `asim` followed by a list of PoC entities. The following options are supported for Active-HDL:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
	–std=[87|93|02|08]	Select a VHDL standard. Default: 08

Note: GUI mode for Active-HDL is not yet supported.

Example:

```
cd PoCRoot
.\poc.ps1 asim PoC.arith.prng --std=93
```

Cocotb with QuestaSim backend

The command to invoke a Cocotb simulation using QuestaSim is `cocotb` followed by a list of PoC entities. The following options are supported for Cocotb:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
-g	–gui	Start the simulation in the QuestaSim GUI.

Note: Cocotb is currently only on Linux with QuestaSim supported. We are working to support the Windows platform and the GHDL backend.

Example:

```
cd PoCRoot
.\poc.ps1 cocotb PoC.cache.par
```

GHDL (plus GTKwave)

The command to invoke a simulation using GHDL is `ghdl` followed by a list of PoC entities. The following options are supported for GHDL:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
-g	–gui	Start GTKwave, if installed. Open *.gtkw, if available.
	–std=[87|93|02|08]	Select a VHDL standard. Default: 08

Example:

```
cd PoCRoot
.\poc.ps1 ghdl PoC.arith.prng --board=Atlys -g
```

Mentor Graphics QuestaSim

The command to invoke a simulation using QuestaSim or ModelSim is `vsim` followed by a list of PoC entities. The following options are supported for QuestaSim:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
-g	–gui	Start the simulation in the QuestaSim GUI.
	–std=[87|93|02|08]	Select a VHDL standard. Default: 08

Example:

```
cd PoCRoot
.\poc.ps1 vsim PoC.arith.prng --board=DE4 --gui
```

If QuestaSim is started in GUI mode (`--gui`), PoC will provide several Tcl files (`*.do`) in the simulator's working directory to recompile, restart or rerun the current simulation. The rerun command is based on the saved IP core's run script, which may default to `run -all`.

Tcl Script	Performed Tasks
`recompile.do`	recompile and restart
`relaunch.do`	recompile, restart and rerun
`saveWaveform.do`	save the current waveform viewer settings

Xilinx ISE Simulator

The command to invoke a simulation using ISE Simulator (isim) is `isim` followed by a list of PoC entities. The following options are supported for ISE Simulator:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
-g	–gui	Start the simulation in the ISE Simulator GUI (iSim).

Example:

```
cd PoCRoot
.\poc.ps1 isim PoC.arith.prng --board=Atlys -g
```

Xilinx Vivado Simulator

The command to invoke a simulation using Vivado Simulator (isim) is `xsim` followed by a list of PoC entities. The following options are supported for Vivado Simulator:

Simulator Option		Description
	–board=<BOARD>	Specify a target board.
	–device=<DEVICE>	Specify a target device.
-g	–gui	Start Vivado in simulation mode.
	–std=[93|08]	Select a VHDL standard. Default: 93

Example:

```
cd PoCRoot
.\poc.ps1 xsim PoC.arith.prng --board=Atlys -g
```

4.7.5 Running a Group of Testbenches

Each simulator can be invoked with a space seperated list of PoC entiries or a wildcard at the end of the fully qualified entity name.

Supported wildcard patterns are `*` and `?`. Question mark refers to all entities in a PoC (sub-)namespace. Asterisk refers to all PoC entiries in the current namespace and all sub-namespaces.

Examples for testbenches groups:

PoC entity list	Description
PoC.arith.prng	A single PoC entity: `arith_prng`
PoC.*	All entities in the whole library
PoC.io.ddrio.?	All entities in `PoC.io.ddrio`: `ddrio_in`, `ddrio_inout`, `ddrio_out`
PoC.fifo.* PoC.cache.* PoC.dstruct.*	All FIFO, cache and data-structure testbenches.

```
cd PoCRoot
.\poc.ps1 -q asim PoC.arith.prng PoC.io.ddrio.* PoC.sort.lru_cache
```

Resulting output:

4.7.6 Continuous Integration (CI)

All PoC testbenches are executed on every GitHub upload (push) via Travis-CI. The testsuite runs all testbenches for the virtual board `GENERIC` with an FPGA device called `GENERIC`. We can't run vendor dependent testbenches, because we can't upload the vendor simulation libraries to Travis-CI.

To reproduce the Travis-CI results on a local machine, run the following command. The `-q` option, launches the frontend in quiet mode to reduce the command line messages:

```
cd PoCRoot
.\poc.ps1 -q ghdl PoC.*
```

```
Windows PowerShell                                                    —   □   ✕
Testbench: PoC.sort.sortnet.OddEvenSort
Testbench: PoC.sort.sortnet.OddEvenMergeSort
Testbench: PoC.sort.sortnet.Stream_Adapter
Testbench: PoC.sort.sortnet.Stream_Adapter2
Testbench: PoC.sort.lru_cache
=============================================================================
                            Overall Simulation Report
=============================================================================
Name                        | Time |  Status
-----------------------------------------------------------------------------
arith                       |      |
  addw                      | 3:04 |   PASSED
  convert_bin2bcd           | 0:01 |   PASSED
  counter_bcd               | 0:02 |   PASSED
  div                       | 0:03 |   PASSED
  firstone                  | 0:02 |   PASSED
  prefix_and                | 0:01 |   PASSED
  prefix_or                 | 0:01 |   PASSED
  prng                      | 0:01 |   PASSED
  scaler                    | 0:02 |   PASSED
dstruct                     |      |
  deque                     | 0:02 |   PASSED
  stack                     | 0:02 |   PASSED
fifo                        |      |
  cc_got                    | 0:02 |   PASSED
  cc_got_tempput            | 0:02 |   PASSED
  ic_assembly               | 0:02 |   PASSED
  ic_got                    | 0:02 |   PASSED
io                          |      |
  ddrio                     |      |
    in                      | 0:01 |   PASSED
    inout                   | 0:01 |   PASSED
    out                     | 0:01 |   PASSED
  uart                      |      |
    rx                      | 0:02 |   PASSED
  Debounce                  | 0:02 |   PASSED
mem                         |      |
  lut                       |      |
    Sine                    | 0:01 | NO ASSERTS
  ocram                     |      |
    sdp                     | 0:01 |   PASSED
misc                        |      |
  gearbox                   |      |
    down_cc                 | 0:02 | NO ASSERTS
    down_dc                 | 0:02 |   FAILED
    up_cc                   | 0:02 | NO ASSERTS
    up_dc                   | 0:01 |   FAILED
  stat                      |      |
    Average                 | 0:02 |   PASSED
    Histogram               | 0:04 | NO ASSERTS
    Minimum                 | 0:01 |   PASSED
    Maximum                 | 0:01 |   PASSED
  sync                      |      |
    Bits                    | 0:01 |   PASSED
    Reset                   | 0:01 | NO ASSERTS
    Strobe                  | 0:02 | NO ASSERTS
    Vector                  | 0:02 | NO ASSERTS
    Command                 | 0:02 | NO ASSERTS
sort                        |      |
  sortnet                   |      |
    BitonicSort             | 0:56 |   PASSED
    OddEvenSort             | 2:54 |   PASSED
    OddEvenMergeSort        | 0:46 |   PASSED
    Stream_Adapter          | 0:03 |   PASSED
    Stream_Adapter2         | 0:04 |   PASSED
  lru_cache                 | 0:02 | NO ASSERTS
=============================================================================
Time:  9:07  Count: 41   Passed: 30   No Asserts: 9   Failed: 2   Errors: 0
=============================================================================
PS G:\git\PoC>
```

If the vendor libraries are available and pre-compiled, then it's also possible to run a CI flow for a specific vendor. This is an Altera example for the Terrasic DE4 board:

```
cd PoCRoot
.\poc.ps1 -q vsim PoC.* --board=DE4
```

See also:

PoC Configuration See the Configuration page on how to configure PoC and your installed simulator tool chains. This is required to invoke the simulators.

Latest Travis-CI Report Browse the list of branches at Travis-CI.org.

4.8 Synthesis

Contents of this Page

4.8.1 Overview

The Python infrastructure shipped with the PoC-Library can launch manual, half-automated and fully automated synthesis runs. This can be done by invoking one of PoC's frontend script:

- **poc.sh:** `poc.sh <common options> <compiler> <module> <compiler options>` Use this fronend script on Darwin, Linux and Unix platforms.

- **poc.ps1:** `poc.ps1 <common options> <compiler> <module> <compiler options>` Use this frontend script Windows platforms.

> **Attention:** All Windows command line instructions are intended for Windows PowerShell, if not marked otherwise. So executing the following instructions in Windows Command Prompt (`cmd.exe`) won't function or result in errors!

See also:

PoC Configuration See the Configuration page on how to configure PoC and your installed synthesis tool chains. This is required to invoke the compilers.

Supported Compiler See the Intruction page for a list of supported compilers.

See also:

List of Supported FPGA Devices See this list to find a supported and well known target device.

List of Supported Development Boards See this list to find a supported and well known development board.

4.8.2 Quick Example

The following quick example uses the Xilinx Systesis Tool (XST) to synthesize a netlist for IP core `arith_prng` (Pseudo Random Number Generator - PRNG). The VHDL file `arith_prng.vhdl` is located at `PoCRoot\src\arith` and virtually a member in the *PoC.arith* namespace. So the module can be identified by an unique name: `PoC.arith.prng`, which is passed to the frontend script.

Example 1:

```
cd PoCRoot
.\poc.ps1 xst PoC.arith.prng --board=KC705
```

The CLI command `xst` chooses *Xilinx Synthesis Tool* as the synthesizer and passes the fully qualified PoC entity name `PoC.arith.prng` as a parameter to the tool. Additionally, the development board name is required to load the correct `my_config.vhdl` file. All required source file are gathered and synthesized to a netlist.

4.8.3 Running a single Synthesis

A synthesis run is supervised by PoC's PoCRoot\py\PoC.py service tool, which offers a consistent interface to all synthesizers. Unfortunately, every platform has it's specialties, so a wrapper script is needed as abstraction from the host's operating system. Depending on the choosen tool chain, the wrapper script will source or invoke the vendor tool's environment scripts to pre-load the needed environment variables, paths or license file settings.

The order of options to the frontend script is as following: `<common options> <synthesizer> <module> [<module>] <synthesizer options>`

The frontend offers several common options:

Common Option		Description
-q	--quiet	Quiet-mode (print nothing)
-v	--verbose	Print more messages
-d	--debug	Debug mode (print everything)
	--dryrun	Run in dry-run mode

One of the following supported synthesizers can be choosen, if installed and configured in PoC:

Synthesizer	Command Reference
Altera Quartus II or Intel Quartus Prime	PoC.py quartus
Lattice (Diamond) Synthesis Engine (LSE)	PoC.py lse
Xilinx ISE Systhesis Tool (XST)	PoC.py xst
Xilinx ISE Core Generator (CoreGen)	PoC.py coregen
Xilinx Vivado Synthesis	PoC.py vivado

Altera / Intel Quartus

The command to invoke a synthesis using Altera Quartus II or Intel Quartus Prime is quartus followed by a list of PoC entities. The following options are supported for Quartus:

Simulator Option		Description
	`--board=<Board>`	Specify a target board.
	`--device=<Device>`	Specify a target device.

Example:

```
cd PoCRoot
.\poc.ps1 quartus PoC.arith.prng --board=DE4
```

Lattice Diamond

The command to invoke a synthesis using Lattice Diamond is lse followed by a list of PoC entities. The following options are supported for the Lattice Synthesis Engine (LSE):

Simulator Option		Description
	`--board=<Board>`	Specify a target board.
	`--device=<Device>`	Specify a target device.

Example:

```
cd PoCRoot
.\poc.ps1 lse PoC.arith.prng --board=ECP5Versa
```

Xilinx ISE Synthesis Tool (XST)

The command to invoke a synthesis using Xilinx ISE Synthesis is xst followed by a list of PoC entities. The following options are supported for the Xilinx Synthesis Tool (XST):

Simulator Option		Description
	`--board=<Board>`	Specify a target board.
	`--device=<Device>`	Specify a target device.

Example:

```
cd PoCRoot
.\poc.ps1 xst PoC.arith.prng --board=KC705
```

Xilinx ISE Core Generator

The command to invoke an IP core generation using Xilinx Core Generator is coregen followed by a list of PoC entities. The following options are supported for Core Generator (CG):

Simulator Option		Description
	`--board=<Board>`	Specify a target board.
	`--device=<Device>`	Specify a target device.

Example:

```
cd PoCRoot
.\poc.ps1 coregen PoC.xil.mig.Atlys_1x128 --board=Atlys
```

Xilinx Vivado Synthesis

The command to invoke a synthesis using Xilinx Vivado Synthesis is vivado followed by a list of PoC entities. The following options are supported for Vivado Synthesis (Synth):

Simulator Option		Description
	`--board=<Board>`	Specify a target board.
	`--device=<Device>`	Specify a target device.

Example:

```
cd PoCRoot
.\poc.ps1 vivado PoC.arith.prng --board=KC705
```

4.9 Project Management

4.9.1 Overview

4.9.2 Solutions

4.9.3 Projects

4.10 Pre-Compiling Vendor Libraries

Contents of this Page

- *Overview*
- *Supported Simulators*
- *FPGA Vendor's Primitive Libraries*
  - *Altera*
  - *Lattice*
  - *Xilinx ISE*
  - *Xilinx Vivado*
- *Third-Party Libraries*
  - *OSVVM*
  - *UVVM*
- *Simulator Adapters*
  - *Cocotb*

4.10.1 Overview

Running vendor specific testbenches may require pre-compiled vendor libraries. Some vendors ship their simulators with diverse pre-compiled libraries, but these don't include primitive libraries from hardware vendors. More over, many auxillary libraries are outdated. Hardware vendors ship their tool chains with pre-compile scripts or user guides to pre-compile the primitive libraries for a list of supported simulators on a target system.

PoC is shipped with a set of pre-compile scripts to offer a unified interface and common storage for all supported vendor's pre-compile procedures. The scripts are located in `\tools\precompile\` and the output is stored in `\temp\precompiled\<Simulator>\<Library>`.

4.10.2 Supported Simulators

The current set of pre-compile scripts support these simulators:

Vendor	Simulator and Edition	Altera	Lattice	Xilinx (ISE)	Xilinx (Vivado)
T. Gingold	GHDL with `--std=93c` GHDL with `--std=08`	yes yes	yes yes	yes yes	yes yes
Aldec	Active-HDL (or Stududent Ed.) Active-HDL Lattice Ed. Reviera-PRO	planned planned planned	planned shipped planned	planned planned planned	planned planned planned
Mentor	ModelSim PE (or Stududent Ed.) ModelSim SE ModelSim Altera Ed. QuestaSim	yes yes shipped yes	yes yes yes yes	yes yes yes yes	yes yes yes yes
Xilinx	ISE Simulator Vivado Simulator			shipped not supported	not supported shipped

4.10.3 FPGA Vendor's Primitive Libraries

Altera

Note: The Altera Quartus tool chain needs to be configured in PoC. See *Configuring PoC's Infrastruture* for further details.

On Linux

```
# Example 1 - Compile for all Simulators
./tools/precompile/compile-altera.sh --all
# Example 2 - Compile only for GHDL and VHDL-2008
./tools/precompile/compile-altera.sh --ghdl --vhdl2008
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.
	--vhdl93	GHDL only: Compile only for VHDL-93.
	--vhdl2008	GHDL only: Compile only for VHDL-2008.

On Windows

```
# Example 1 – Compile for all Simulators
.\tools\precompile\compile-altera.ps1 -All
# Example 2 – Compile only for GHDL and VHDL-2008
.\tools\precompile\compile-altera.ps1 -GHDL -VHDL2008
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.
	-VHDL93	GHDL only: Compile only for VHDL-93.
	-VHDL2008	GHDL only: Compile only for VHDL-2008.

Lattice

Note: The Lattice Diamond tool chain needs to be configured in PoC. See *Configuring PoC's Infrastruture* for further details.

On Linux

```
# Example 1 – Compile for all Simulators
./tools/precompile/compile-lattice.sh --all
# Example 2 – Compile only for GHDL and VHDL-2008
./tools/precompile/compile-lattice.sh --ghdl --vhdl2008
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.
	--vhdl93	GHDL only: Compile only for VHDL-93.
	--vhdl2008	GHDL only: Compile only for VHDL-2008.

On Windows

```
# Example 1 - Compile for all Simulators
.\tools\precompile\compile-lattice.ps1 -All
# Example 2 - Compile only for GHDL and VHDL-2008
.\tools\precompile\compile-lattice.ps1 -GHDL -VHDL2008
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.
	-VHDL93	GHDL only: Compile only for VHDL-93.
	-VHDL2008	GHDL only: Compile only for VHDL-2008.

Xilinx ISE

Note: The Xilinx ISE tool chain needs to be configured in PoC. See *Configuring PoC's Infrastruture* for further details.

On Linux

```
# Example 1 - Compile for all Simulators
./tools/precompile/compile-xilinx-ise.sh --all
# Example 2 - Compile only for GHDL and VHDL-2008
./tools/precompile/compile-xilinx-ise.sh --ghdl --vhdl2008
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.
	--vhdl93	GHDL only: Compile only for VHDL-93.
	--vhdl2008	GHDL only: Compile only for VHDL-2008.

On Windows

```
# Example 1 - Compile for all Simulators
.\tools\precompile\compile-xilinx-ise.ps1 -All
# Example 2 - Compile only for GHDL and VHDL-2008
.\tools\precompile\compile-xilinx-ise.ps1 -GHDL -VHDL2008
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.
	-VHDL93	GHDL only: Compile only for VHDL-93.
	-VHDL2008	GHDL only: Compile only for VHDL-2008.

Xilinx Vivado

Note: The Xilinx Vivado tool chain needs to be configured in PoC. See *Configuring PoC's Infrastruture* for further details.

On Linux

```
# Example 1 - Compile for all Simulators
./tools/precompile/compile-xilinx-vivado.sh --all
# Example 2 - Compile only for GHDL and VHDL-2008
./tools/precompile/compile-xilinx-vivado.sh --ghdl --vhdl2008
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.
	--vhdl93	GHDL only: Compile only for VHDL-93.
	--vhdl2008	GHDL only: Compile only for VHDL-2008.

On Windows

```
# Example 1 - Compile for all Simulators
.\tools\precompile\compile-xilinx-vivado.ps1 -All
# Example 2 - Compile only for GHDL and VHDL-2008
.\tools\precompile\compile-xilinx-vivado.ps1 -GHDL -VHDL2008
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.
	-VHDL93	GHDL only: Compile only for VHDL-93.
	-VHDL2008	GHDL only: Compile only for VHDL-2008.

4.10.4 Third-Party Libraries

OSVVM

On Linux

```
# Example 1 – Compile for all Simulators
./tools/precompile/compile-osvvm.sh --all
# Example 2 – Compile only for GHDL
./tools/precompile/compile-osvvm.sh --ghdl
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.

On Windows

```
# Example 1 – Compile for all Simulators
.\tools\precompile\compile-osvvm.ps1 -All
# Example 2 – Compile only for GHDL
.\tools\precompile\compile-osvvm.ps1 -GHDL
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.

UVVM

On Linux

```
# Example 1 – Compile for all Simulators
./tools/precompile/compile-uvvm.sh --all
# Example 2 – Compile only for GHDL
./tools/precompile/compile-uvvm.sh --ghdl
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.

On Windows

```
# Example 1 - Compile for all Simulators
.\tools\precompile\compile-uvvm.ps1 -All
# Example 2 - Compile only for GHDL
.\tools\precompile\compile-uvvm.ps1 -GHDL
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.

4.10.5 Simulator Adapters

Cocotb

On Linux

> **Attention:** This is an experimental compile script.

```
# Example 1 - Compile for all Simulators
./tools/precompile/compile-cocotb.sh --all
# Example 2 - Compile only for GHDL
./tools/precompile/compile-cocotb.sh --ghdl
```

List of command line arguments:

Common Option		Parameter Description
-h	--help	Print embedded help page(s).
-c	--clean	Clean-up directories.
-a	--all	Compile for all simulators.
	--ghdl	Compile for GHDL.
	--questa	Compile for QuestaSim.

On Windows

> **Attention:** This is an experimental compile script.

```
# Example 1 - Compile for all Simulators
.\tools\precompile\compile-cocotb.ps1 -All
# Example 2 - Compile only for GHDL
.\tools\precompile\compile-cocotb.ps1 -GHDL
```

List of command line arguments:

Common Option		Parameter Description
-h	-Help	Print embedded help page(s).
-c	-Clean	Clean-up directories.
-a	-All	Compile for all simulators.
	-GHDL	Compile for GHDL.
	-Questa	Compile for QuestaSim.

4.11 Miscellaneous

The directory `PoCRoot\tools\` contains several tools and addons to ease the work with the PoC-Library and VHDL.

4.11.1 GNU Emacs

Todo: No documentation available.

4.11.2 Git

- `git-alias.setup.ps1`/`git-alias.setup.sh` registers new global aliasses in Git

    - `git tree` - Prints the colored commit tree into the console

    - `git treea` - Prints the colored commit tree into the console

```
git config --global alias.tree 'log --decorate --pretty=oneline --abbrev-
→commit --date-order --graph'
git config --global alias.tree 'log --decorate --pretty=oneline --abbrev-
→commit --date-order --graph --all'
```

Browse the Git directory.

4.11.3 Notepad++

The PoC-Library is shipped with syntax highlighting rules for Notepad++. The following additional file types are supported:

- PoC Configuration Files (*.ini)

- PoC *.Files Files* (.files)

- PoC *.Rules Files* (.rules)

- Xilinx User Constraint Files (*.ucf): `Syntax Highlighting – Xilinx UCF`

Browse the Notepad++ directory.

CHAPTER 5

Third Party Libraries

The pyIPCMI is shipped with different third party libraries, which are located in the `<pyIPCMIRoot>/lib/` folder. This document lists all these libraries, their websites and licenses.

Part III

References

CHAPTER 6

IP Core Management Infrastructure

class pyIPCMI.**pyIPCMIEntityAttribute**
    Bases: lib.pyAttribute.Attribute

    classmethod GetAttributes(*method*)

    classmethod GetMethods(*cl*)

class pyIPCMI.**BoardDeviceAttributeGroup**
    Bases: lib.pyAttribute.Attribute

    classmethod GetAttributes(*method*)

    classmethod GetMethods(*cl*)

class pyIPCMI.**VHDLVersionAttribute**
    Bases: lib.pyAttribute.Attribute

    classmethod GetAttributes(*method*)

    classmethod GetMethods(*cl*)

class pyIPCMI.**SimulationStepsAttributeGroup**
    Bases: lib.pyAttribute.Attribute

    classmethod GetAttributes(*method*)

    classmethod GetMethods(*cl*)

class pyIPCMI.**CompileStepsAttributeGroup**
    Bases: lib.pyAttribute.Attribute

    classmethod GetAttributes(*method*)

    classmethod GetMethods(*cl*)

class pyIPCMI.**IPCoreManagementInfrastructure**(*debug*, *verbose*, *quiet*, *dryRun*, *sphinx=False*)
    Bases: pyIPCMI.Base.Logging.ILogable, lib.pyAttribute.ArgParseAttributes.
    ArgParseMixin

    HeadLine = 'pyIPCMI - Service Tool'

    Platform

    DryRun

Directories

ConfigFiles

Config

Root

Repository

SaveAndReloadConfiguration()

Run()

PrintHeadline()

HandleDefault(_)

HandleHelp(*args*)

HandleInfo(*args*)

HandleConfiguration(*args*)
    Handle 'configure' command.

HandleSelection(*args*)
    Handle 'select' command.

HandleAddSolution(_)

HandleListSolution(_)

HandleRemoveSolution(*args*)

HandleListProject(*args*)

HandleQueryConfiguration(*args*)

HandleListTestbenches(*args*)

HandleActiveHDLSimulation(*args*)

HandleGHDLSimulation(*args*)

HandleISESimulation(*args*)

HandleModelSimSimulation(*args*)

HandleAnyMentorSimulation(*args*)

HandleRivieraPROSimulation(*args*)

HandleQuestaSimSimulation(*args*)

HandleVivadoSimulation(*args*)

HandleCocotbSimulation(*args*)

HandleListNetlist(*args*)

HandleISECompilation(*args*)

HandleCoreGeneratorCompilation(*args*)

HandleXstCompilation(*args*)

HandleIpCatalogCompilation(*args*)

HandleVivadoCompilation(*args*)

HandleQuartusCompilation(*args*)

static GetAttributes(*method*)

GetMethods()

HandleLSECompilation(*args*)

static HasAttribute(*method*)

Log(*entry*, *condition=True*)
    Write an entry to the local logger.

LogDebug(**args*, *condition=True*, ***kwargs*)

LogDryRun(**args*, *condition=True*, ***kwargs*)

LogError(**args*, *condition=True*, ***kwargs*)

LogFatal(**args*, *condition=True*, ***kwargs*)

LogInfo(**args*, *condition=True*, ***kwargs*)

LogNormal(**args*, *condition=True*, ***kwargs*)

LogQuiet(**args*, *condition=True*, ***kwargs*)

LogVerbose(**args*, *condition=True*, ***kwargs*)

LogWarning(**args*, *condition=True*, ***kwargs*)

Logger
    Return the local logger instance.

MainParser

SubParsers

Part IV

Appendix

CHAPTER 7

Change Log

CHAPTER 8

Index

Python Module Index

p

Index