# Firmware Integration Testing with Python

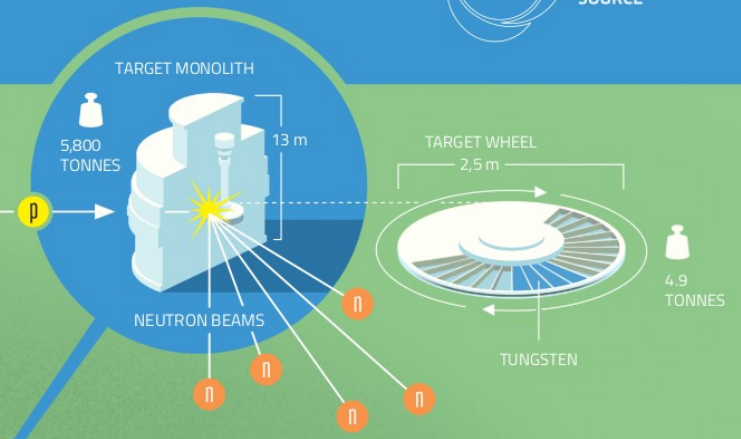
CHRISTIAN AMSTUTZ

7.12.20023

# European Spallation Source

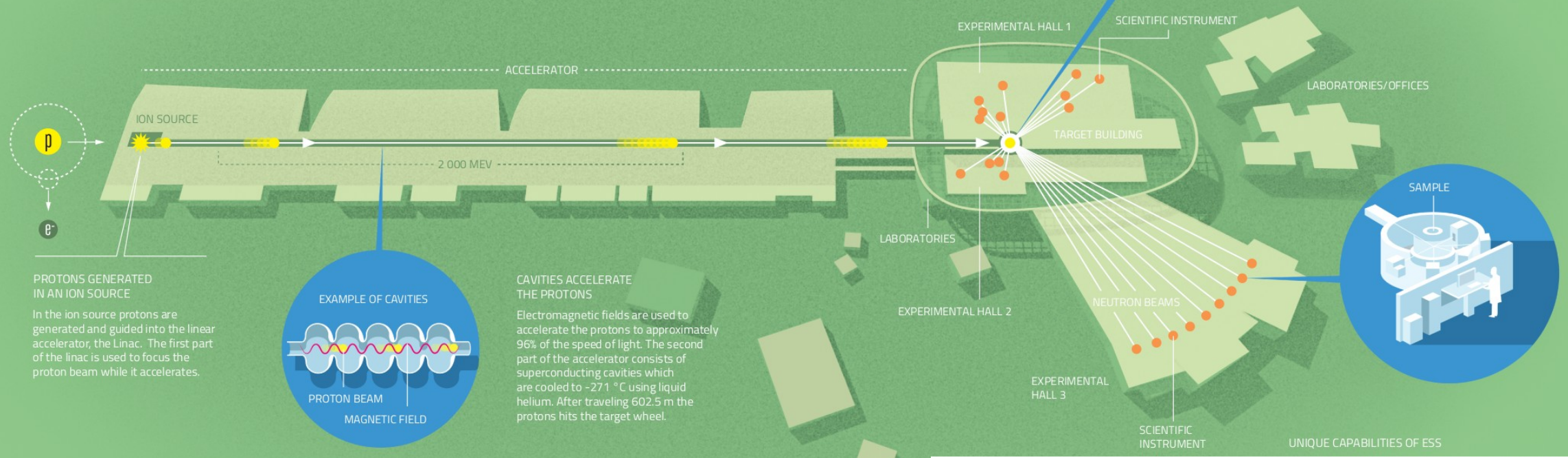
EUROPEAN SPALLATION SOURCE

The European Spallation Source (ESS) is a multi-disciplinary research centre based on the world's most powerful neutron source. ESS will give scientists new possibilities in a broad range of research, from life science to engineering materials, from heritage conservation to magnetism. ESS is a pan-European project, with Sweden and Denmark serving as host countries. The main research facility is being built in Lund, Sweden, and the Data Management and Software Centre (DMSC) is located in Copenhagen, Denmark.


THE TARGET IS THE NEUTRON SOURCE

When the accelerated protons hit the rotating tungsten target wheel spallation occurs and neutrons are scattered from the tungsten nucleus. The more neutrons produced and collected in the target, the "brighter" the neutron source. The neutrons are directed through moderators and neutron guides to the scientific instruments where they are used for experiments. The Target monolith consists of the Target wheel, moderators, cooling systems and shielding and weighs approximately 5,800 tonnes.




PROTONS GENERATED IN AN ION SOURCE

In the ion source protons are generated and guided into the linear accelerator, the Linac. The first part of the linac is used to focus the proton beam while it accelerates.




CAVITIES ACCELERATE THE PROTONS

Electromagnetic fields are used to accelerate the protons to approximately 96% of the speed of light. The second part of the accelerator consists of superconducting cavities which are cooled to -271 °C using liquid helium. After traveling 602.5 m the protons hits the target wheel.

TOTAL BUILDING AREA 65 000 m²

The ESS facility will be approximately 650 metres in total length. The target building will be 125 metres long, and about 30 metres high. The 537-meter-long accelerator tunnel is built underground and will be covered with soil.

| | |
|---|---|
| Concrete: | 50 000 m³ |
| Rebar: | 6 000 tonnes |
| Pipes: | 40 km |
| Cables: | 2,000 km |
| Total volume: | 400,000 m³ |

602.5 m

UNIQUE CAPABILITIES OF ESS

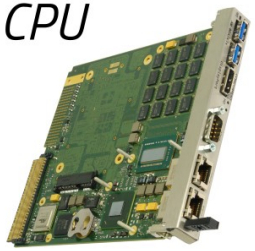
Availability goal: >95%

# Low-level RF Crate

*Crate Controll*

NAT MCH-PHYS

*CPU*



Concurrent Technologies
AM 90x AMC CPU

*Timing*

MRF EVR-300U

***Field Control***

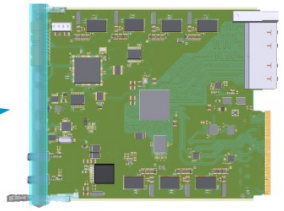
Struck DWC8VM1

Struck SIS8300-KU

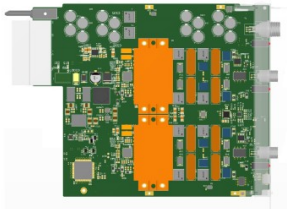
*Power*

Wiener - 1000 W

*Piezo Control*



PEG RTM Carrier



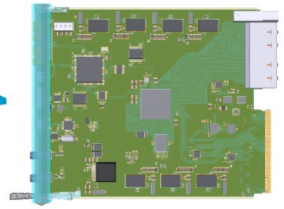
PEG Piezo Driver

Schroff 12 slot MTCA crate

*Interlock*

IOxOS IFC1410

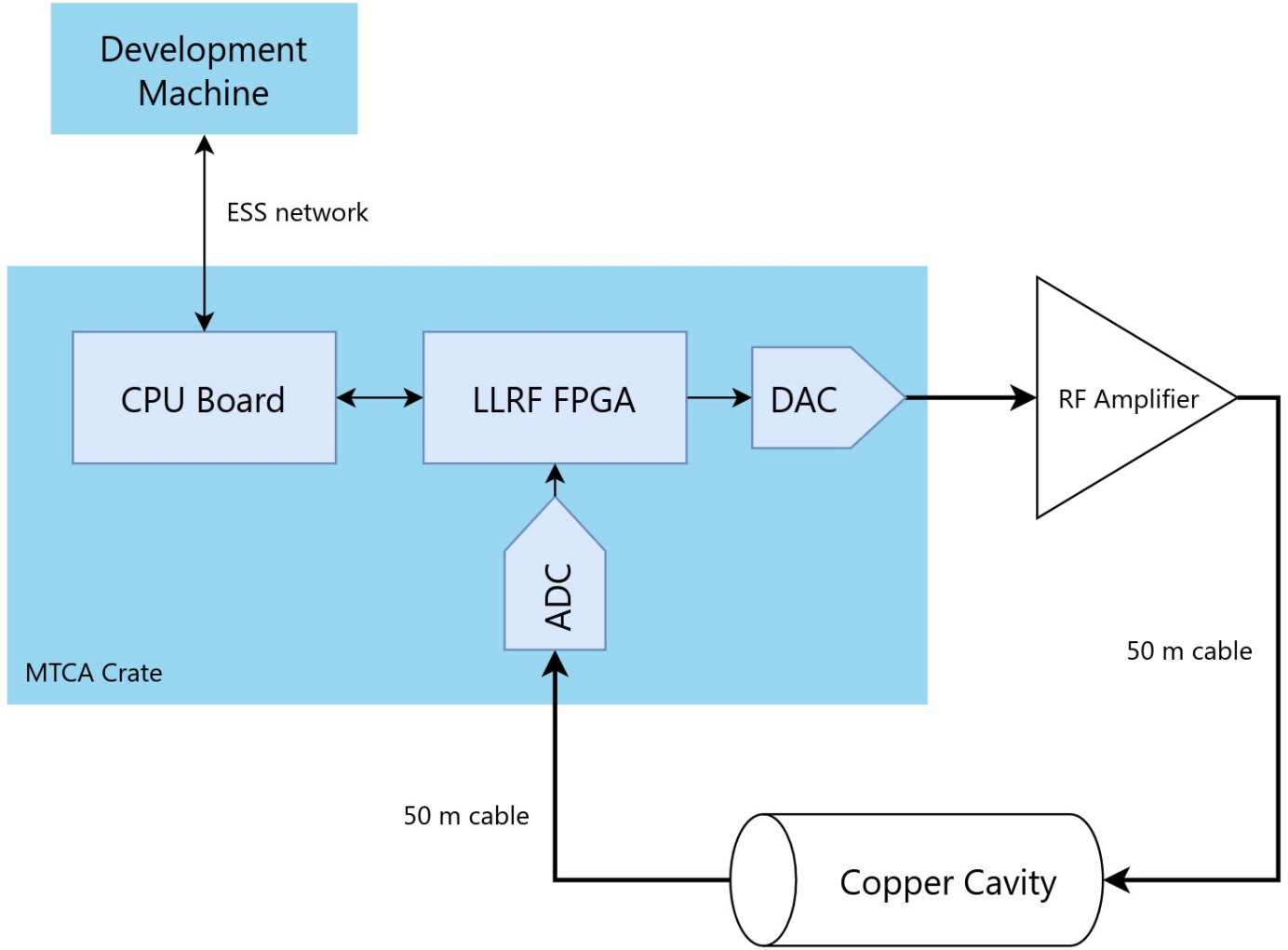
*Frequency Generation*



PEG RTM Carrier

LO RTM

# Low-Level RF Testbench




# Testing

- Full test coverage for every version
- Repeatability
- Reusability of tests

# Testing

- Full test coverage for every version
- Repeatability
- Reusability of tests

→ **Automated testing**

**What is needed?**

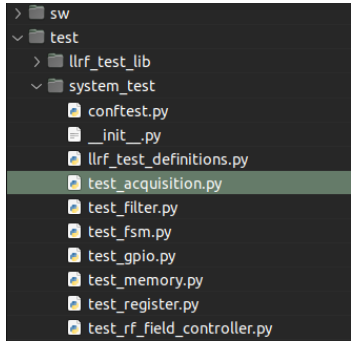
- FPGA design that aids testing
- Language to implement tests
- Test framework
- Method to access parameters (registers) of the system
- Management of a complex system

# Python!


- Widely used
- Non-FPGA people also can contribute to testing
- Many libraries freely available
- Interaction with scopes, signal generators
- Code re-use with FPGA-firmware verification: cocotb

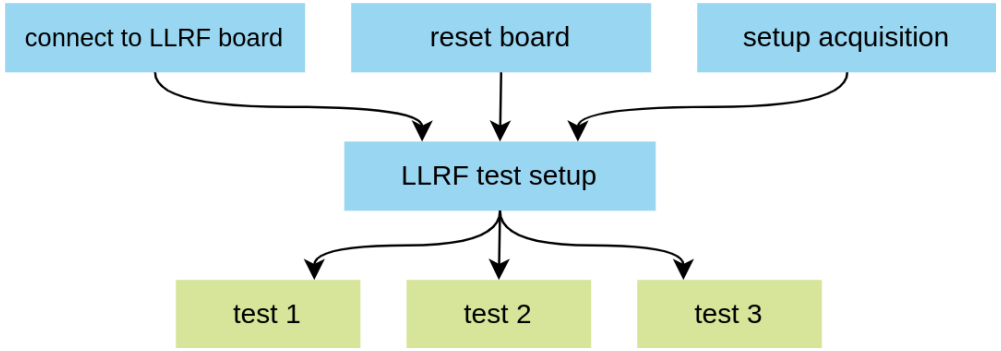
# pytest – Python test framework

- Automatic test discovery

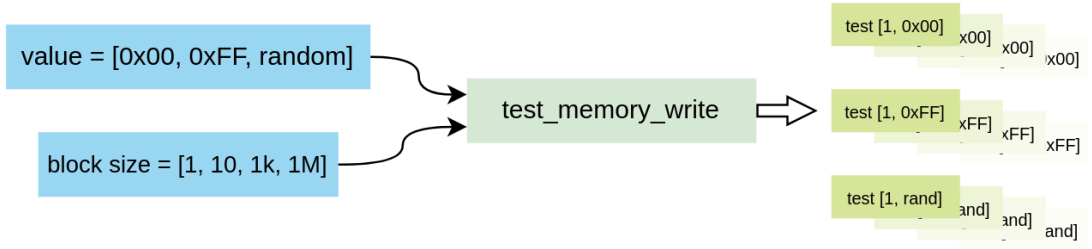



```python
import math
import random

from .llrf_test_definitions import MEM_SIZE


def test_ddr_write_single(test_board) -> None:
    for i in range(1000):
        mem_location = math.floor(random.random()*MEM_SIZE)
        test_board[(mem_location,1)].write(0x00)
        assert test_board[(mem_location,1)].read() == [0x00]
        test_board[(mem_location,1)].write(0xFF)
        assert test_board[(mem_location,1)].read() == [0xFF]

# Check write blocks
def test_ddr_write_blocks(test_board) -> None:
    BLOCK_SIZE = 2**12
    for i in range(10000):
        test_area = test_board[i*BLOCK_SIZE, BLOCK_SIZE]
```

- Modular test setups




- Test parametrization

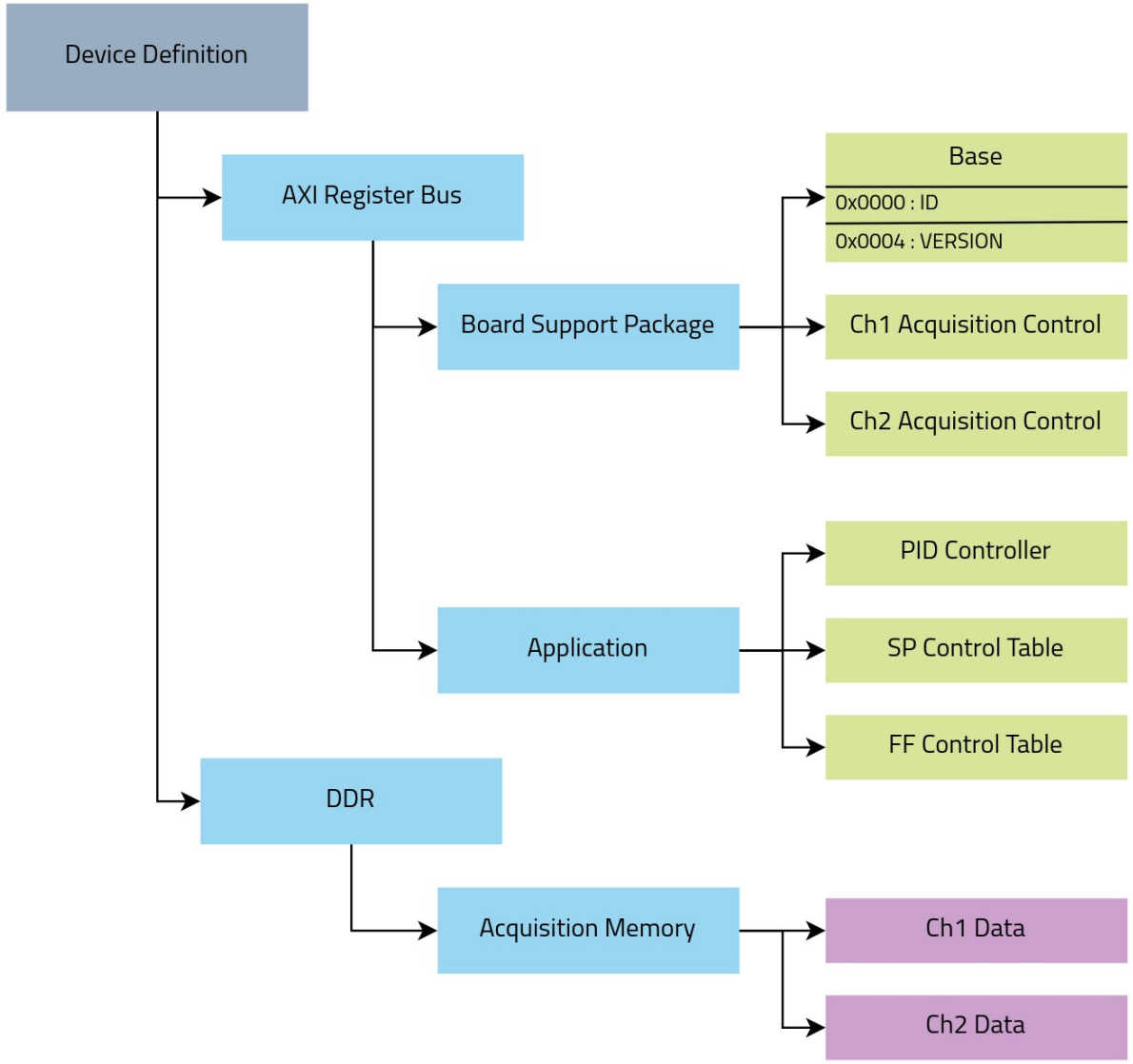



- Reporting

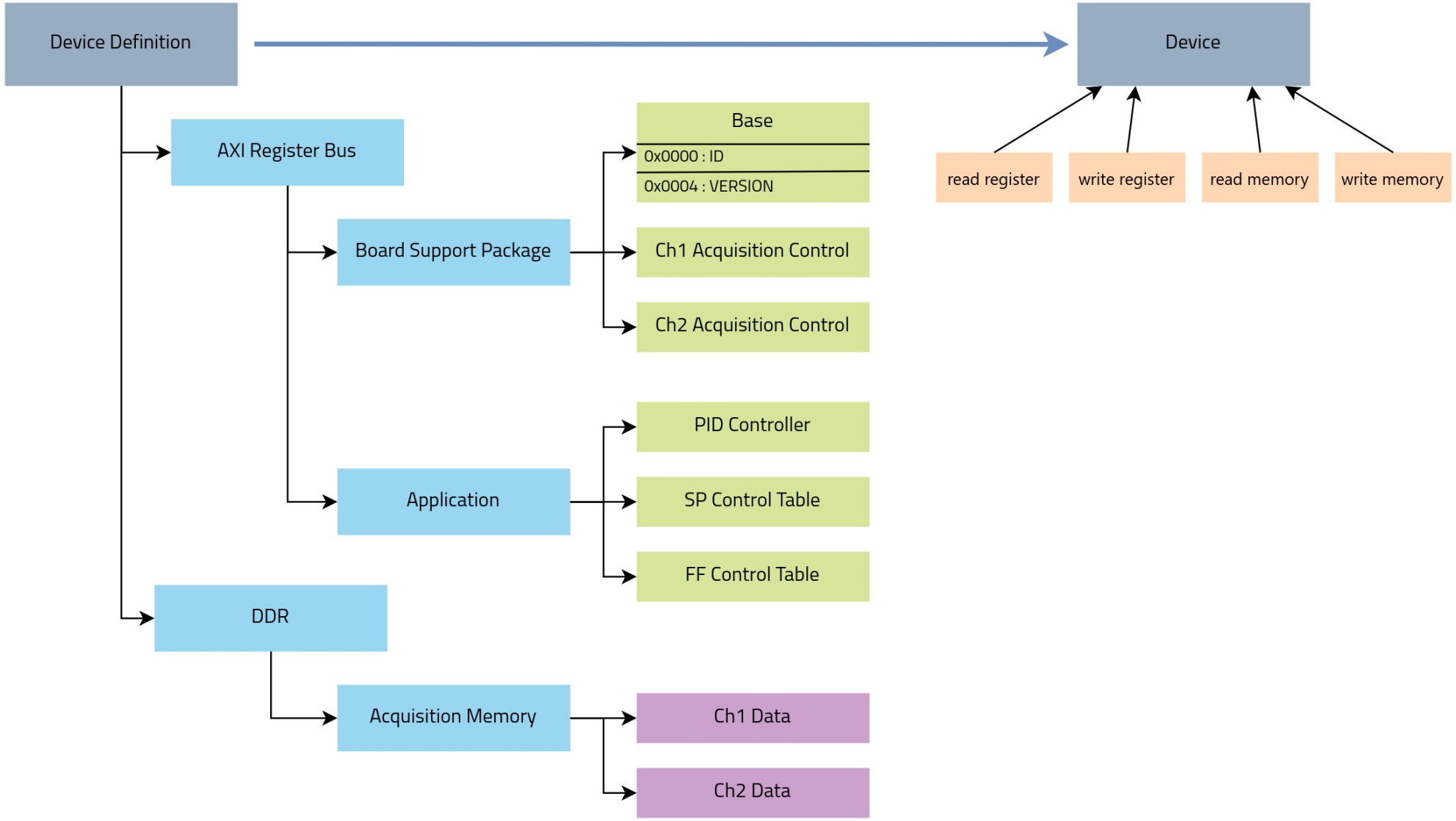
```
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_table_inactive[ff_table-33] - AssertionError: as...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_read_error[ff_table-33] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_load_error[ff_table-33] - AssertionError: assert...
==================== 475 failed, 8175 passed, 1936 skipped, 44 errors in 1277.89s (0:21:17) ====================
[iocuser@llrf-lion-cpu system_test]$
```


more information: https://docs.pytest.org

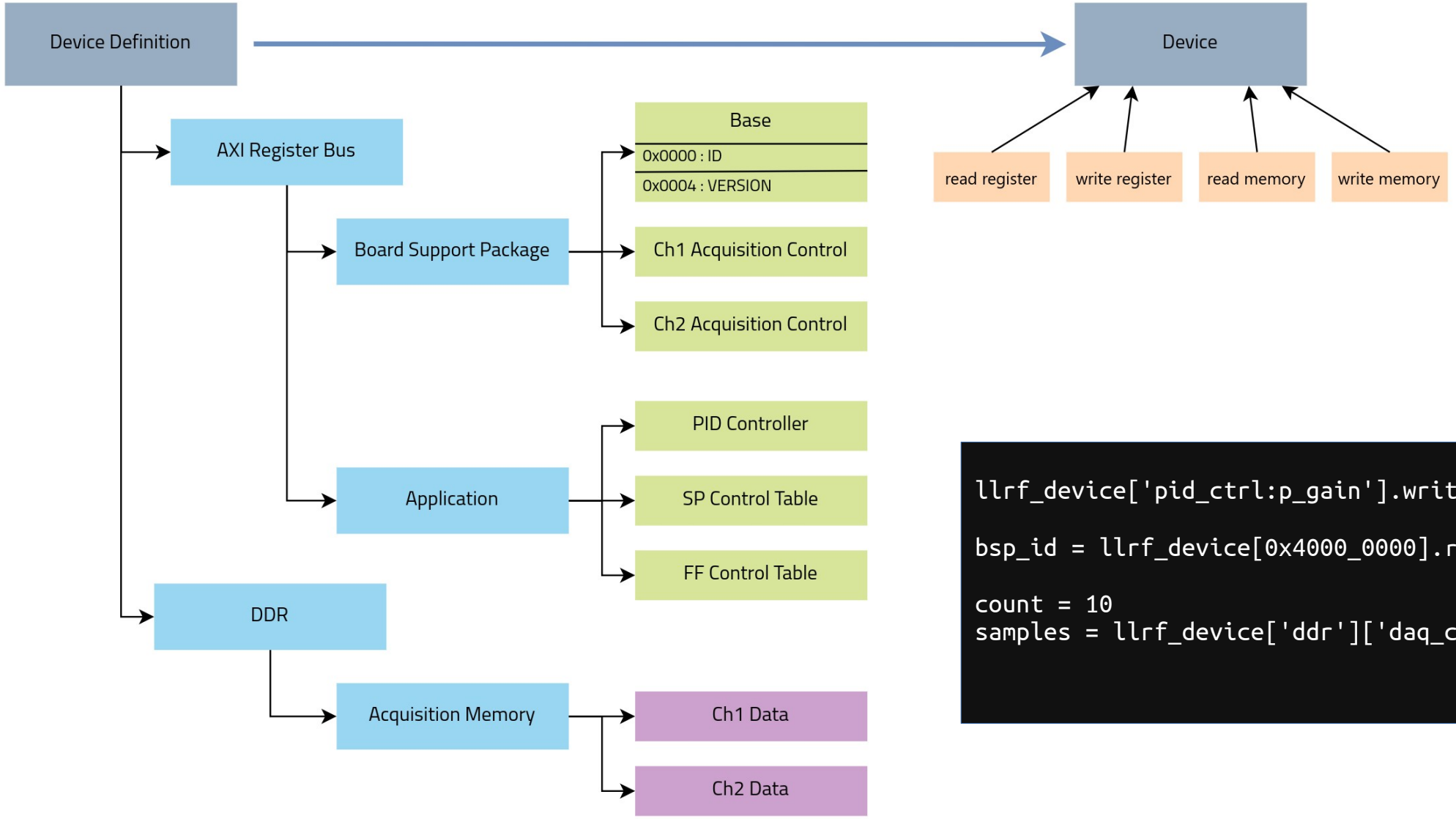
# lowlevhw – Python hardware handling



# lowlevhw – Python hardware handling



# lowlevhw – Python hardware handling



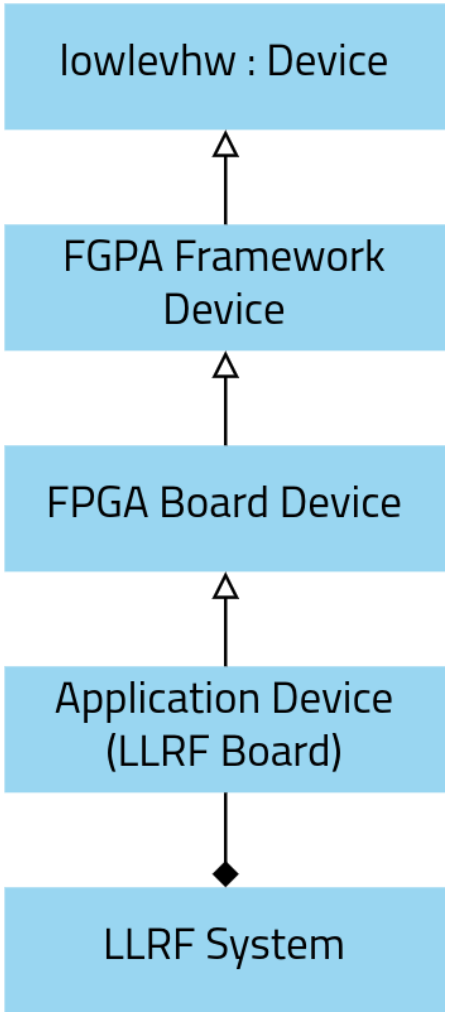

```
llrf_device['pid_ctrl:p_gain'].write(0.5)

bsp_id = llrf_device[0x4000_0000].read()

count = 10
samples = llrf_device['ddr']['daq_ch_1'].read(count)
```

# lowlevhw – it is more!


- Providing some test helpers
- Register bank generator
- Documentation generator: console, markdown
- C-header generator
- Prototyping of algorithms before the implementation in the control system

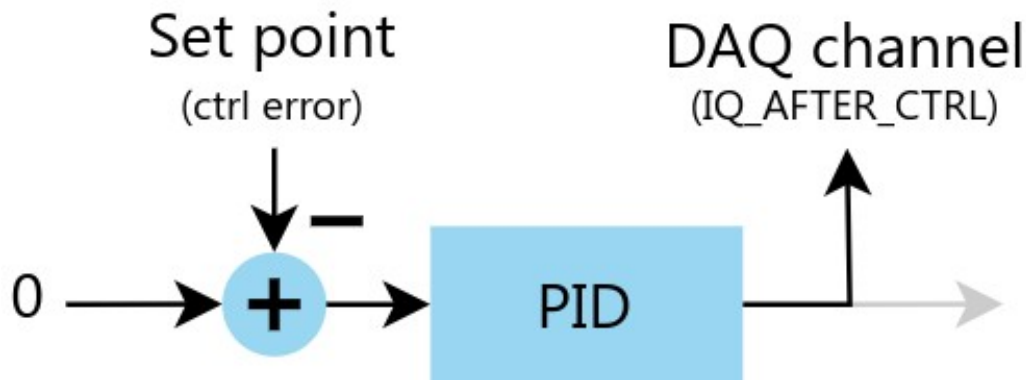
available at: https://gitlab.esss.lu.se/fpga/lowlevhw

# Device Model for reusablity



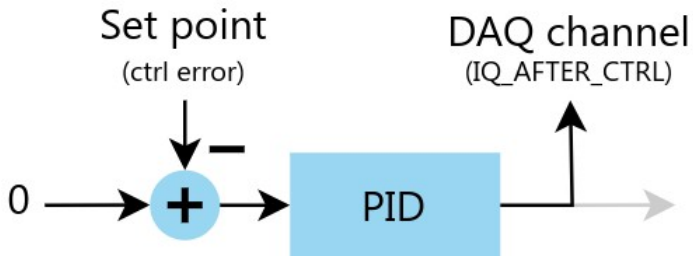

- Implementing low-level device access
- IP management
- Automatic IP detection in FPGA
- General functions shared with other projects
- Application-specific functions
- System level functionality, for example:
  - Management of multiple boards
  - control of timing receiver
  - access to EPICS IOC

# Example – P component of PID controller



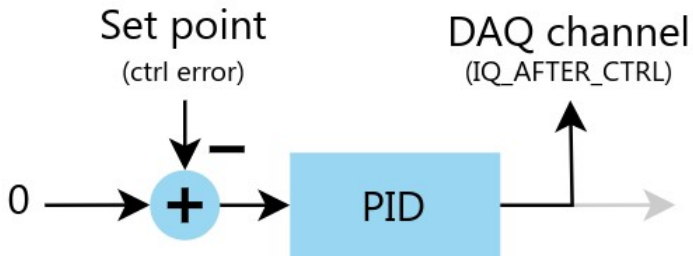
# Example – P component of PID controller

```
def test_k_p(self, idle_test_board, ctrl_error, k_p):

    # Local test setup
    dut = idle_test_board
    dut['LLRF_IQ_CTRL:IQ_CTRL_OUT_SOURCE'].write(0b111)
    dut['LLRF_SP_CONFIG:CONSTANT_EN'].write(1)
    dut['LLRF_PI_FIXED_SP:CONST_SP'].write(lowlevhw.float_to_fixed(ctrl_error, 1, 15))
    dut['LLRF_PI_K'].write(lowlevhw.float_to_fixed(k_p, 8, 2)
```



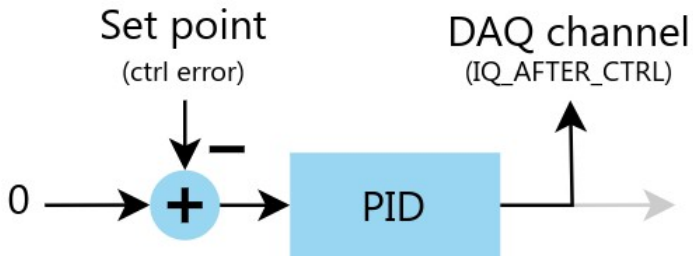
# Example – P component of PID controller

```
def test_k_p(self, idle_test_board, ctrl_error, k_p):

    # Local test setup
    dut = idle_test_board
    dut['LLRF_IQ_CTRL:IQ_CTRL_OUT_SOURCE'].write(0b111)
    dut['LLRF_SP_CONFIG:CONSTANT_EN'].write(1)
    dut['LLRF_PI_FIXED_SP:CONST_SP'].write(lowlevhw.float_to_fixed(ctrl_error, 1, 15))
    dut['LLRF_PI_K'].write(lowlevhw.float_to_fixed(k_p, 8, 2)


    # Execute test and acquire data
    dut.force_timing_pulse()
    time.sleep(0.01)
    measurement = dut['IQ_AFTER_CTRL'].read(test_length*4)
```



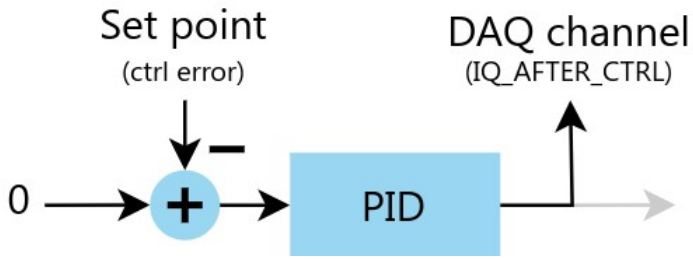
# Example – P component of PID controller

```
def test_k_p(self, idle_test_board, ctrl_error, k_p):

    # Local test setup
    dut = idle_test_board
    dut['LLRF_IQ_CTRL:IQ_CTRL_OUT_SOURCE'].write(0b111)
    dut['LLRF_SP_CONFIG:CONSTANT_EN'].write(1)
    dut['LLRF_PI_FIXED_SP:CONST_SP'].write(lowlevhw.float_to_fixed(ctrl_error, 1, 15))
    dut['LLRF_PI_K'].write(lowlevhw.float_to_fixed(k_p, 8, 2)


    # Execute test and acquire data
    dut.force_timing_pulse()
    time.sleep(0.01)
    measurement = dut['IQ_AFTER_CTRL'].read(test_length*4)


    # Calculate expected value
    expected_value = max(-2.0, min(1.99999, k_p * ctrl_error))
    expected = [(expected_value, 0.0)] * len(measurement)

    # Compare measurement with expected value
    assert_waveform_allclose(measurement, expected, shifttol=0, atol=TOLERANCE)
```

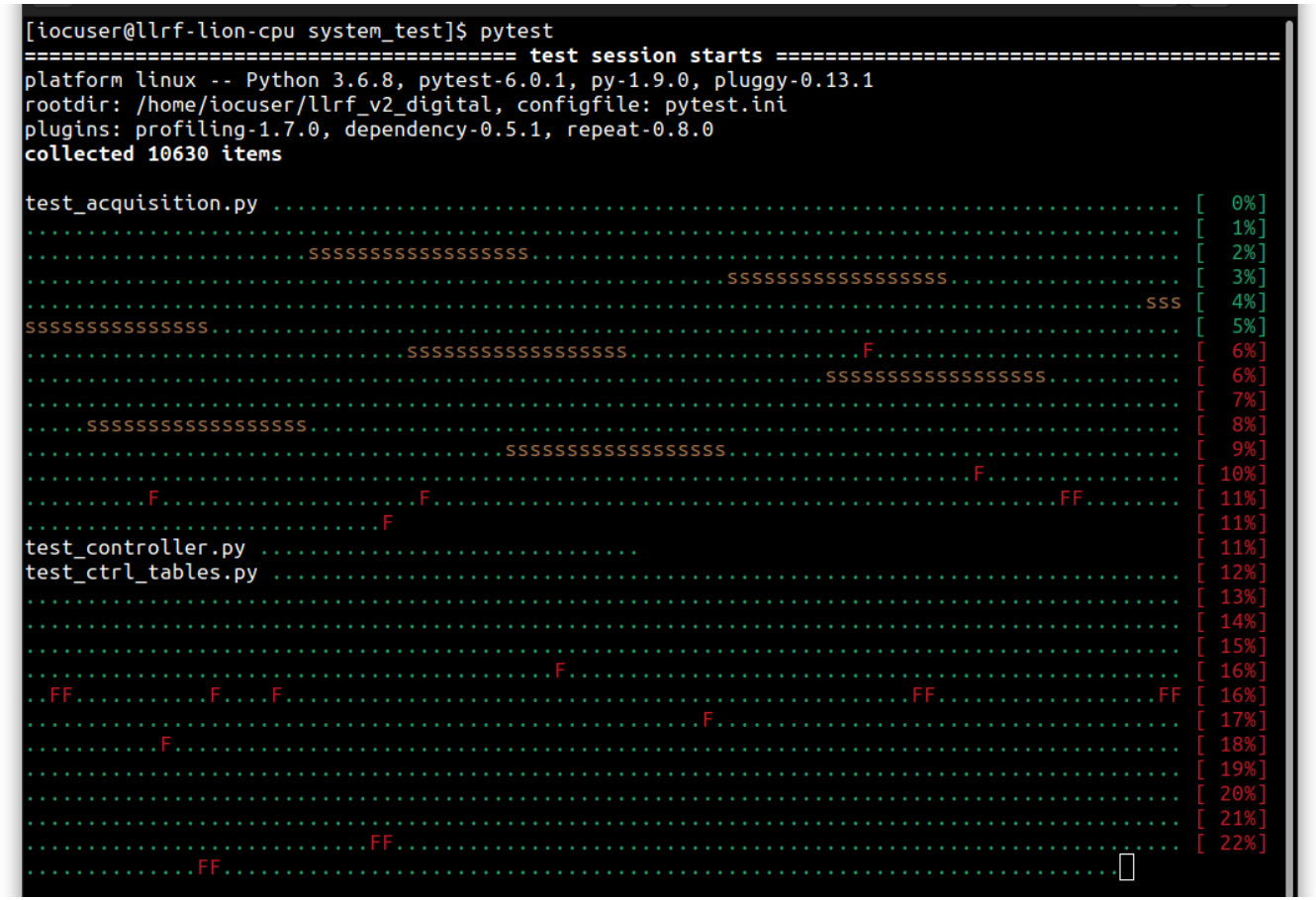


# Example – P component of PID controller

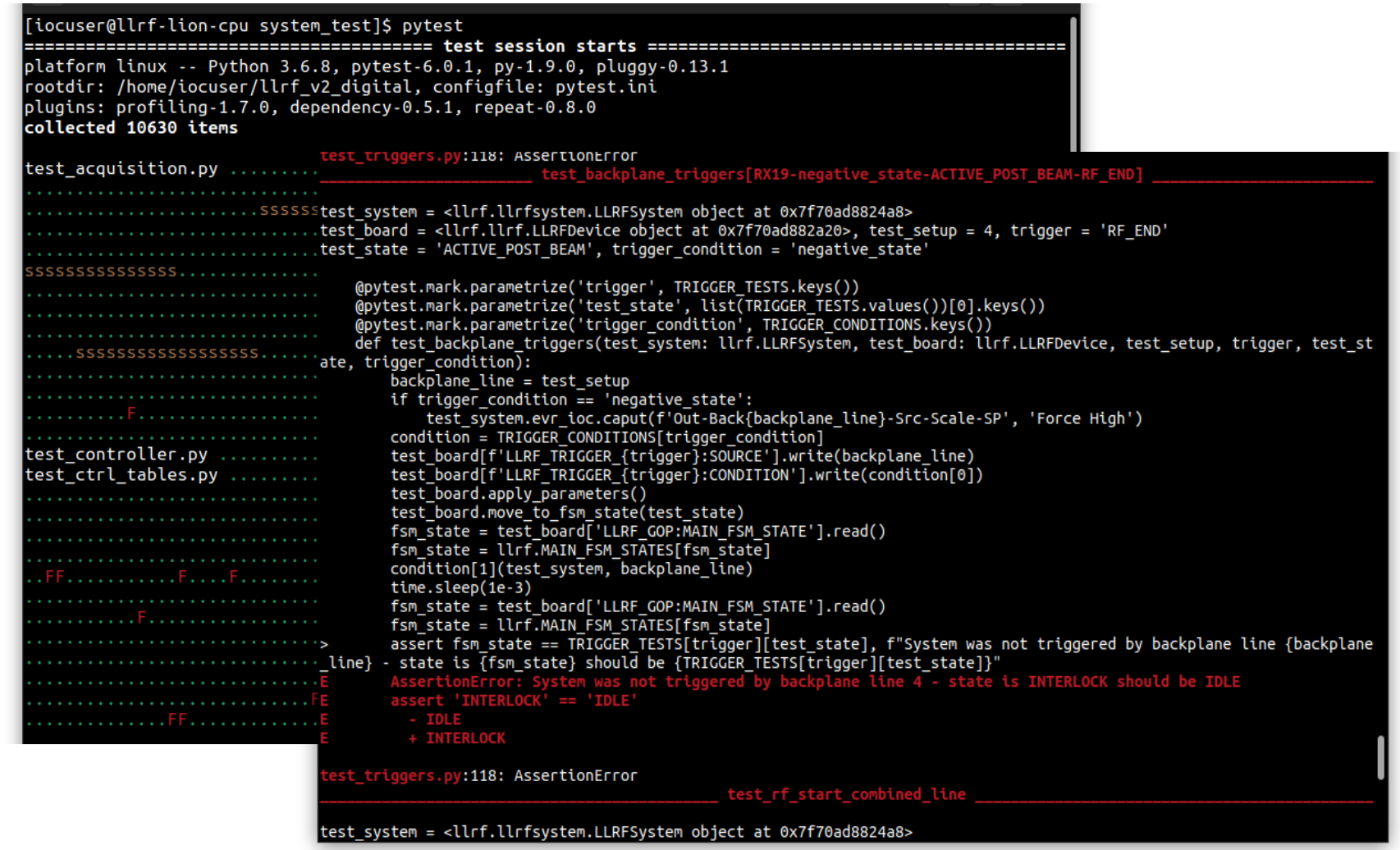
```
@pytest.mark.parametrize('ctrl_error', [-1.0, -0.5, 0.0, 0.01, 0.1, 0.5, 0.999])
@pytest.mark.parametrize('k_p', [0.0, 0.1, 0.2, 0.5, 0.9, 1.0, 2.0, 10.0, 100.0])

def test_k_p(self, idle_test_board, ctrl_error, k_p):

    # Local test setup
    dut = idle_test_board
    dut['LLRF_IQ_CTRL:IQ_CTRL_OUT_SOURCE'].write(0b111)
    dut['LLRF_SP_CONFIG:CONSTANT_EN'].write(1)
    dut['LLRF_PI_FIXED_SP:CONST_SP'].write(lowlevhw.float_to_fixed(ctrl_error, 1, 15))
    dut['LLRF_PI_K'].write(lowlevhw.float_to_fixed(k_p, 8, 2)


    # Execute test and acquire data
    dut.force_timing_pulse()
    time.sleep(0.01)
    measurement = dut['IQ_AFTER_CTRL'].read(test_length*4)


    # Calculate expected value
    expected_value = max(-2.0, min(1.99999, k_p * ctrl_error))
    expected = [(expected_value, 0.0)] * len(measurement)

    # Compare measurement with expected value
    assert_waveform_allclose(measurement, expected, shifttol=0, atol=TOLERANCE)
```



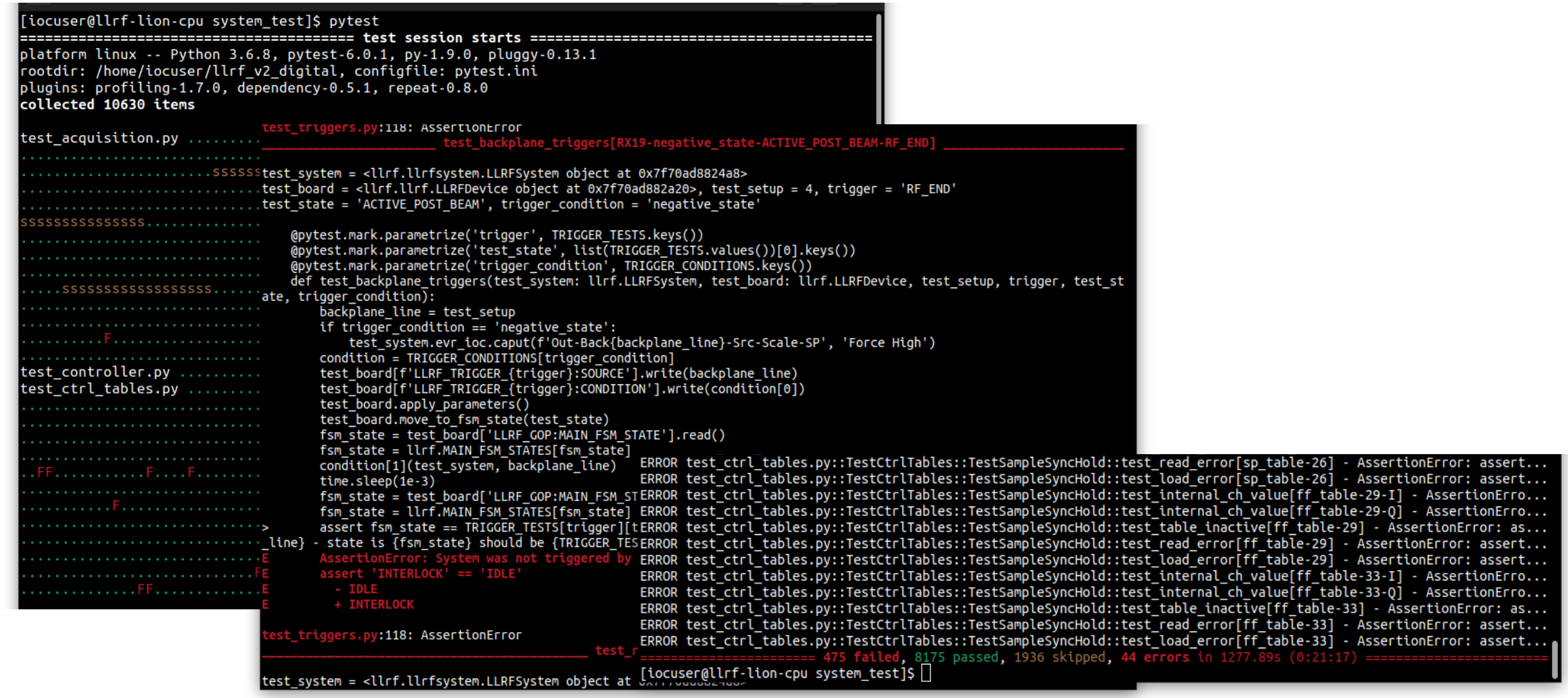
# Result


[iocuser@llrf-lion-cpu system_test]$ pytest
test session starts
platform linux -- Python 3.6.8, pytest-6.0.1, py-1.9.0, pluggy-0.13.1
rootdir: /home/iocuser/llrf_v2_digital, configfile: pytest.ini
plugins: profiling-1.7.0, dependency-0.5.1, repeat-0.8.0
collected 10630 items
test_acquisition.py
test_controller.py
test_ctrl_tables.py

# Result


[iocuser@llrf-lion-cpu system_test]$ pytest
test session starts
platform linux -- Python 3.6.8, pytest-6.0.1, py-1.9.0, pluggy-0.13.1
rootdir: /home/iocuser/llrf_v2_digital, configfile: pytest.ini
plugins: profiling-1.7.0, dependency-0.5.1, repeat-0.8.0
collected 10630 items
test_acquisition.py
test_controller.py
test_ctrl_tables.py
test_triggers.py:118: AssertionError
test_backplane_triggers[RX19-negative_state-ACTIVE_POST_BEAM-RF_END]
test_system = <llrf.llrfsystem.LLRFSystem object at 0x7f70ad8824a8>
test_board = <llrf.llrf.LLRFDevice object at 0x7f70ad882a20>, test_setup = 4, trigger = 'RF_END'
test_state = 'ACTIVE_POST_BEAM', trigger_condition = 'negative_state'
@pytest.mark.parametrize('trigger', TRIGGER_TESTS.keys())
@pytest.mark.parametrize('test_state', list(TRIGGER_TESTS.values())[0].keys())
@pytest.mark.parametrize('trigger_condition', TRIGGER_CONDITIONS.keys())
def test_backplane_triggers(test_system: llrf.LLRFSystem, test_board: llrf.LLRFDevice, test_setup, trigger, test_state, trigger_condition):
backplane_line = test_setup
if trigger_condition == 'negative_state':
test_system.evr_ioc.caput(f'Out-Back{backplane_line}-Src-Scale-SP', 'Force High')
condition = TRIGGER_CONDITIONS[trigger_condition]
test_board[f'LLRF_TRIGGER_{trigger}:SOURCE'].write(backplane_line)
test_board[f'LLRF_TRIGGER_{trigger}:CONDITION'].write(condition[0])
test_board.apply_parameters()
test_board.move_to_fsm_state(test_state)
fsm_state = test_board['LLRF_GOP:MAIN_FSM_STATE'].read()
fsm_state = llrf.MAIN_FSM_STATES[fsm_state]
condition[1](test_system, backplane_line)
time.sleep(1e-3)
fsm_state = test_board['LLRF_GOP:MAIN_FSM_STATE'].read()
fsm_state = llrf.MAIN_FSM_STATES[fsm_state]
> assert fsm_state == TRIGGER_TESTS[trigger][test_state], f"System was not triggered by backplane line {backplane_line} - state is {fsm_state} should be {TRIGGER_TESTS[trigger][test_state]}"
E AssertionError: System was not triggered by backplane line 4 - state is INTERLOCK should be IDLE
E assert 'INTERLOCK' == 'IDLE'
E - IDLE
E + INTERLOCK
test_triggers.py:118: AssertionError
test_rf_start_combined_line
test_system = <llrf.llrfsystem.LLRFSystem object at 0x7f70ad8824a8>

# Result


[iocuser@llrf-lion-cpu system_test]$ pytest
test session starts
platform linux -- Python 3.6.8, pytest-6.0.1, py-1.9.0, pluggy-0.13.1
rootdir: /home/iocuser/llrf_v2_digital, configfile: pytest.ini
plugins: profiling-1.7.0, dependency-0.5.1, repeat-0.8.0
collected 10630 items
test_acquisition.py
test_controller.py
test_ctrl_tables.py
test_triggers.py:118: AssertionError
test_backplane_triggers[RX19-negative_state-ACTIVE_POST_BEAM-RF_END]
test_system = <llrf.llrfsystem.LLRFSystem object at 0x7f70ad8824a8>
test_board = <llrf.llrf.LLRFDevice object at 0x7f70ad882a20>, test_setup = 4, trigger = 'RF_END'
test_state = 'ACTIVE_POST_BEAM', trigger_condition = 'negative_state'
@pytest.mark.parametrize('trigger', TRIGGER_TESTS.keys())
@pytest.mark.parametrize('test_state', list(TRIGGER_TESTS.values())[0].keys())
@pytest.mark.parametrize('trigger_condition', TRIGGER_CONDITIONS.keys())
def test_backplane_triggers(test_system: llrf.LLRFSystem, test_board: llrf.LLRFDevice, test_setup, trigger, test_state, trigger_condition):
backplane_line = test_setup
if trigger_condition == 'negative_state':
test_system.evr_ioc.caput(f'Out-Back{backplane_line}-Src-Scale-SP', 'Force High')
condition = TRIGGER_CONDITIONS[trigger_condition]
test_board[f'LLRF_TRIGGER_{trigger}:SOURCE'].write(backplane_line)
test_board[f'LLRF_TRIGGER_{trigger}:CONDITION'].write(condition[0])
test_board.apply_parameters()
test_board.move_to_fsm_state(test_state)
fsm_state = test_board['LLRF_GOP:MAIN_FSM_STATE'].read()
fsm_state = llrf.MAIN_FSM_STATES[fsm_state]
condition[1](test_system, backplane_line)
time.sleep(1e-3)
fsm_state = test_board['LLRF_GOP:MAIN_FSM_ST
fsm_state = llrf.MAIN_FSM_STATES[fsm_state]
> assert fsm_state == TRIGGER_TESTS[trigger][t
_line} - state is {fsm_state} should be {TRIGGER_TES
E AssertionError: System was not triggered by
E assert 'INTERLOCK' == 'IDLE'
E - IDLE
E + INTERLOCK
test_triggers.py:118: AssertionError
test_r
test_system = <llrf.llrfsystem.LLRFSystem object at
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_read_error[sp_table-26] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_load_error[sp_table-26] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_internal_ch_value[ff_table-29-I] - AssertionErro...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_internal_ch_value[ff_table-29-Q] - AssertionErro...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_table_inactive[ff_table-29] - AssertionError: as...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_read_error[ff_table-29] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_load_error[ff_table-29] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_internal_ch_value[ff_table-33-I] - AssertionErro...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_internal_ch_value[ff_table-33-Q] - AssertionErro...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_table_inactive[ff_table-33] - AssertionError: as...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_read_error[ff_table-33] - AssertionError: assert...
ERROR test_ctrl_tables.py::TestCtrlTables::TestSampleSyncHold::test_load_error[ff_table-33] - AssertionError: assert...
475 failed, 8175 passed, 1936 skipped, 44 errors in 1277.89s (0:21:17)
[iocuser@llrf-lion-cpu system_test]$

# Conclusion

- Some initial effort needed to implement Python libraries/models
- No bigger issues with LLRF during beam commissioning
- Used with other systems within the ESS FPGA Framework

For more information, contact me: christian.amstutz@ess.eu

# European Spallation Source

Goal: Availability