2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
OCTOBER 14-15, 2025

FPGA Firmware Verification: a common approach for simulation and hardware tests

S. Pavinato, E. D'Costa, S. Gabourin

European Spallation Source

accellera
SYSTEMS INITIATIVE

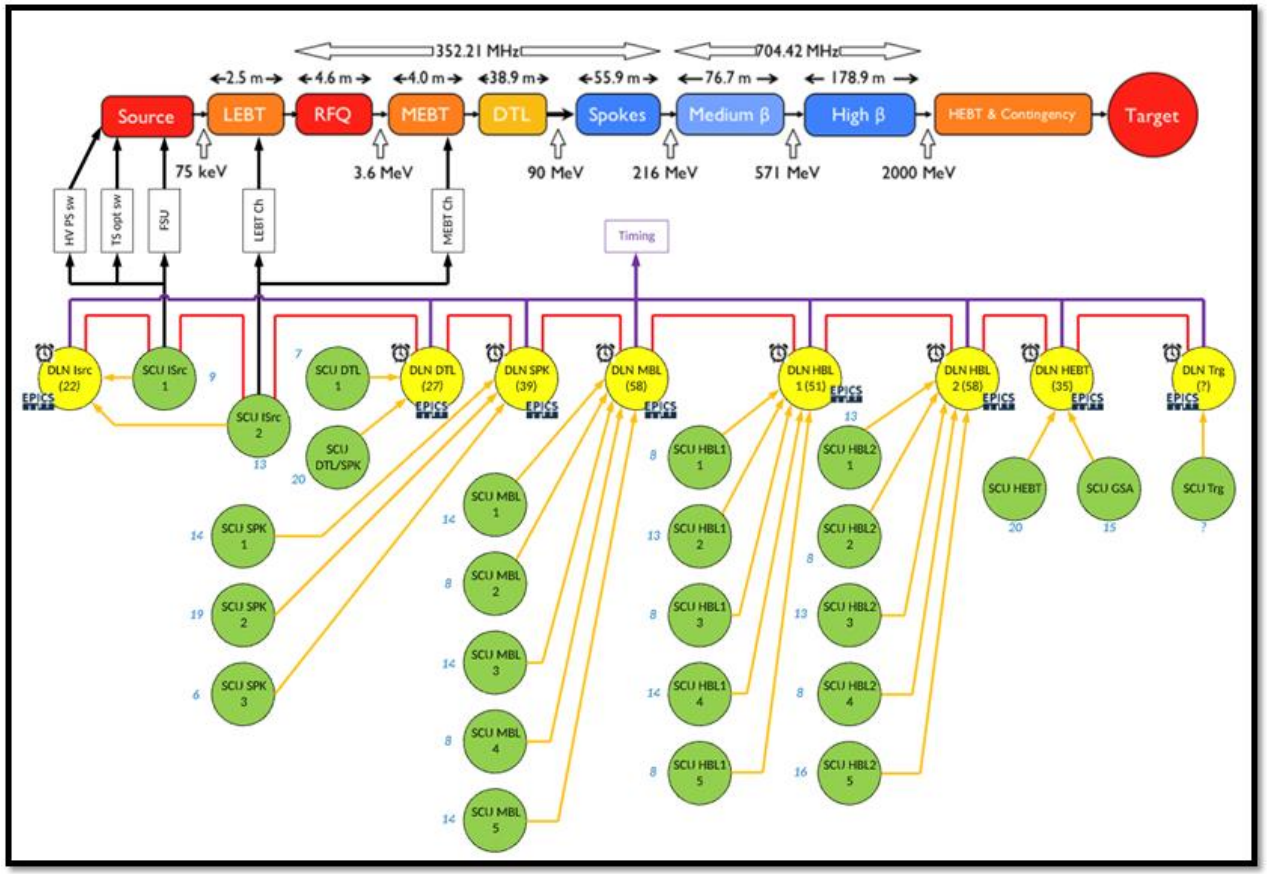
Context

- **ESS** (European Spallation Source): the most powerful neutron source ever built. (Lund, Sweden)
- **FBIS** (Fast Beam Interlock System): protects the machine gathering signals sensor systems (FPGA and PLC based) and acts on actuators
- **FPGA** + **VHDL**: core of the FBIS

FBIS Architecture

- **DLN**(mTCA) -> Real logic
- **SCU**(cPCI) -> Interfaces sensor systems

- ~30 different firmware.



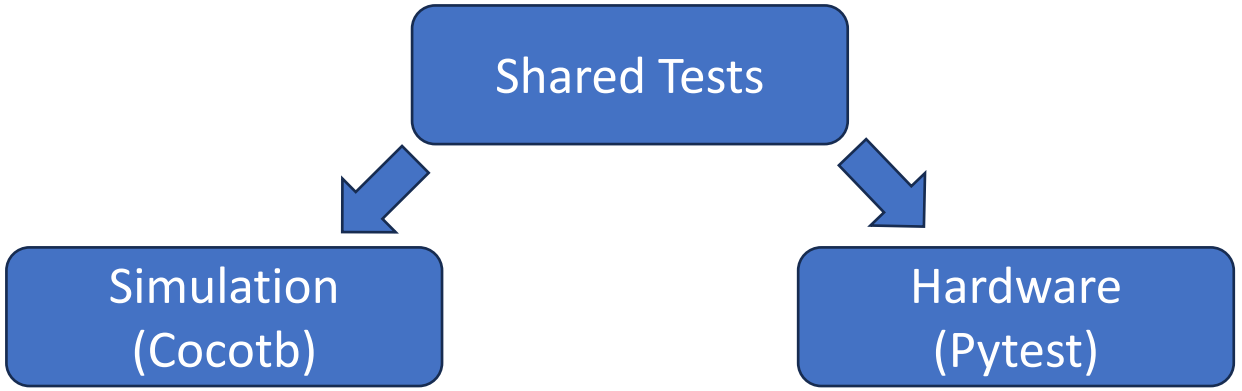
Gap between simulation & hw validation

Simulation	Hardware validation
Unit/Integration/System Tests	(Factory) Acceptance Tests
UVM, VUnit, OSVVM, **Cocotb**	Vendor tools, HIL, analyzers, custom boards, high-level framework(i.e. **Python**)
Test reuse difficult among the two environments.	

Key Challenges

- Reduce gap between **simulation** <-> **hardware**
- Apply concepts from **UVM** & **PSS**
- Lower the entry barrier.

Our Approach



- Focus on common tests only
- Exploit the UVM **modularity** (driver, monitor, scoreboard, environment)
- PSS inspired, stimuli **portability**
- Use **python** as glue

Hardware Tester

- Inputs -> DLN/SCU FPGAs -> Outputs
  - Ethernet, Profinet, Manchester Encoder, Aurora Cores, Access the FPGA registers.
- EPICS. (software infrastructure used in building distributed control as Particle Accelerators)
  - It allows to abstract data exchange with heterogeneous devices, regardless of the drivers or protocols they interface with, as standardized process variables. Thanks to this uniformity, high-level tools (Python) can interact with all devices by the same mechanism.
- **Pytest** Automation

Cocotb & Pytest

- **Cocotb 2.0** -> pytest style.
- **Parametrization** -> uniform testing
- **CI/CD integration** -> JunitXML reports

```python
@cocotb.test()
@parametrize(a = [2,4,5])
async def test_dummy(dut, a):
    assert a % 2 == 0, "Not even"
```

Cocotb parametrized test.

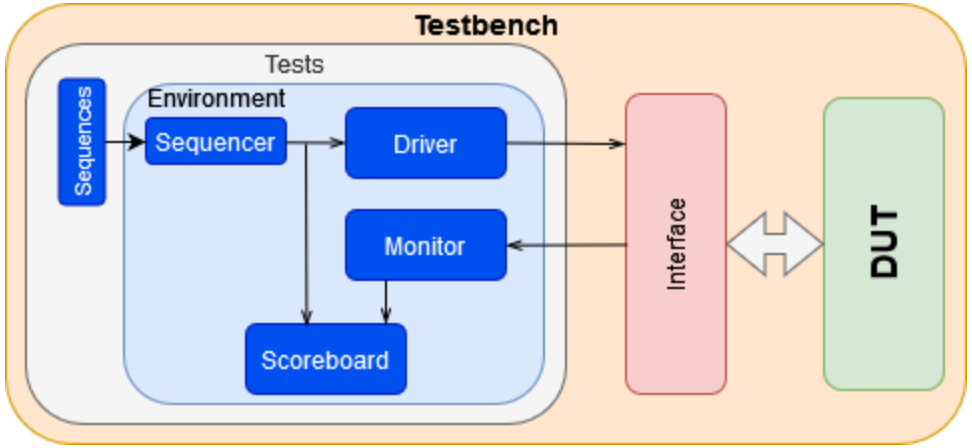
```python
@mark.parametrize("a" , [2,4,5])
def test_dummy(a):
    assert a % 2 == 0, "Not even"
```

Pytest parametrized test.

```xml
<testsuites>
  <testsuite name="pytest" errors="0" failures="1" skipped="0" tests="3">
    <testcase classname="test" name="test_dummy[2]" time="0.000"/>
    <testcase classname="test" name="test_dummy[4]" time="0.000"/>
    <testcase classname="test" name="test_dummy[5]" time="0.001">
      <failure message="AssertionError: Not even assert (5 % 2) == 0"/>
  </testcase>
  </testsuite>
</testsuites>
```

JUnit XML report.

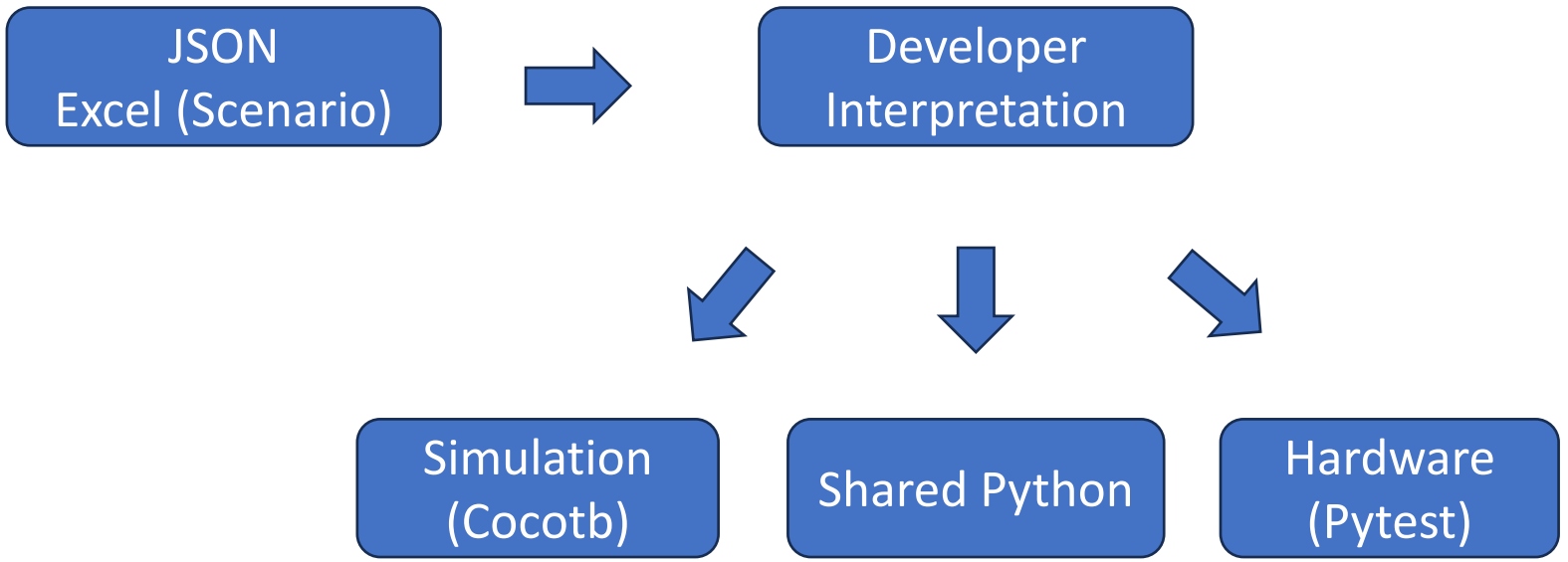
UVM Concepts Applied



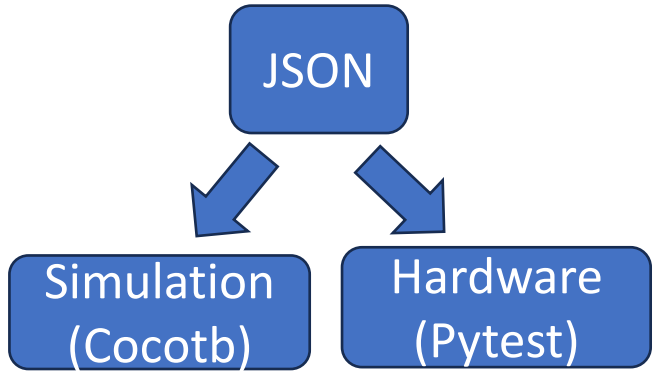

```python
class Environment:
    def __init__(self, driver_func, monitor_func, Scoreboard_class):
        self.driver = driver_func
        self.monitor = monitor_func
        self.scoreboard = Scoreboard_class

        if self.__class__.run_check_phase is Environment.run_check_phase:
            raise NotImplementedError(f"{self.__class__.__name__} must be overriden")


    def run_check_phase(self, *args, **kwargs):
        raise NotImplementedError("Subclasses must implement run_check_phase.")
```

- Same structure across simulation and hardware.
  - Driver and Monitor tied to the environment. Same scoreboard.
- Factory mechanism -> override & inheritance.

PSS Concepts Applied



Case Study: RISC-V Monitoring




```
"test_name": "DDS_21_5_rflps",
"data": {
  "input": [
    { "name": "slot", "type": "int", "domain": [0, 1, 2, 3] },
    { "name": "port", "type": "int", "domain": [0, 1] },
    { "name": "signal", "type": "string", "domain": ["Beam Permit", "Redundant Beam Permit"]
    { "name": "slot_value", "type": "string", "domain": ["OK", "NOK"] },
    { "name": "signal_value", "type": "string", "domain": ["OK", "NOK"] }],
```

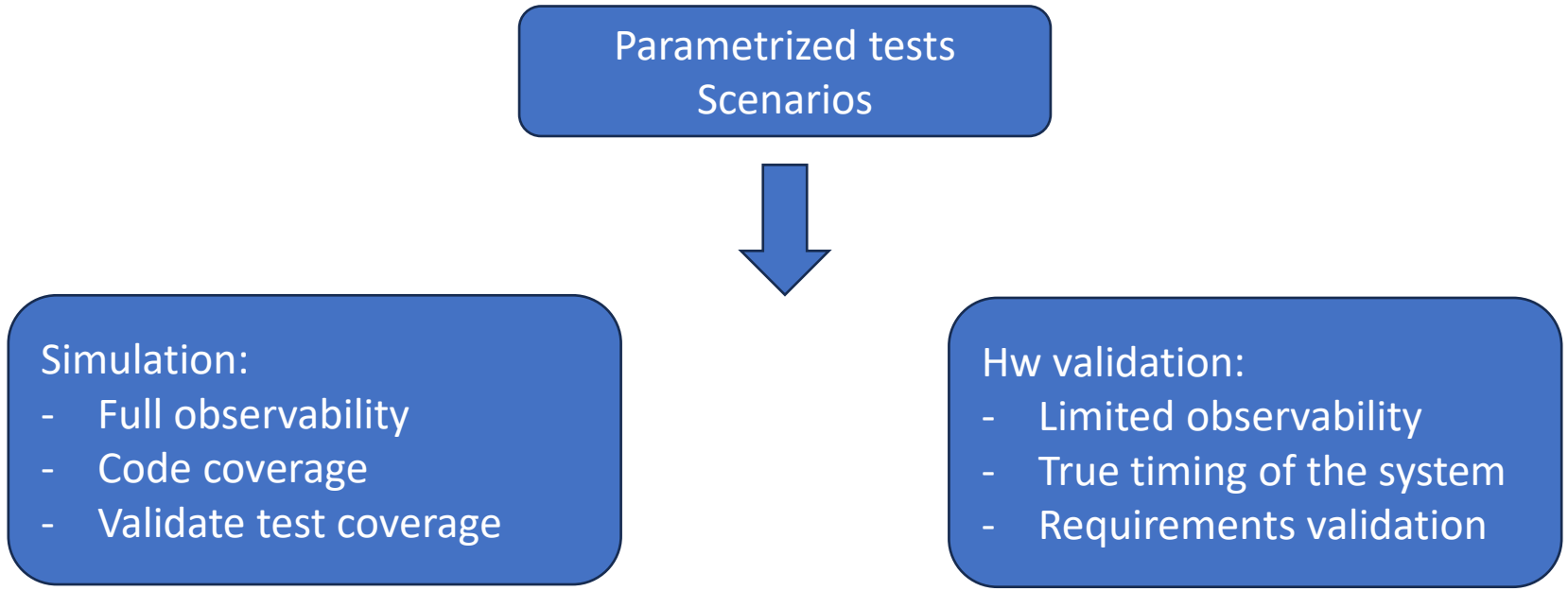
```
@cocotb.test()
@parametrize((("scu", "slot", "port"),  ru.get_system_params(RFLPS)),
             slot_value = [OK, NOK],
             signal = ["Beam Permit", "Redundant Beam Permit"],
             signal_value = [OK, NOK])
async def DDS_21_5_rflps(dut, scu, slot, port, slot_value, signal, signal_value):
    shift = (port-1) * 8 + ((get_data_type("scu_rflps_discrete_signals_index")[signal]-1)*2)
    await environment.run_check_phase(slot, port, slot_value, signal_value, shift)
```

```
@pytest.mark.rflps
@parametrize( serializer = ["SerializerB"],
              slot_value = [OK, NOK],
              signal = ["Beam Permit", "Redundant Beam Permit"],
              signal_value = [OK, NOK])
def DDS_21_5_rflps(self, serializer, scu, slot, port, slot_value, signal, signal_value):
   translated_signal = get_data_type("scu_discrete_signal_translator")[signal]
   test_environment.run_check_phase(scu,serializer,mc_type,slot,port,translated_signal,s
```

Verification Metrics

- Requirements Traceability Matrix (RTM)
- Scenario Coverage via parameterized tests.
- Adequate code coverage.

Transition simulation <-> hardware



Some Limitations

- JSON as DSL
- PSS not usable with cocotb
- ~~Cocotb 2.0 lacks verification libraries~~ (2025-09-20)
- FAT relies on EPICS
  - absence demands custom drivers and abstraction.

Benefits

- **Reuse** across simulation & hardware
- Improved **FAT** testing
- Common **reporting & traceability**
- Easier **collaboration** & **understanding**

Conclusions

- Smooth transition simulation <-> hardware.
- UVM/PSS adopted, not imposed
- Common language, style, libraries, reports.

Questions


python
simulation
verification
test
hardware
testing
coverage
environment
environments
firmware
fpga
cocotb
pytest
approach
requirements
systems
scenarios
json
implementation
validation
system
code
scoreboard
tests
epics
development
methodologies
traceability
specification
functionalities
common
unified
uvm
framework
monitor
components
python-based
pss
modular
fbis
data
design
parametrized
interlock
implemented
developed
libraries
language
factory
scoreboards
sensor
key
run check phase
reusable
stimuli
function
context
machine
concepts
conditions
drivers
definition
heterogeneous
format
libraries
shared
sharing
standardized
outputs
equipped
paper