Open-Source Verification of digital ASIC/FPGA circuits

Stefano Minigutti

SyoSil, 19/06/2024

Agenda

- Who are we?
- Open-Source Verification (OSV) framework
- Functional Verification with pyUVM
- OSV tools
- UVM testbench in Python:
  - Vertical reuse
  - Randomization
  - Coverage collection and report
  - Differences and limitations compared to System Verilog (SV)
- Strengths and advantages of Open-Source
- Course at the Technical University of Denmark (DTU) and White Paper
- The road ahead

Who are we?

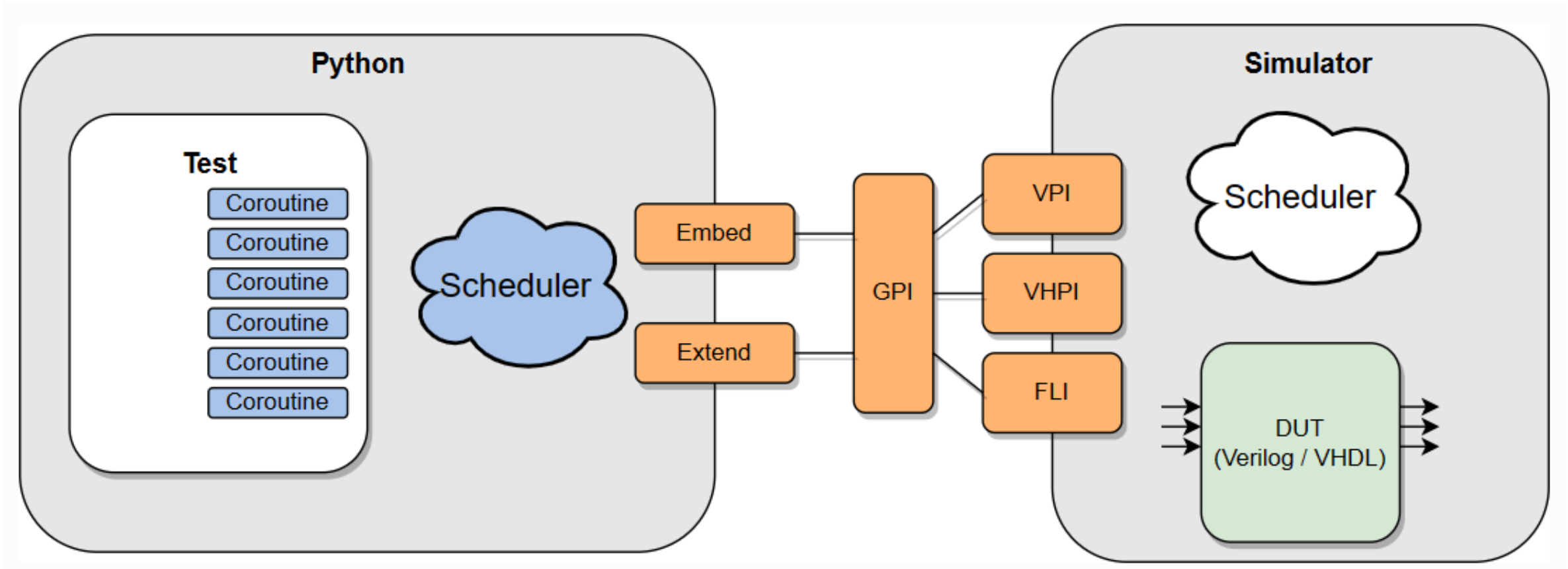
- Leading Design and Verification Company in EU
- 15+ years of experience in ASIC/FPGA Design and Verification
- **Team**: 45+ engineers
- **Services**:
  - **Products**: IPs and VIPs Development
  - **Consultation**: Consultancy services in design and verification
  - **R&D:** Internal & university education & research projects
- Where to find us www.syosil.com

Open-Source verification framework (1/3)

Cocotb (Co-routine-based Co-simulation Testbench) is an Open-Source verification framework used for testing digital designs in Python.

- **Python-Based**: Leverages Python's simplicity and flexibility for testbench development.
- **Co-Simulation**: Enables interaction between SystemVerilog or VHDL simulations and Python testbenches.
- **Co-routine-Based**: Utilizes Python's co-routine features for concurrent and asynchronous testing.
- **Open-Source**: Cocotb is freely available under the Apache License 2.0, fostering community collaboration and contributions.

Open-Source verification framework (2/3)



Open-Source verification framework (3/3)

- **PyUVM** is an Open-Source implementation of the Universal Verification Methodology (UVM) in Python.
- **Key Features:**
  - Based on the IEEE 1800.2 UVM specification
  - Python-Based
  - Uses cocotb for the interaction with the simulator
  - Object-Oriented
  - Ease of Use
  - Community Support

Functional Verification with pyUVM

- Implementation of Universal Verification Components (UVCs)
- Register model support
- Testcases development
- Randomization and coverage collection
  - Cocotb-coverage
  - PyVSC
- Reference model
- Challenges:
  - Device Under Test (DUT) to TestBench (TB) connection
  - No support for concurrent assertions

Open-Source verification tools

- Simulators:
  - Icarus Verilog
- Formal verification:
  - SymbiYosys
- Waveform viewer:
  - GTKWave

- Other tools are available, these are the ones which we have explored at the moment

PyUVM testbench (1/3)

- Structure of TB is very similar in PyUVM compared to SV-UVM
- Importing all of PyUVM
  - from pyuvm import *
- Names of base classes are as in SV
- Almost all components and objects from UVM also exists in PyUVM

```
import pyuvm
from pyuvm import *

@pyuvm.test()
class test(uvm_test):

   def build_phase(self):
      super().build_phase()
      self.env = environment.create("env",self)

   async def run_phase(self):
      self.raise_objection()

      await super().run_phase()

      self.drop_objection()
```

PyUVM testbench (2/3)

- UVM Factory
  - Components and objects can be instantiated in two ways
    - Using *create* makes override possible

```
inst1 = component("name", parent)
inst2 = component.create("name", parent)
```

  - Factory override support

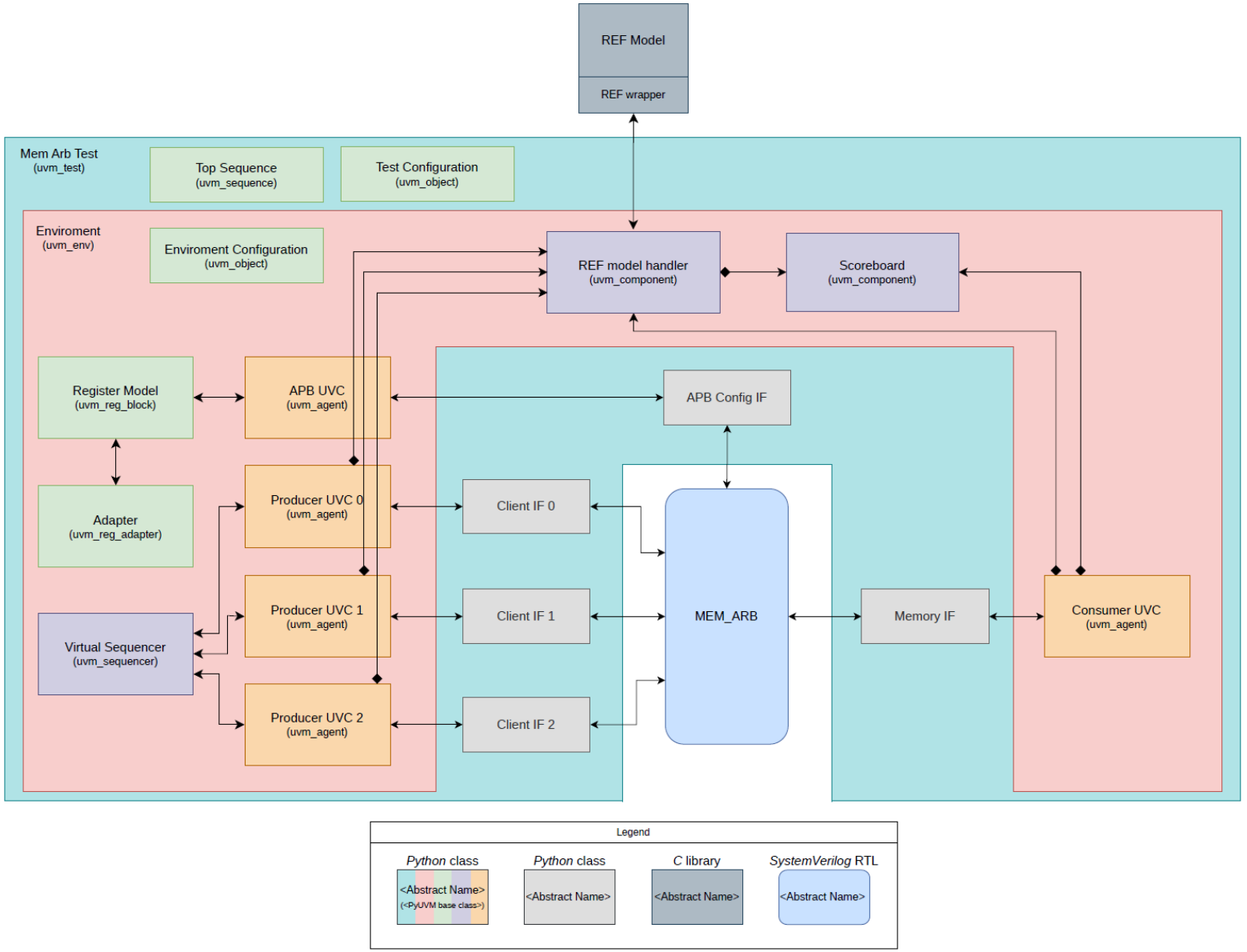
```
uvm_factory().set_type_override_by_type(cl_seq_basic, cl_seq_simple)
```

- All components have a build-in logger
  - With 5 logging levels: *critical*, *error*, *warning*, *info* and *debug*
  - Configurable from command line argument

```
self.logger.info("message")
self.logger.error("error message")
```

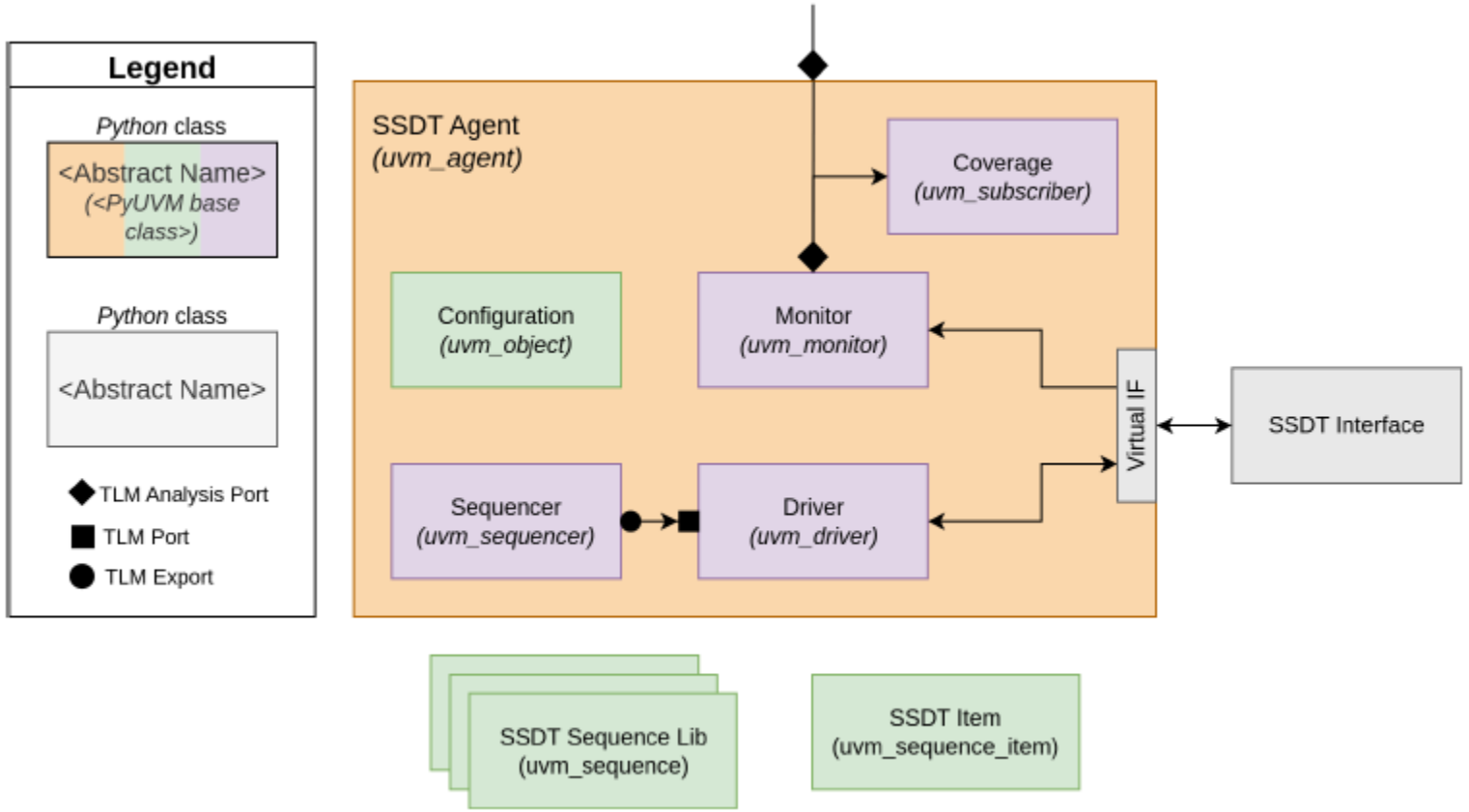
- ConfigDB support
  - Build-in exception if entry does not exist

PyUVM testbench (3/3)



Vertical reuse (1/2)

- Universal Verification Component (UVC)
- pyUVM UVC for the simple SyoSil Data Transfer (sSDT) protocol



Vertical reuse (2/2)

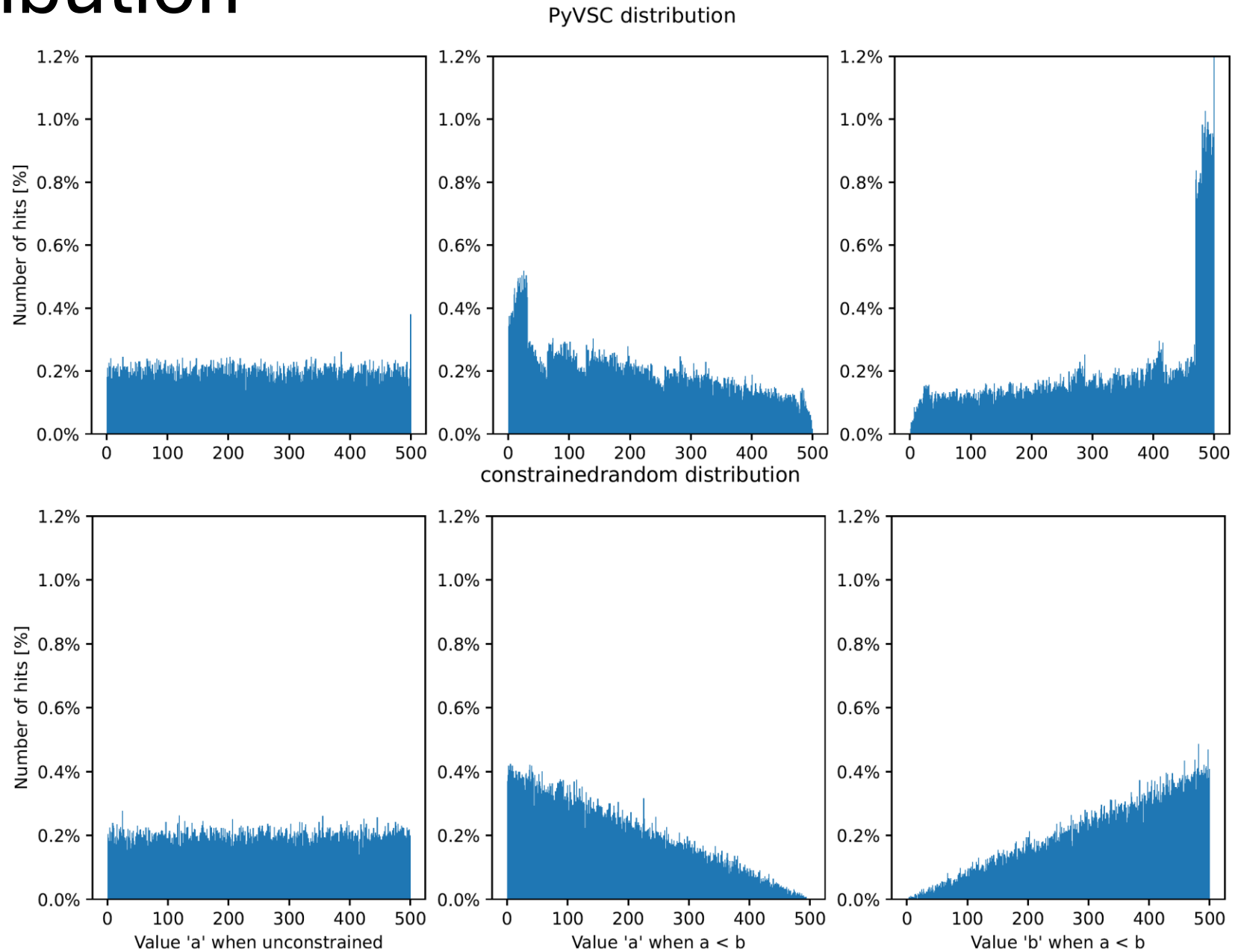
- Interfaces
  - Not available as part of pyUVM library
  - Necessary for UVC reusability
- New interface class was defined
  - Including method to connect DUT signals
  - Instantiated in the base_test
  - Stored in the configDB
- Used by the UVC to access DUT signals

- Cocotb busses is an alternative
  - Presents some limitations
  - Heavily relies on configDB

Randomization

- Crucial for Constrained Random Verification (CRV)
- Multiple existing libraries:
  - Cocotb-coverage
    - Both randomization and coverage support
    - Cannot randomize nested objects
    - Cannot randomize dynamic arrays
  - pyVSC
    - Both randomization and coverage support
    - Supports all basic randomization necessities
  - Constrainedrandom
    - Only randomization support
    - Supports all basic randomization necessities

Distribution



Coverage collection

- Same libraries as randomization
  - Cocotb-coverage
    - Does not support per_instance coverage
  - pyVSC
    - Binsof and intersect not supported in crosses
    - “with” clause not supported
    - Only procedural sampling

```
@vsc.covergroup
class my_covergroup(object):
     def __init__(self, a :callable):

         self.cp1 = vsc.coverpoint(a, bins={
             "a" : vsc.bin(1, 2, 4),
             "b" : vsc.bin(8, [12,15])
             })
```

```
@vsc.covergroup
class my_covergroup(object):
     def __init__(self, a : callable):

         self.cp1 = vsc.coverpoint(a, bins={
             "a" : vsc.bin_array([], 1, 2, 4),
             "b" : vsc.bin_array([4], [8,15])
             })
```

Open-Source strengths

- **Cost-effectiveness**: Lower or no licensing fees.
- **Community collaboration**: Knowledge sharing and collective problem-solving.
- **Flexibility and customization**: Tailoring tools to specific needs.
- **Transparency and trust**: Access to source code for inspection.
- **Vendor independence**: Freedom from vendor lock-in and proprietary ecosystems.

Course at the Technical University of Denmark

- SyoSil has started a 3-weeks course at the Technical University of Denmark (DTU)
- Prototype, possibility to become an official course in 2025
- The course focuses on:
  - Verification concepts
  - Cocotb and pyUVM introduction
  - TB development using python, including:
    - UVCs integration
    - Coverage
    - Reference model and Scoreboard
- Part of the EDU4Chip project
  - https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/how-to-participate/org-details/884004457/project/101123086/program/43152860/details

White paper & source code

- Focuses on:
  - Open-Source verification tools
  - Test Bench development using Python
  - Advantages and limitations
  - Our experience and conclusions
  - Soon available at SyoSil's LinkedIn page:
    - https://www.linkedin.com/company/syosil/posts/?feedView=all

- An example of a pyUVM TB we developed is available on GitHub:
  - https://github.com/syosil/pyuvm_tb_example

The road ahead

- Verilator as the simulation tool
- More advanced and complicated coverage scenarios
- Investigate randomization distribution
- TLM2.0 support
- SystemC reference model integration
- Formal Verification with Open-Source tools

Our conclusions

- Works for smaller testbenches
  - Nice for teaching
- Difficult to use for complicated scenarios
  - Randomization distribution might be a problem
  - Coverage collections has limitations