

```
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_store_EEPROM_data[S1.4-0042-False-EEPROM_KRIA-U:37] PASSED [ 1%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_kria_clk_chips[SYNC_KRIA-U:49] PASSED [ 2%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_kria_clk_chips[Osc_KRIA-U:52] PASSED [ 4%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_max34451[MAX34451-U:17] PASSED [ 5%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_max34451[MAX34451-U:17] PASSED [ 7%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4676A_Clocks-U:3] PASSED [ 8%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4680_MGT_NORTH-U:12] PASSED [ 9%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4680_MGT_SOUTH-U:15] PASSED [ 11%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4700_VCCINT_1-U:29] PASSED [ 12%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4700_VCCINT_0-U:30] PASSED [ 14%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4676A FireFly North-U:23] PASSED [ 15%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_validate_LTM4xx[LTM4676A FireFly South-U:24] PASSED [ 16%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xx[LTM4676A_Clocks-U:3] PASSED [ 18%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4680_MGT_NORTH-U:12] PASSED [ 19%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4680_MGT_SOUTH-U:15] PASSED [ 21%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4700_VCCINT_1-U:29] PASSED [ 22%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4700_VCCINT_0-U:30] PASSED [ 23%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4676A FireFly North-U:23] PASSED [ 25%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_configure_LTM4xxx[LTM4676A FireFly South-U:24] PASSED [ 26%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_turn_power_on[domain0] PASSED [ 28%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_turn_power_on[domain1] PASSED [ 29%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[3V3_FF_NORTH-VOUT00-3.3-3.135-3.465] PASSED [ 30%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[3V3_FF_SOUTH-VOUT01-3.3-3.135-3.465] PASSED [ 32%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[1V8_FF-VOUT02-1.8-1.71-1.89] PASSED [ 33%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[3V75_FF-VOUT03-3.75-3.6-3.9] PASSED [ 35%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[0V9_FPGA0_MGTAVCC_NORTH-VOUT04-0.9-0.886-0.927] PASSED [ 36%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[0V9_FPGA0_MGTAVCC_SOUTH-VOUT05-0.9-0.886-0.927] PASSED [ 38%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[1V2_FPGA0_MGTAVTT_NORTH-VOUT06-1.2-1.182-1.236] PASSED [ 39%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[1V2_FPGA0_MGTAVTT_SOUTH-VOUT07-1.2-1.182-1.236] PASSED [ 40%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[0V85_FPGA0_VCCINT-VOUT08-0.85-0.837-0.862] PASSED [ 42%]
serenity-toolbox/python/pkg/serenity/tests/test_commissioning.py::test_measure_power_rails[1V8_FPGA0_VCCO-VOUT09-1.8-1.71-1.89] PASSED [ 43%]
```



# Factory Acceptance Test for Serenity-S1, a CMS Phase-2 Back-End Card

Hendrik Krause on behalf of the Serenity consortium

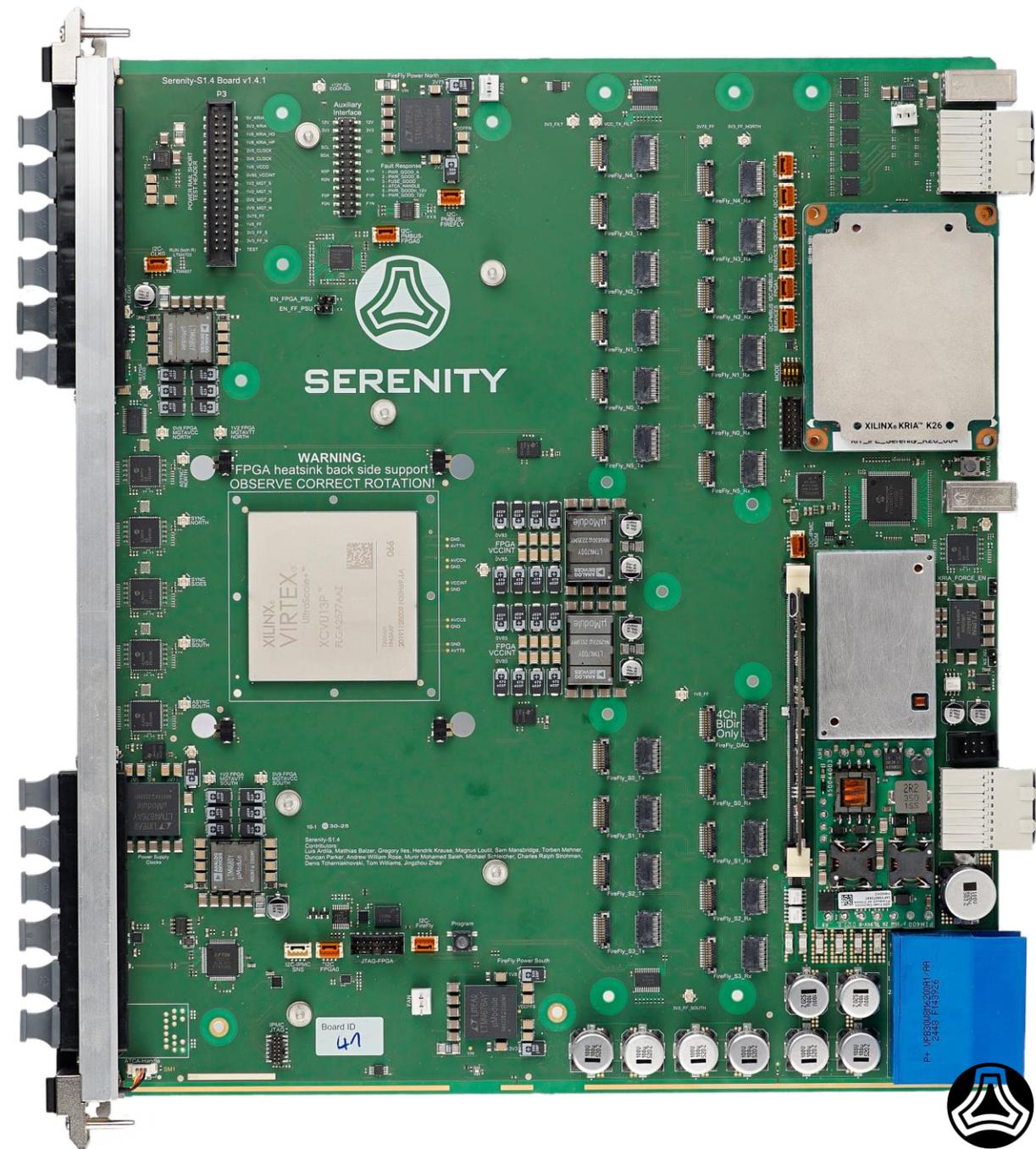
IPE-Seminar, 2026-03-12

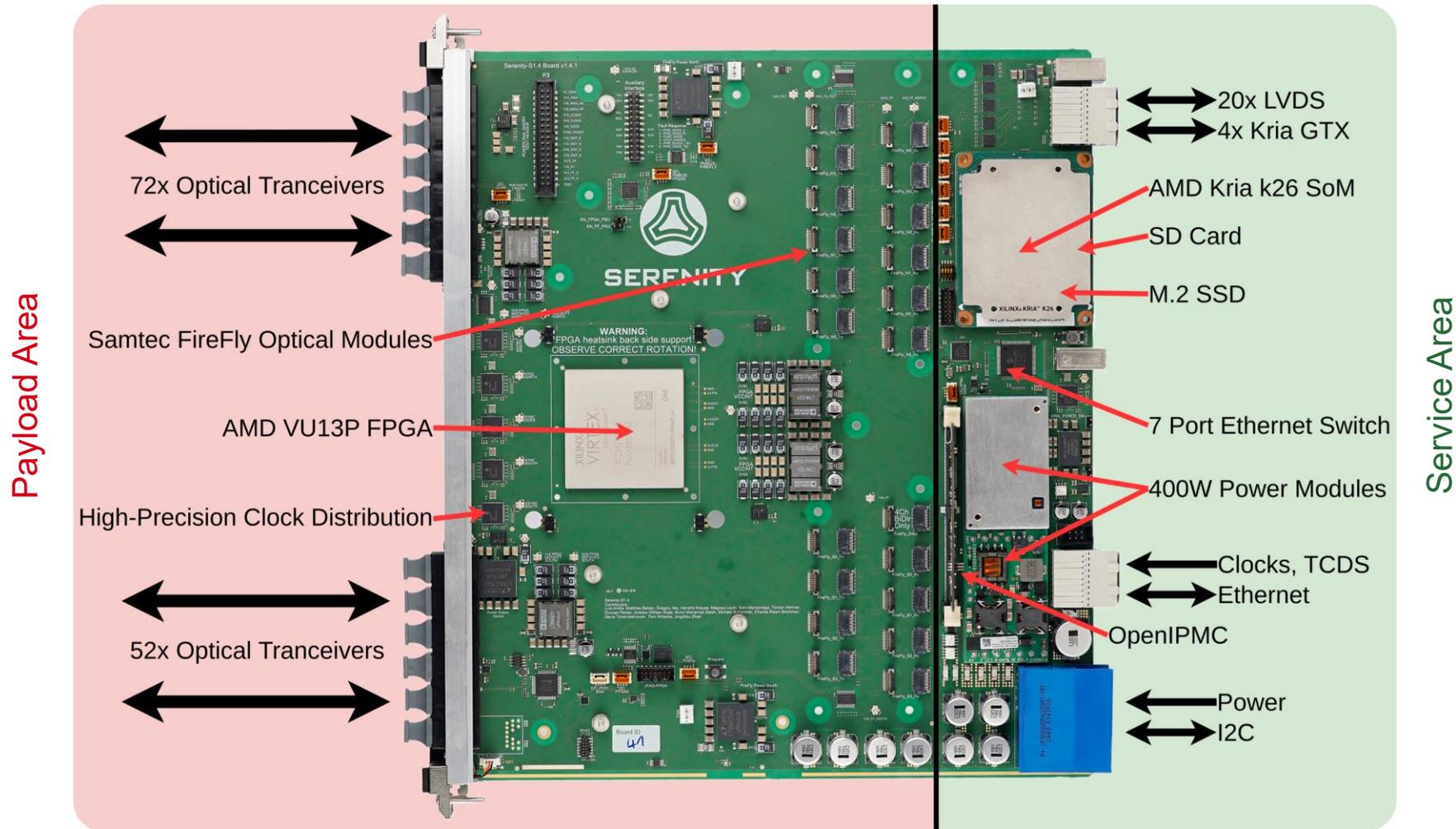


# What Is Serenity-S1 and Why Do We Need It?

- HL-LHC upgrade requires replacing and updating several sub-detector systems within CMS
- Upgrade covers both frontend and backend electronics
- Serenity-S1: common ATCA-based back-end board
  - Generic FPGA processing card
  - Meets requirements of multiple sub-detectors
- Adopted by 7 sub-detector systems
  - More than 700 boards are being produced and require testing

**TWEPP** [Talk] T. Mehner: [Lessons learned while developing the 2023 Serenity-S1 ATCA card](#)





# Architecture

**Tech** [Paper] T. Mehner et al.: [Serenity-S1 - A Versatile ATCA](#)  
**Rxiv** [Processing Card for the CMS Phase-2 Upgrade](#)

The Serenity-S1 is a versatile data processing card with...

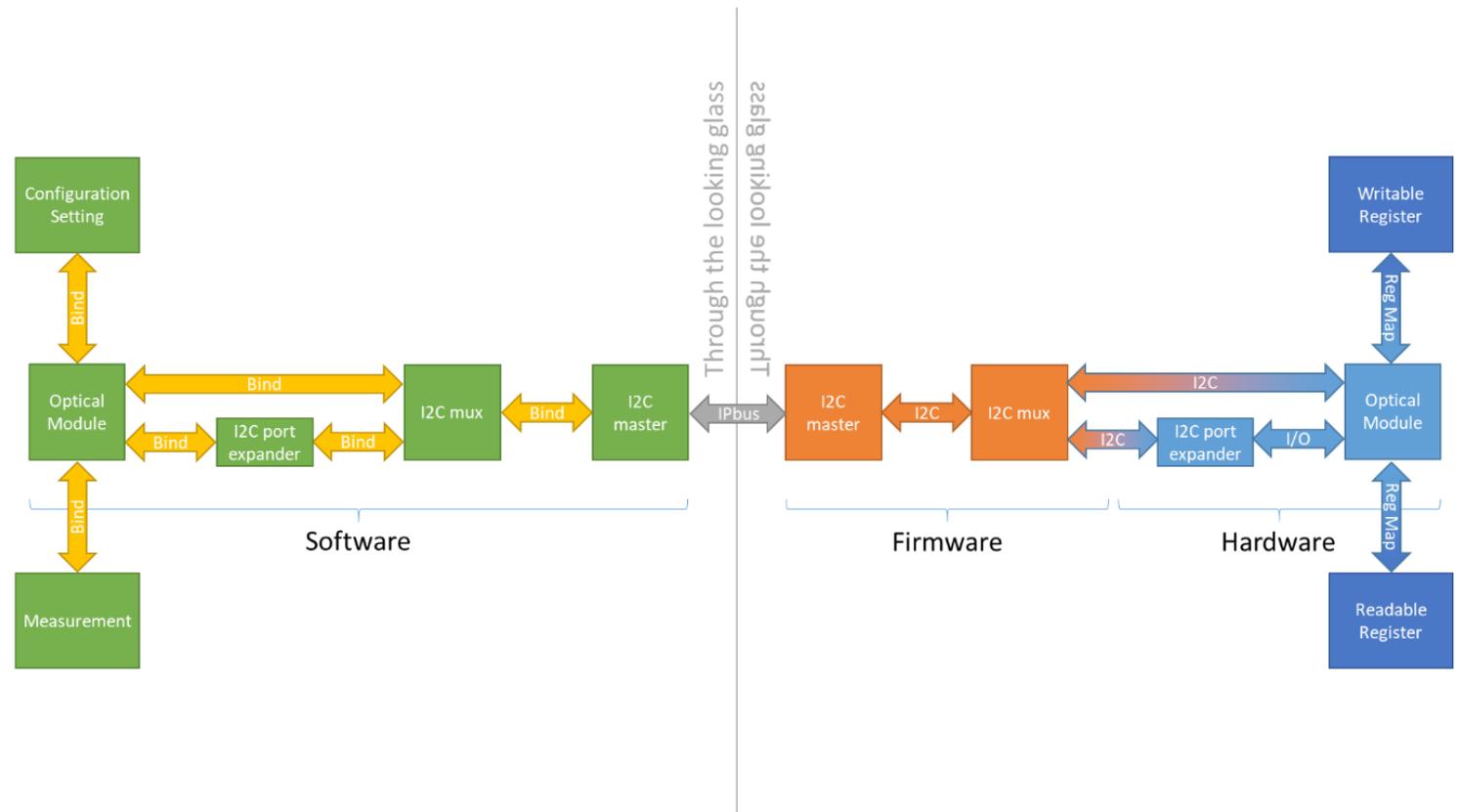
- ...3.1 Tbps digital bandwidth using FireFly optical modules
- ...a large VU13P FPGA for data processing



# SMASH

## Serenity Management SHell ([SMASH](#))

- Software providing hardware abstraction for slow control
  - Comparable to Linux device tree
  - [IPBus](#) based
- Different usage options
  - Interactive shell ([see below](#))
  - SMASH scripts
  - (Limited) Python interface



```
[root@mgmt-kki ~]# smash.exe -i
##### Welcome to the interactive SMASH shell. #####
##### Write SMASH commands and hit <enter> to run commands #####
##### Hit the 'esc' key to quit #####
>> Power:Sequencer Measure VOUT00
Power:Sequencer Measure VOUT00
VOUT00 : +968.000mV
>>
```



# EMP Framework

Extensible, Modular data Processor ([EMP](#))

- Framework providing hardware abstraction for the FPGA
  - Algorithm can use provided buffers for high speed access
  - Control through IPBus registers (over AXI Chip2Chip)
- EMPbutler software can access the EMP framework
  - Interactive CLI or Python API

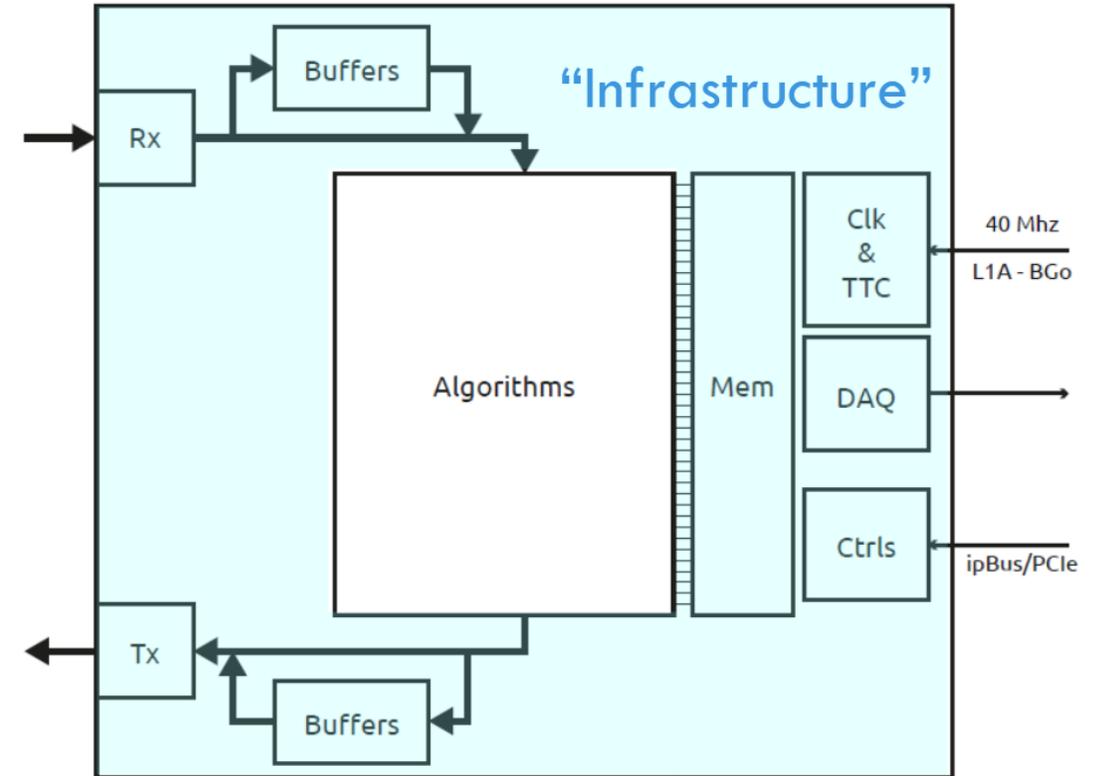
Example to reset FPGA and play and receive data:

```
$ empbutler -c CONNECTIONS.xml do DEVICE_ID reset internal
```

```
$ empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers tx PlayOnce -c CHANNEL_LIST --inject generate://pattern
```

```
$ empbutler -c CONNECTIONS_FILE.xml do BOARD_ID buffers tx Capture -c TX_CHANNEL_LIST
```

```
$ empbutler -c CONNECTIONS_FILE.xml do BOARD_ID capture --rx RX_CHANNEL_LIST --tx TX
```



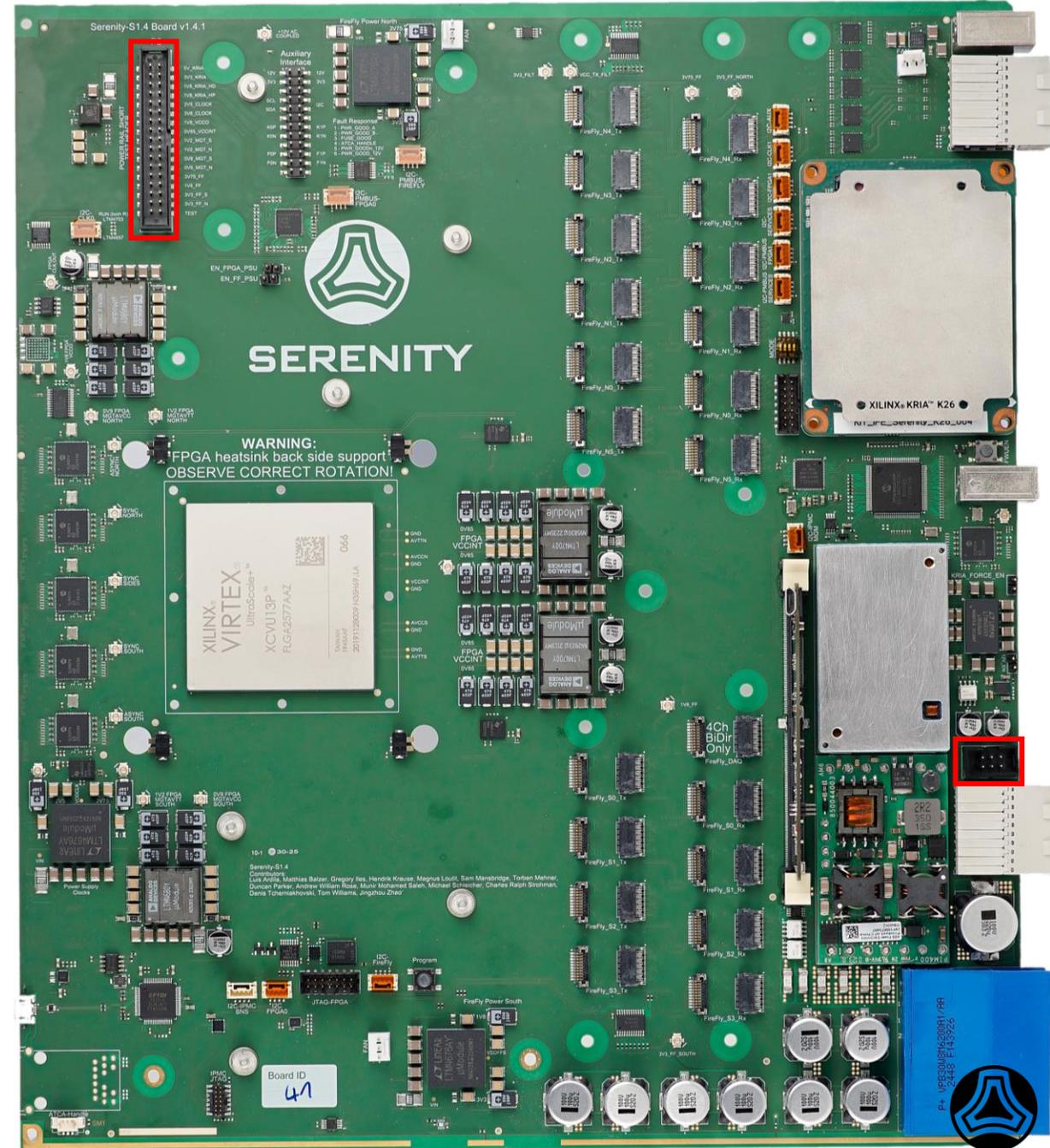
**JINST**

[Paper] T. Williams et al.: [EMP: a common infrastructure firmware framework for the CMS phase-2 upgrades](#)



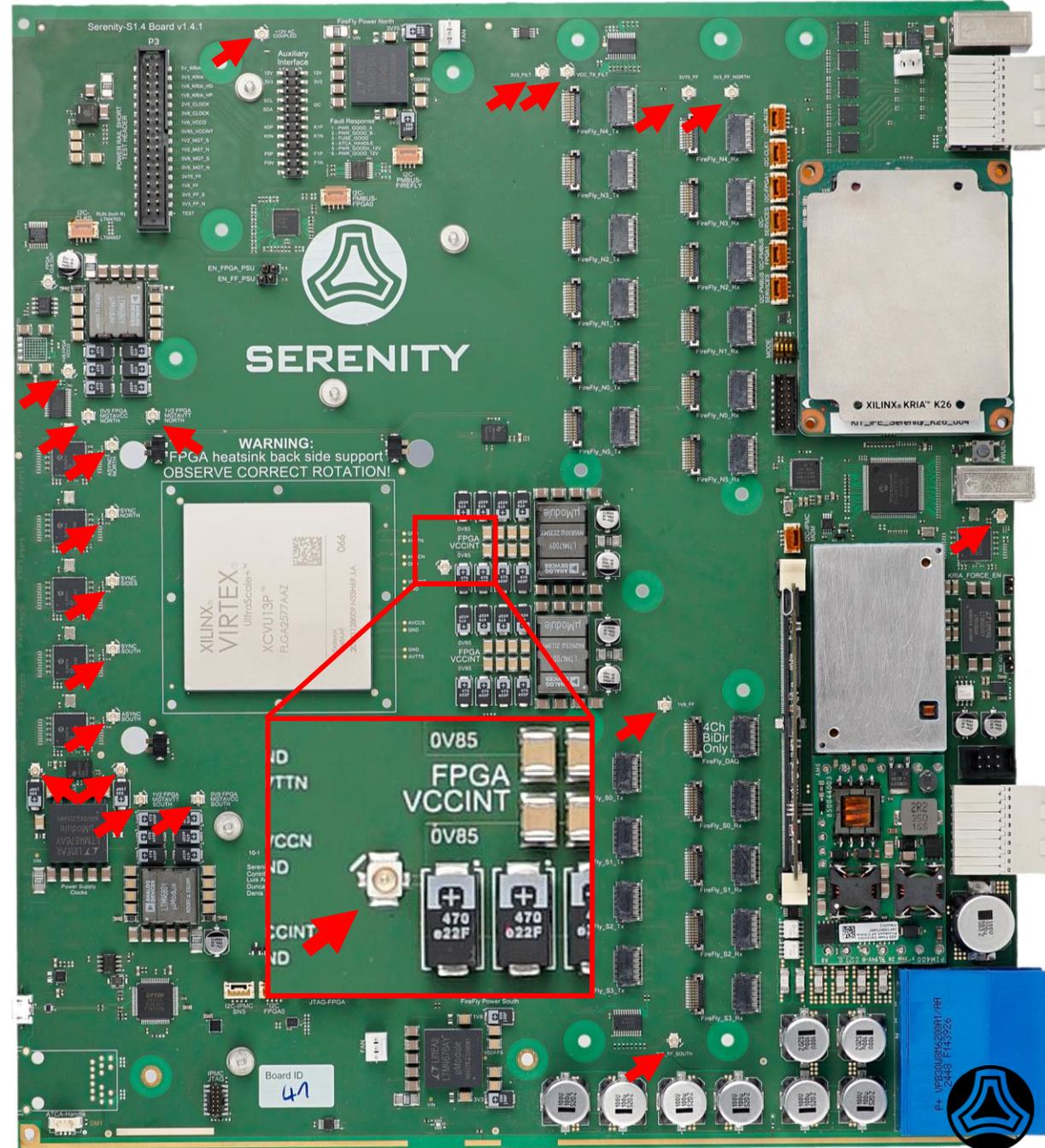
# Design for Debug

- Dedicated test header for short circuit detection
  - Access to the main power rails



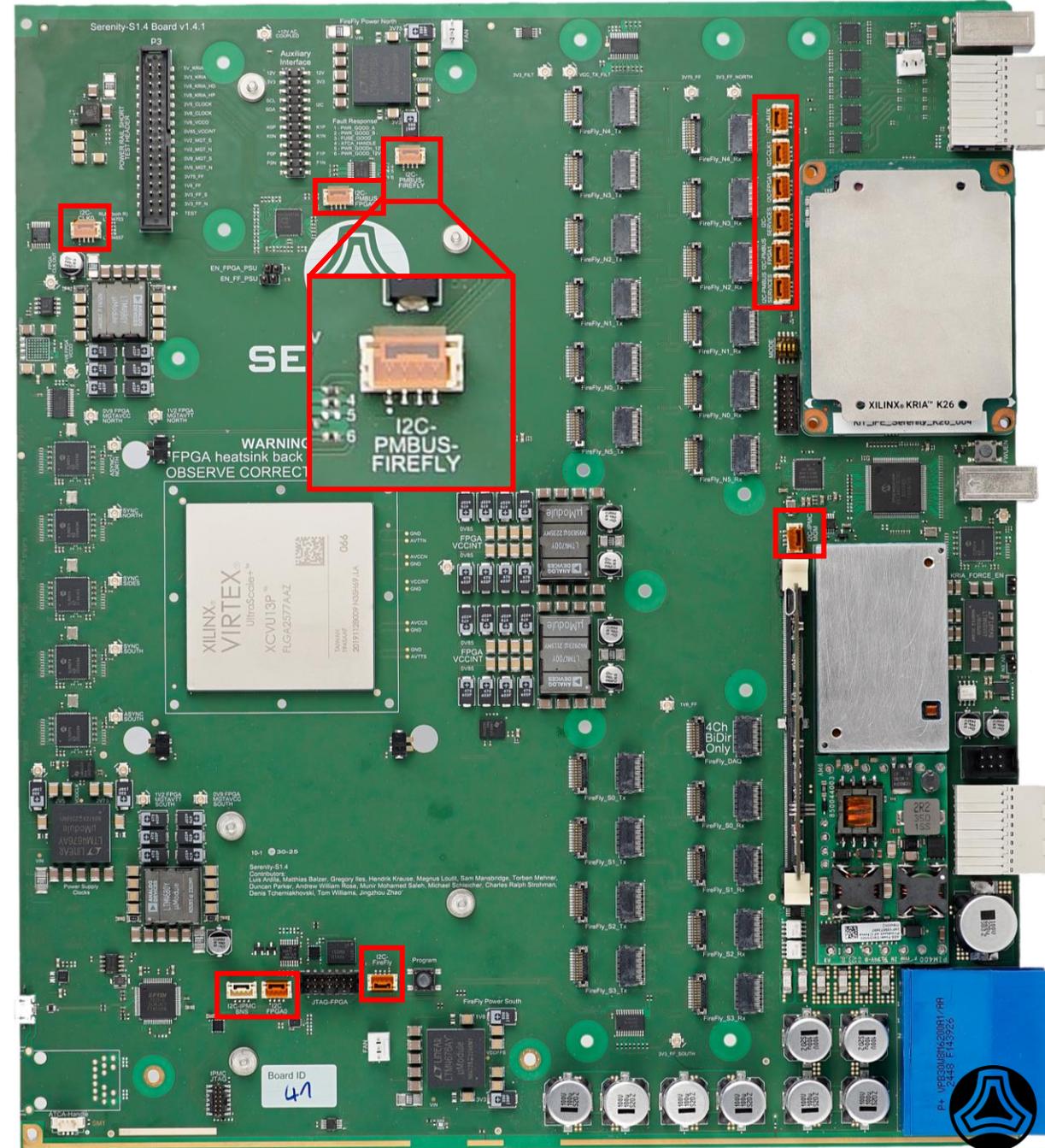
# Design for Debug

- Dedicated test header for short circuit detection
  - Access to the main power rails
- Test points for the most important nets (power and clocks)
  - uFI connectors allow the easy connection to scopes



# Design for Debug

- Dedicated test header for short circuit detection
  - Access to the main power rails
- Test points for the most important nets (power and clocks)
  - uFI connectors allow the easy connection to scopes
- Split the slow control (I2C, PMBus) into individual nets
  - Use dedicated test header for each net

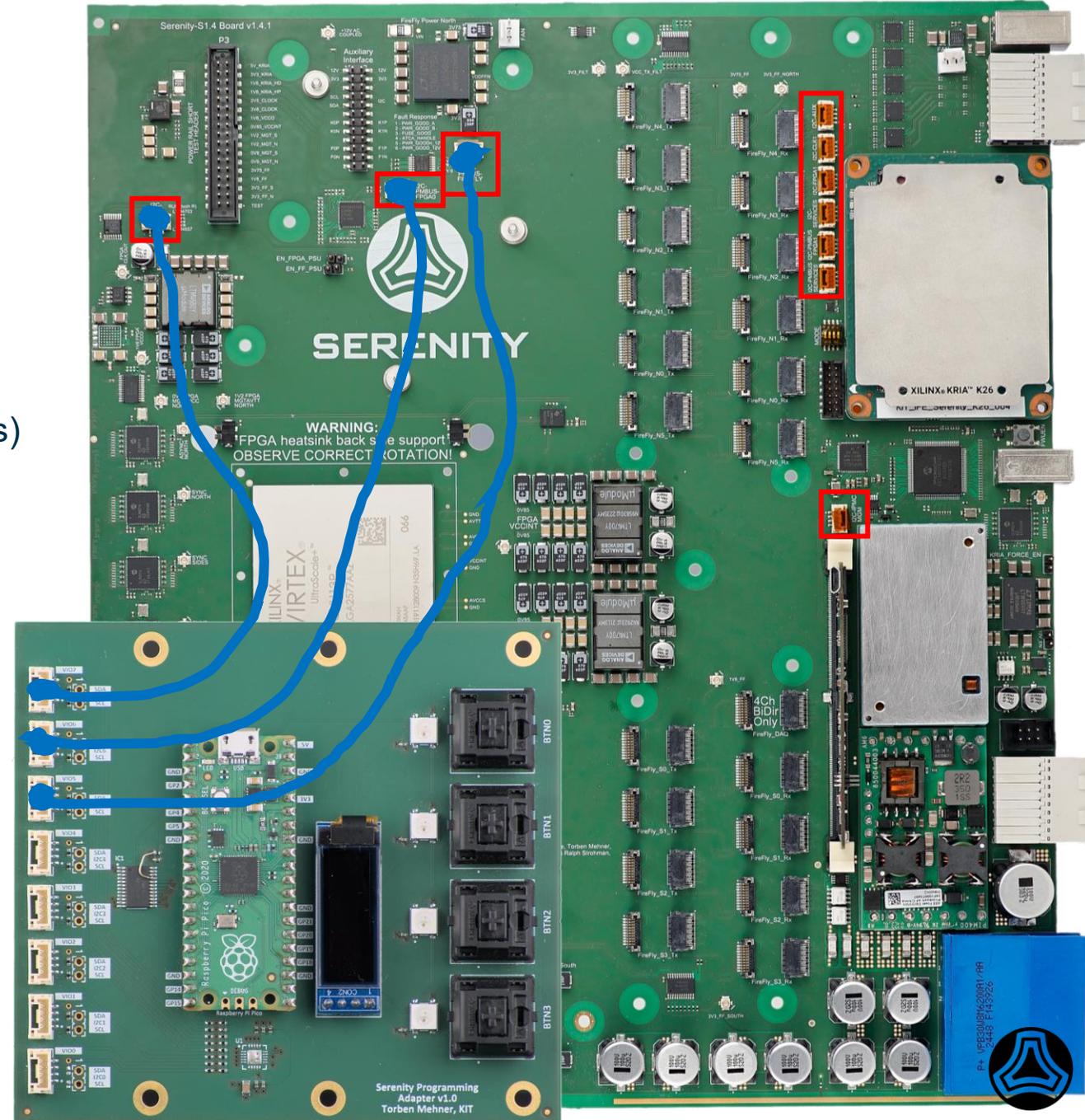


# Design for Debug

- Dedicated test header for short circuit detection
  - Access to the main power rails
- Test points for the most important nets (power and clocks)
  - uFI connectors allow the easy connection to scopes
- Split the slow control (I2C, PMBus) into individual nets
  - Use dedicated test header for each net
- Developed the “Serenity Programming Adapter”
  - Raspberry Pi Pico + MicroPython
  - Easy external debugging of I2C busses and components



MST Group [Talk] T. Mehner: [Improving Testing and Commissioning](#)



# Multi-Stage QA Strategy

## PCB Manufacturer

### Manufacturer QC

- Micro section analysis
- Surface finish
- Impedance measurement
- Solderability
- Ionic contamination
- ...

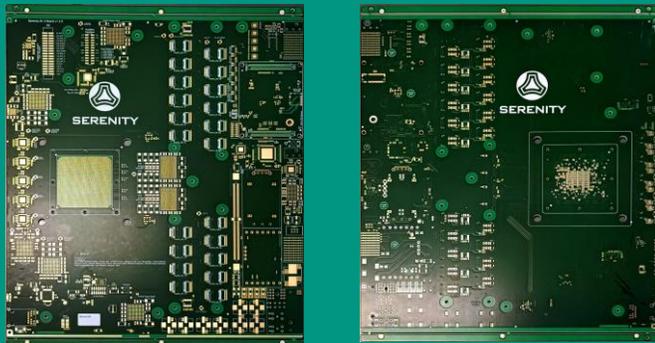


# Multi-Stage QA Strategy

## PCB Manufacturer

### Manufacturer QC

- Micro section analysis
- Surface finish
- Impedance measurement
- Solderability
- Ionic contamination
- ...



## PCB Assembler

### Assembler QC

- Visual Inspection
- Automated Optical Inspection (AOI)
- X-Ray

### Functional Testing

- Factory Acceptance Test (FAT)

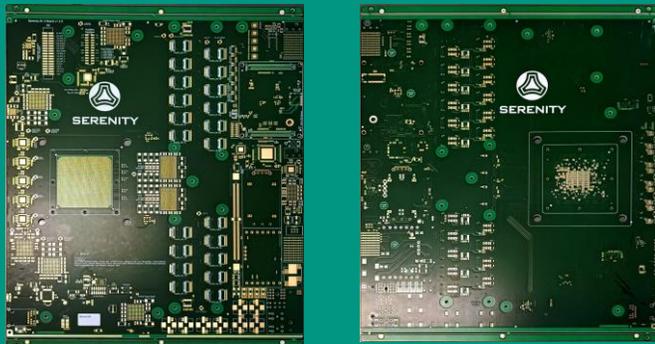


# Multi-Stage QA Strategy

## PCB Manufacturer

### Manufacturer QC

- Micro section analysis
- Surface finish
- Impedance measurement
- Solderability
- Ionic contamination
- ...



## PCB Assembler

### Assembler QC

- Visual Inspection
- Automated Optical Inspection (AOI)
- X-Ray

### Functional Testing

- Factory Acceptance Test (FAT)



## CERN

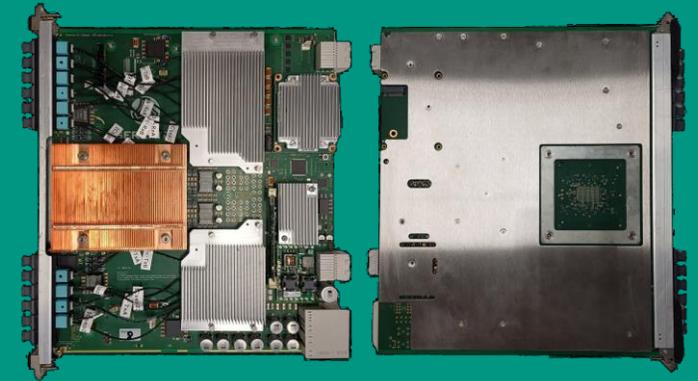
### User Acceptance Test

- Repetition of the FAT with more focus on SerDes links

Add heatsink, optics, back cover, OpenIPMC, Kria, ...

### Sub-detector specific testing

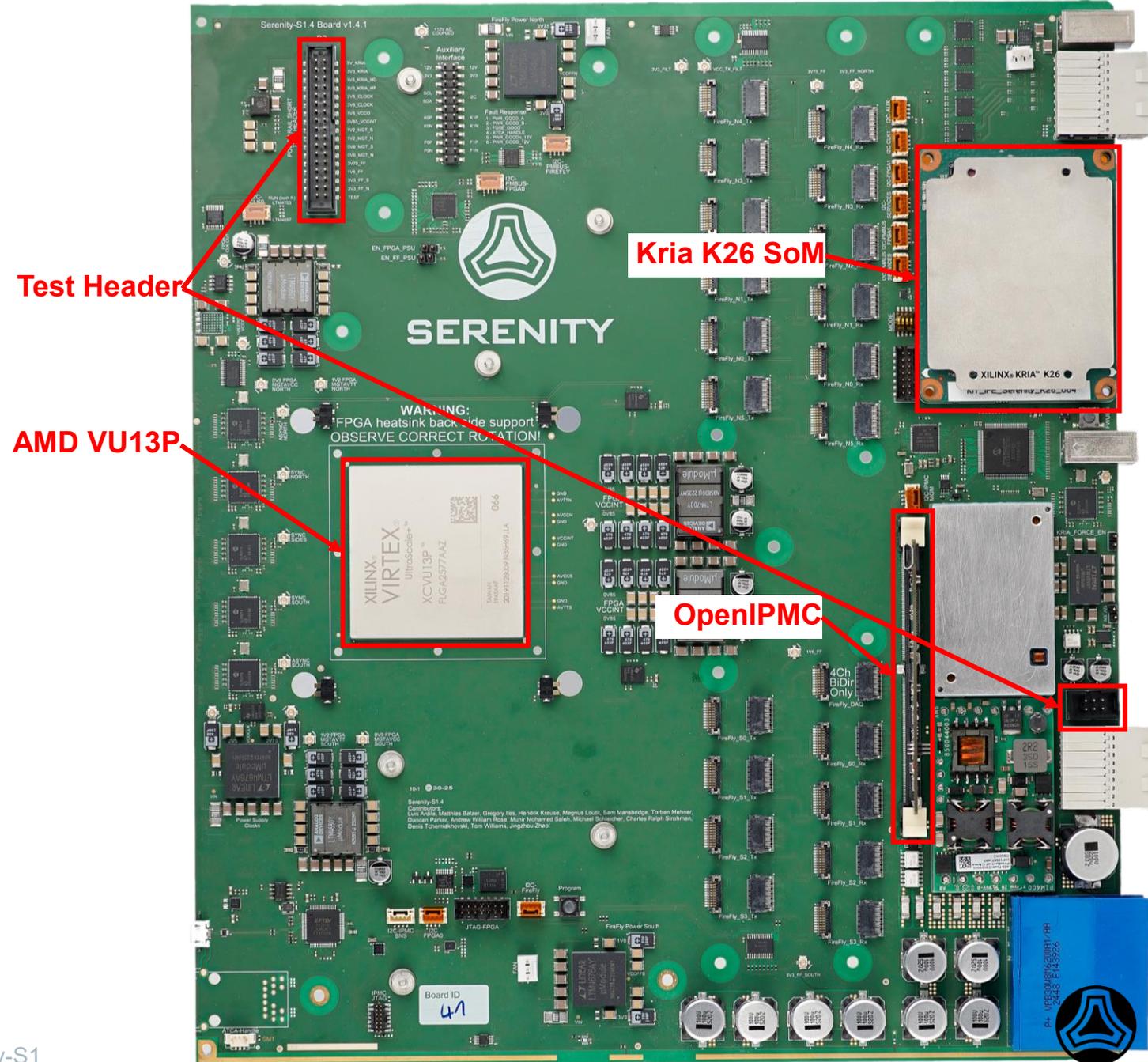
- Burn-in test



# FAT Scope

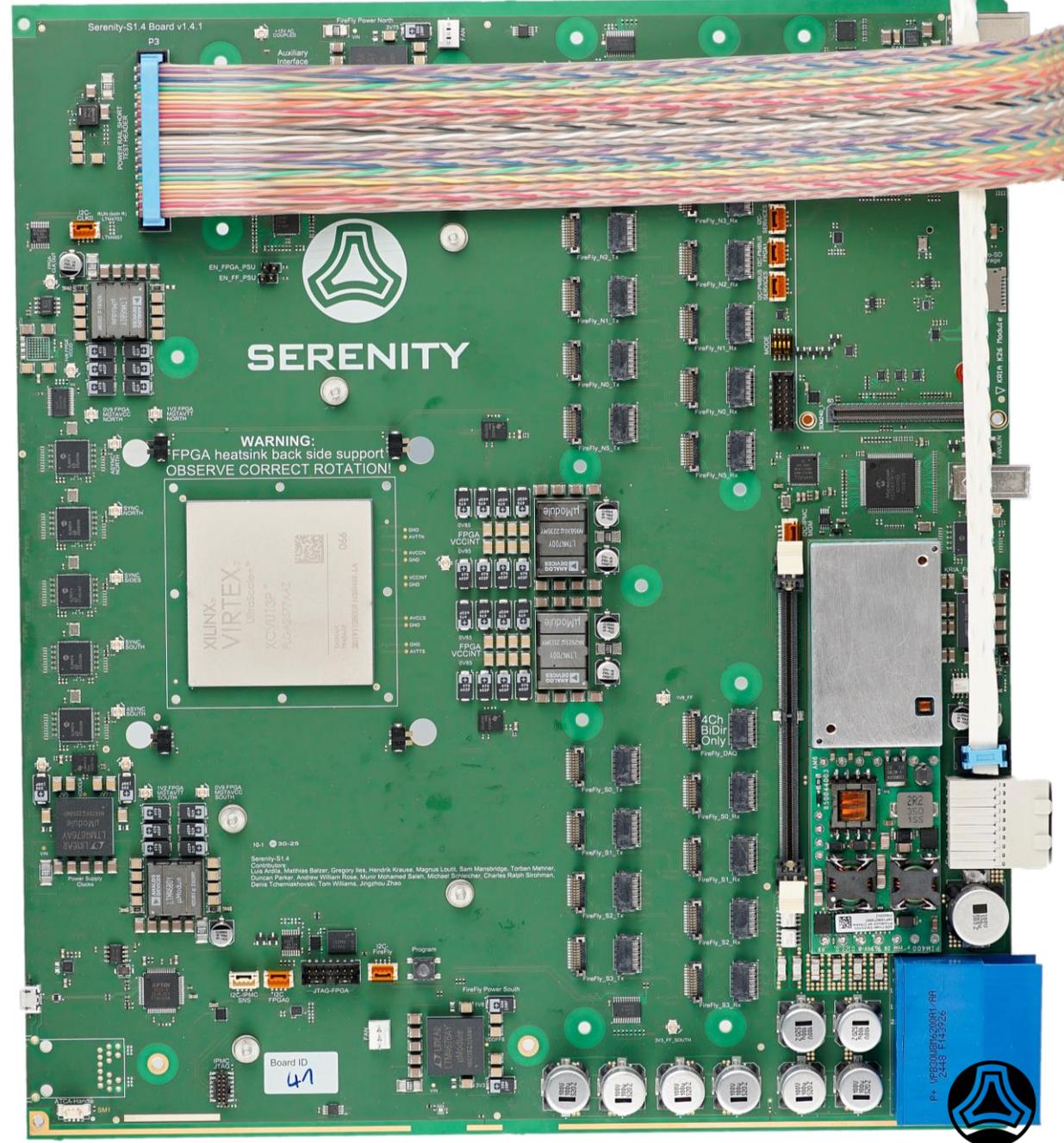
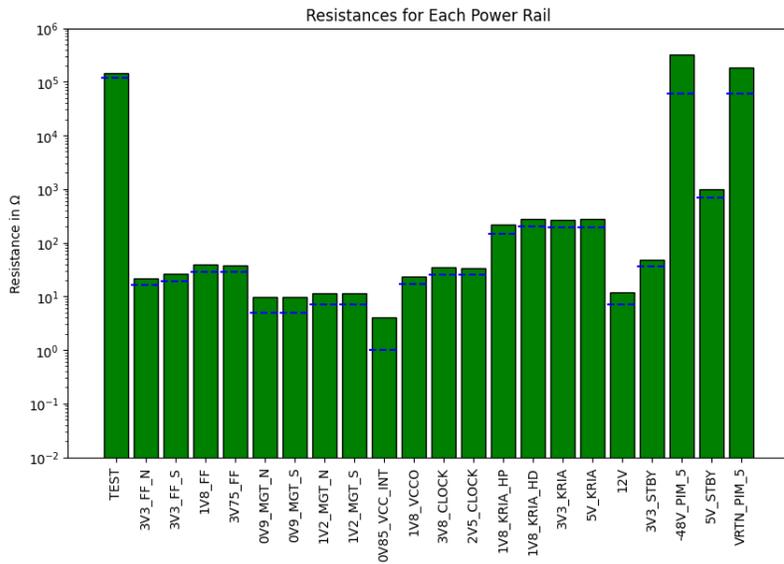
## Goals

- Functional testing directly at the manufacturer
  - Short repair cycles
- End-of-line test, automated <10 min
- Run by non-experts
- Onboard self-test (OpenIPMC, Kria SoM, FPGA)
- Minimal external equipment
- Reusable for maintenance/diagnostic
- Focus on most important functionality
  - Power, slow control, SerDes links



# FAT Procedure – Step 1: Short-Circuit Test

- Use off-the-shelf test equipment
- Measure the resistance of every power rail
- Comparison against threshold
- Controlled via test PC



# FAT Procedure – Step 2: Service Area Tests

## OpenIPMC

- Open-source ATCA IPMC implementation
- Monitors temperature and power
- Reports to the Shelf Manager

## Test Approach

- Expose sensor readings of the OpenIPMC to a telnet interface
- Python access via [telnetlib](#) from a test PC
- Offline check that the reading was successful and the values are plausible



```
[lab@oo-k03 ~]$ telnet 192.168.0.40
Trying 192.168.0.40...
Connected to 192.168.0.40.
Escape character is '^]'.

>> sensors
TEMP_FPGA_INT_[C]: 27.000000
TEMP_NORTH_[C]: 24.000000
TEMP_SOUTH_[C]: 24.000000
TEMP_SERVICE_[C]: 29.000000
TEMP_FPGA_EXT_[C]: 29.000000
PIM400_current_[A]: 1.316000
PIM400_voltage_A_[V]: 48.099998
PIM400_voltage_B_[V]: 48.099998
PIM400_temp_[C]: 38.244999
Barracuda_temp_[C]: 0.000000
Barracuda_voltage_in_[V]: Reading sensor failed!
Barracuda_voltage_out_[V]: 0.000000
Barracuda_current_out_[A]: Reading sensor failed!
PG_5V_KRIA: 1
PG_3V3_KRIA: 1
PG_1V8_KRIA_HD: 1
PG_1V8_KRIA_HP: 1
PG_KRIA_FPD: 1
PG_KRIA_LPD: 1
PG_KRIA_PL: 1
KRIA_ERROR_OUT: 0
PS_ERROR_STATUS: 0

>> □
```

Service Area

# FAT Procedure – Step 3: Payload Area Tests

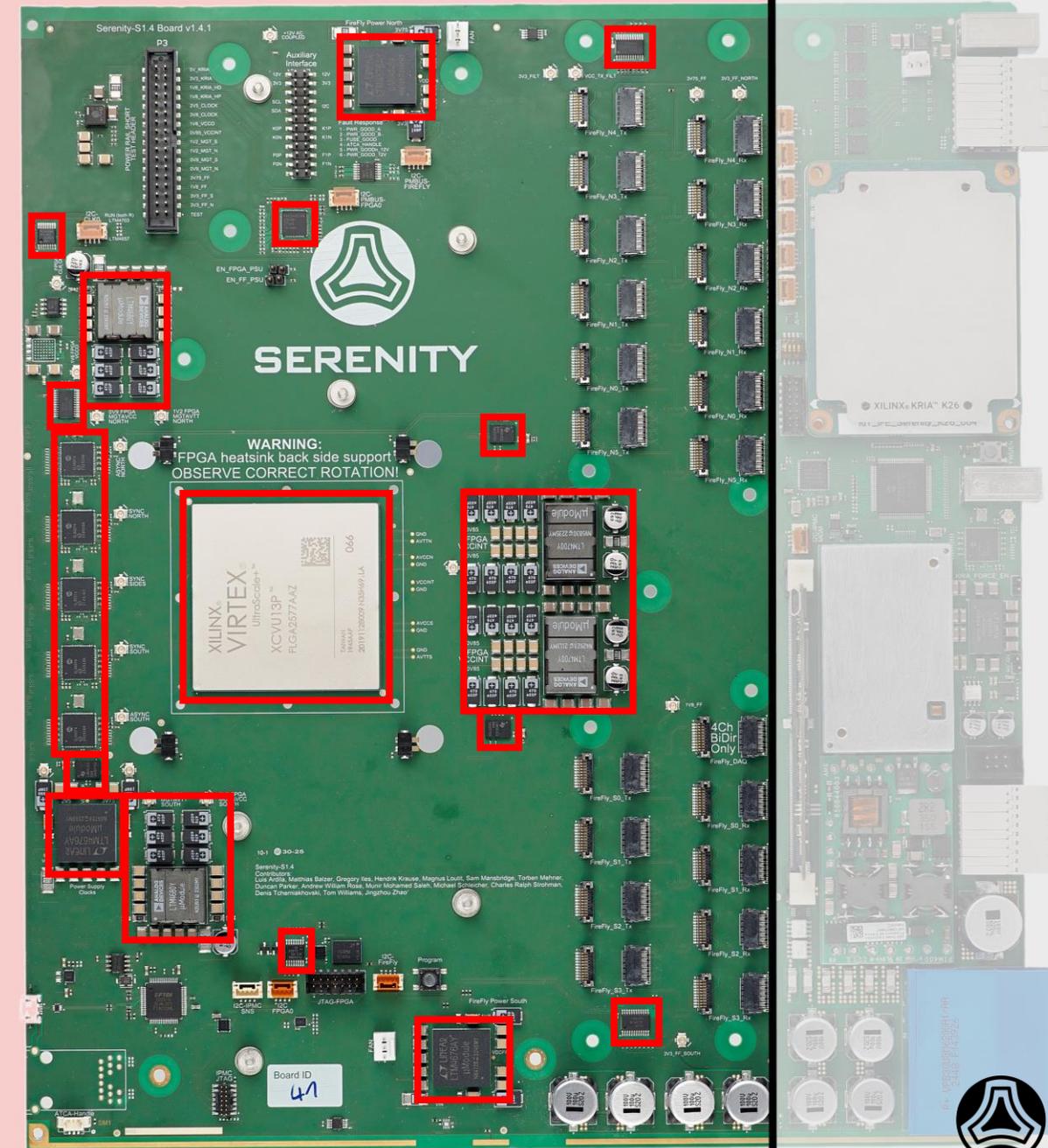
## Challenge

- Verify that all components in the slow control tree work
- Must configure power supplies & sequencer
- Must configure clock chips
- Store board ID in EEPROM

## Advanced board management: [SMASH](#)

- Framework on the Kria SoM
- Provide access to all I2C and PMBus devices + JTAG
- Build in mechanisms for measure, configure and verify
- Python API → use software test frameworks for hardware

Payload Area



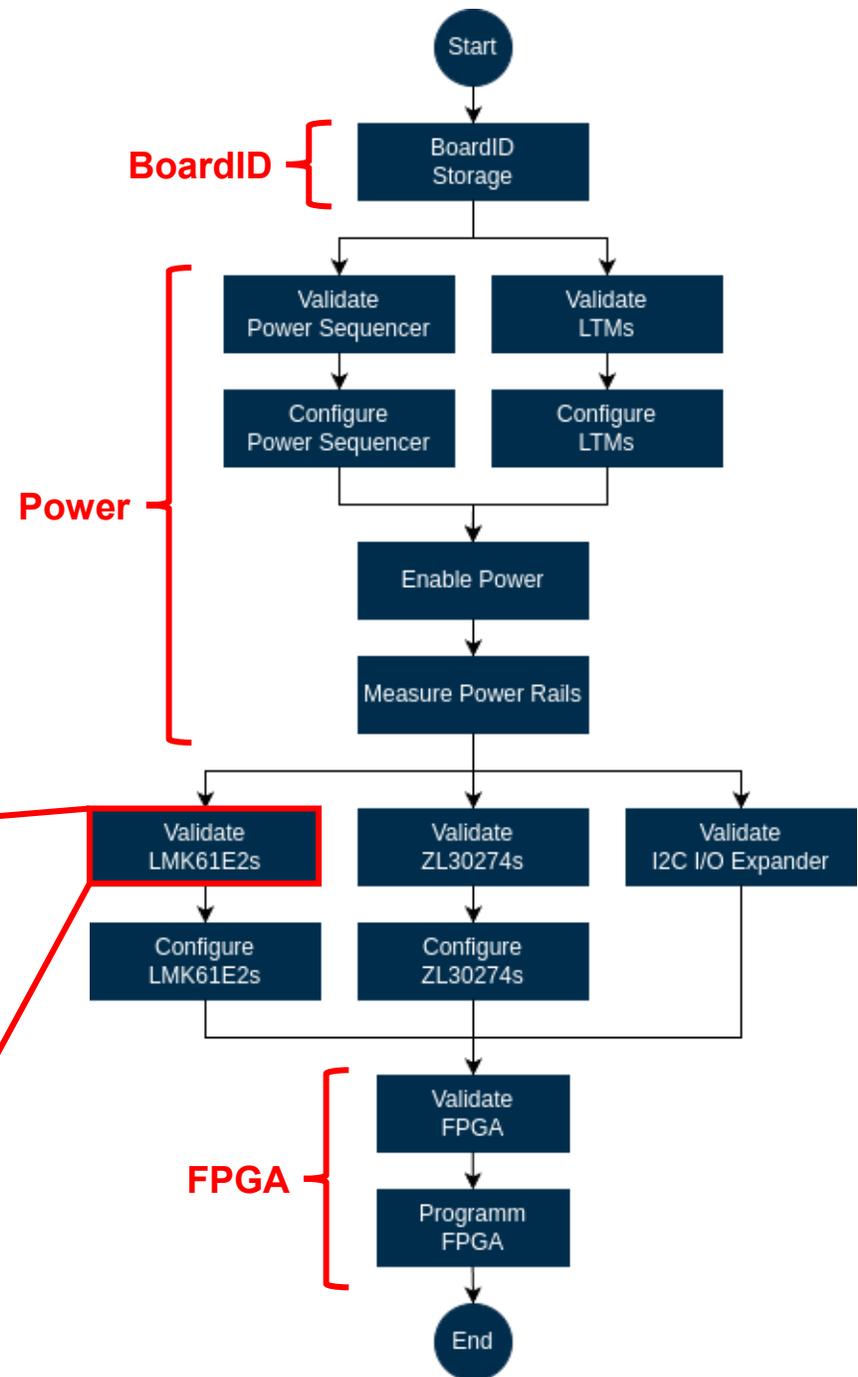
# FAT Procedure – Step 3: Structured Testing With Pytest

- SMASH Python API
  - Wrapped into [Pytest](#) functions
- Pytest features
  - Parameterization
  - [pytest-json](#) → results for reporting
  - [pytest-dependency](#) → enforce safe sequence
- Guarantees: if PSU fails → stop powering the payload area

```

@pytest.mark.dependency(depends=Dependencies
    instances("test_measure_power_rails", param_nominal_rail_voltage))
@pytest.mark.parametrize("name, designator", param_LMK61E2) Parameter
def test_validate_lmk61e2(smash_, name, designator):
    """
    Validate LMK61E2.

    Use the SMASH validate function to test if the
    LMK61E2 respond with the correct device ID.
    """
    answer = smash_.getElement(designator).validate() SMASH API call
    assert answer == True, "Validate failed for %s (%s)" % (name, designator) Evaluation
    
```



# FAT Procedure – Step 4: FPGA Tests

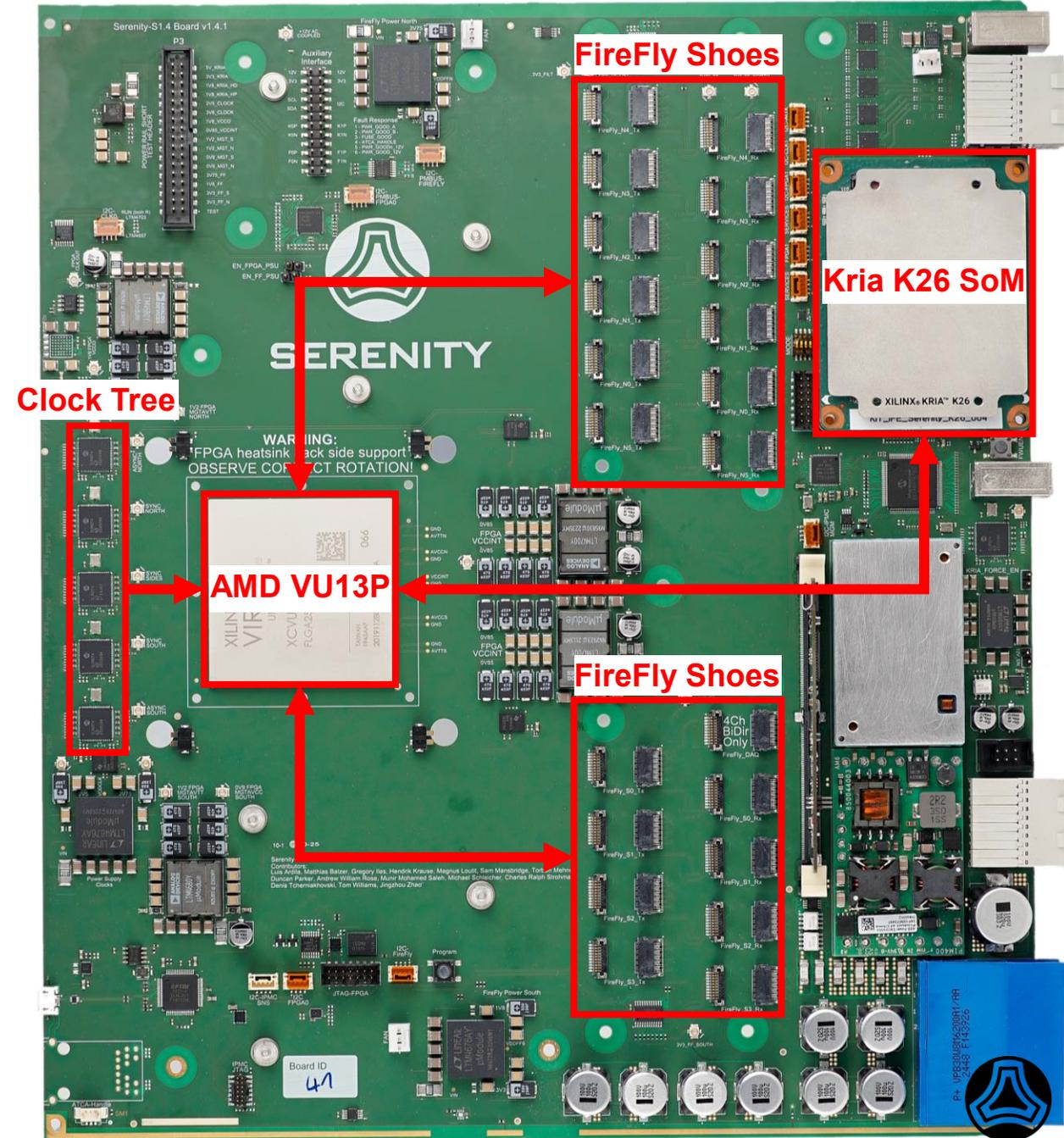
- Use EMP framework + EMP toolbox
  - Bitfile with built-in test functions
- Controlled via EMP Python bindings
  - Integrated with Pytest

Test sequence after programming FPGA

- Verify Kria FPGA Chip2Chip connection
- Measure MGT reclocks (async & sync)
- SerDes link test

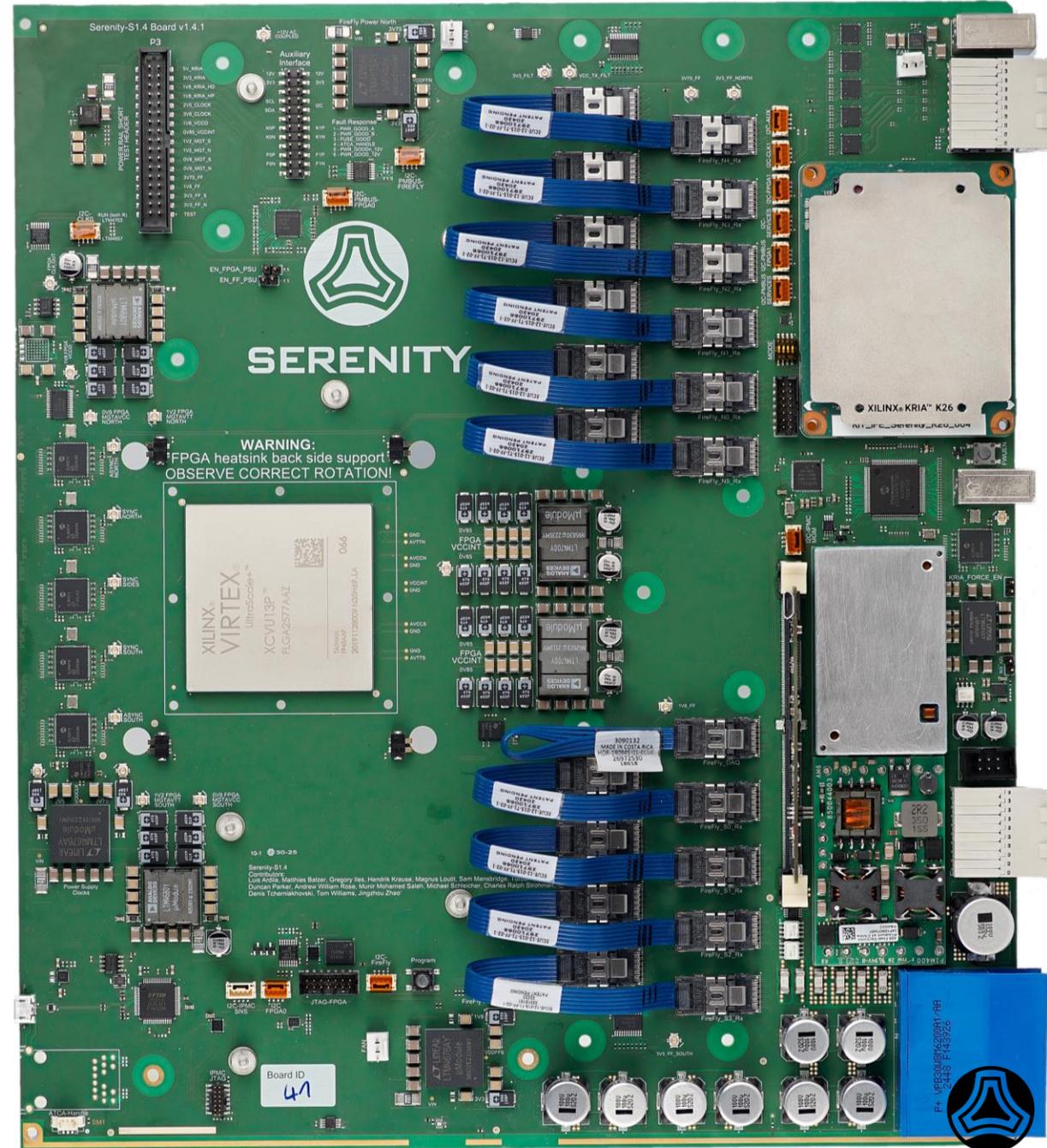
JINST

[Paper] T. Williams et al.: [EMP: a common infrastructure firmware framework for the CMS phase-2 upgrades](#)



# FAT Procedure – Step 5: SerDes Tests

- Based on existing EMP link test functionality (PRBS31)
- Copper loopback cables instead of optical FireFly modules
  - Faster setup & avoids risk of damaging optics
- Operated without heatsink → thermal precautions
  - Enable only 16 channels at a time
  - Run test for 5 seconds per link



# FAT Procedure – Step 6: Report Creation & Database Upload

- Raw outputs: logfiles + pytest reports
- Automated report generation
  - [pylatex](#) + [TeX Live](#)
- Two reports per board:
  - Short-circuit test report
  - Functional test report
- Logs archived (zip) for full traceability
- Upload into the Serenity production database
  - Easy access & board history tracking

The screenshot shows the 'View Asset 00014' page in the Serenity production database. The 'Files' tab is highlighted with a red box and a red arrow. The asset details include:

- Asset Tag: 00014
- Status: Ready to Deploy (deployable)
- Asset Name: Sir Douglas Adams
- Serial: S1.4-0042
- Category: Board
- Model: Serenity-S1.4.1
- Model No.: S1.4.1
- BYOD: No
- Requestable: Yes
- PCB Manufacturer: SOMACIS
- PCB Assembler: KIT

The test report for 'Serenity-S Commissioning Test' is displayed, featuring the Serenity logo and the following details:

- 5 Loopback-Test
- Firmware directory: /root/FAT\_Firmware/s1\_vu13p\_max-2\_de
- Link speed: 25.78125 GHz
- Test duration: 5s

The report includes a table with the following columns: PRBS lock, Error count, and Outcome. The table shows 20 rows of data, all with 'TRUE' for PRBS lock, '0' for Error count, and 'PASSED' for Outcome.

PRBS lock	Error count	Outcome
TRUE	0	PASSED

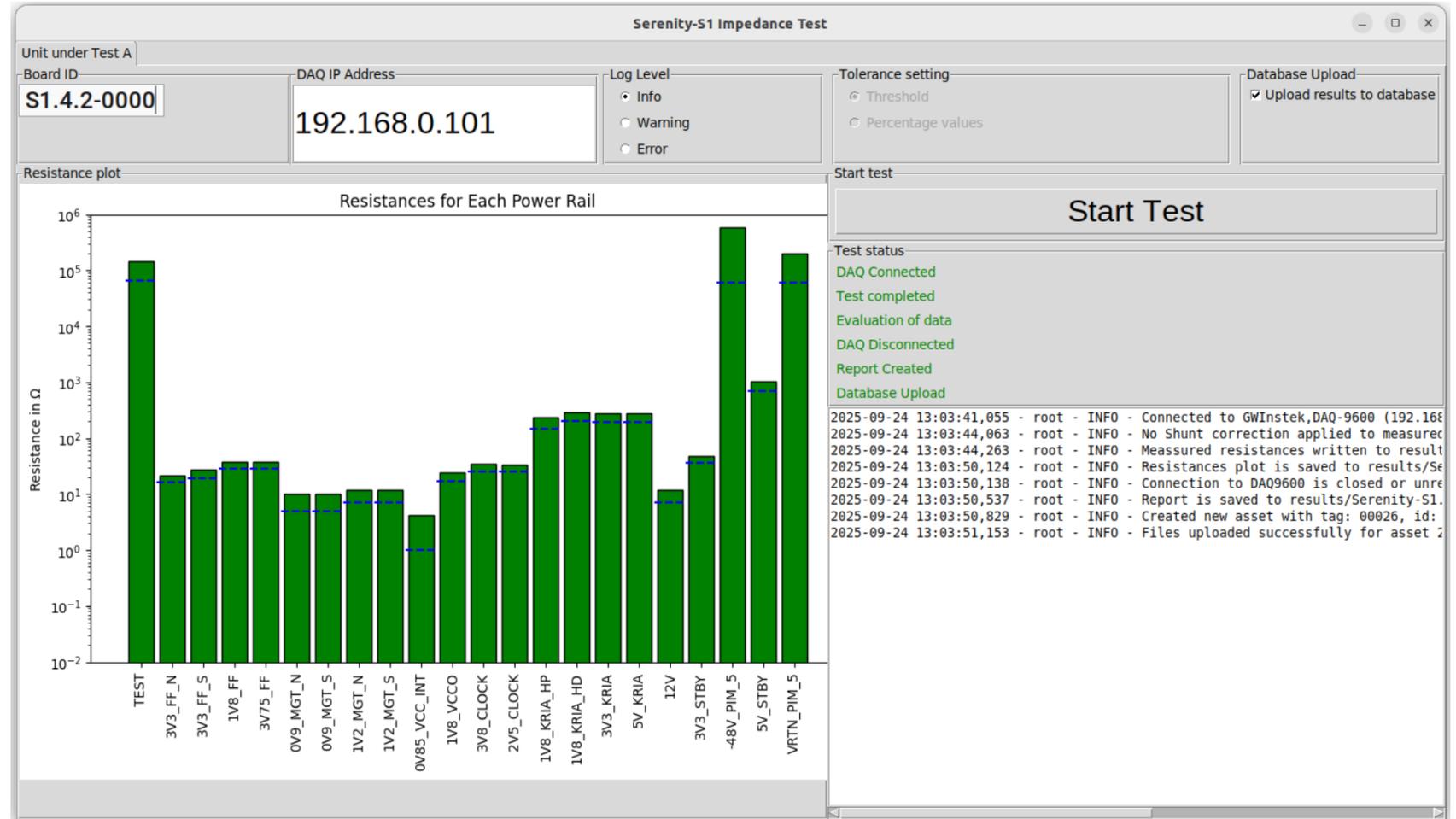
The report also includes an 'Abstract' section and a footer with the following information:

- Board ID: S1.4-0042
- Creation Date: 2025-08-19\_07-23-24

# How to Control the FAT?

## Python based GUI

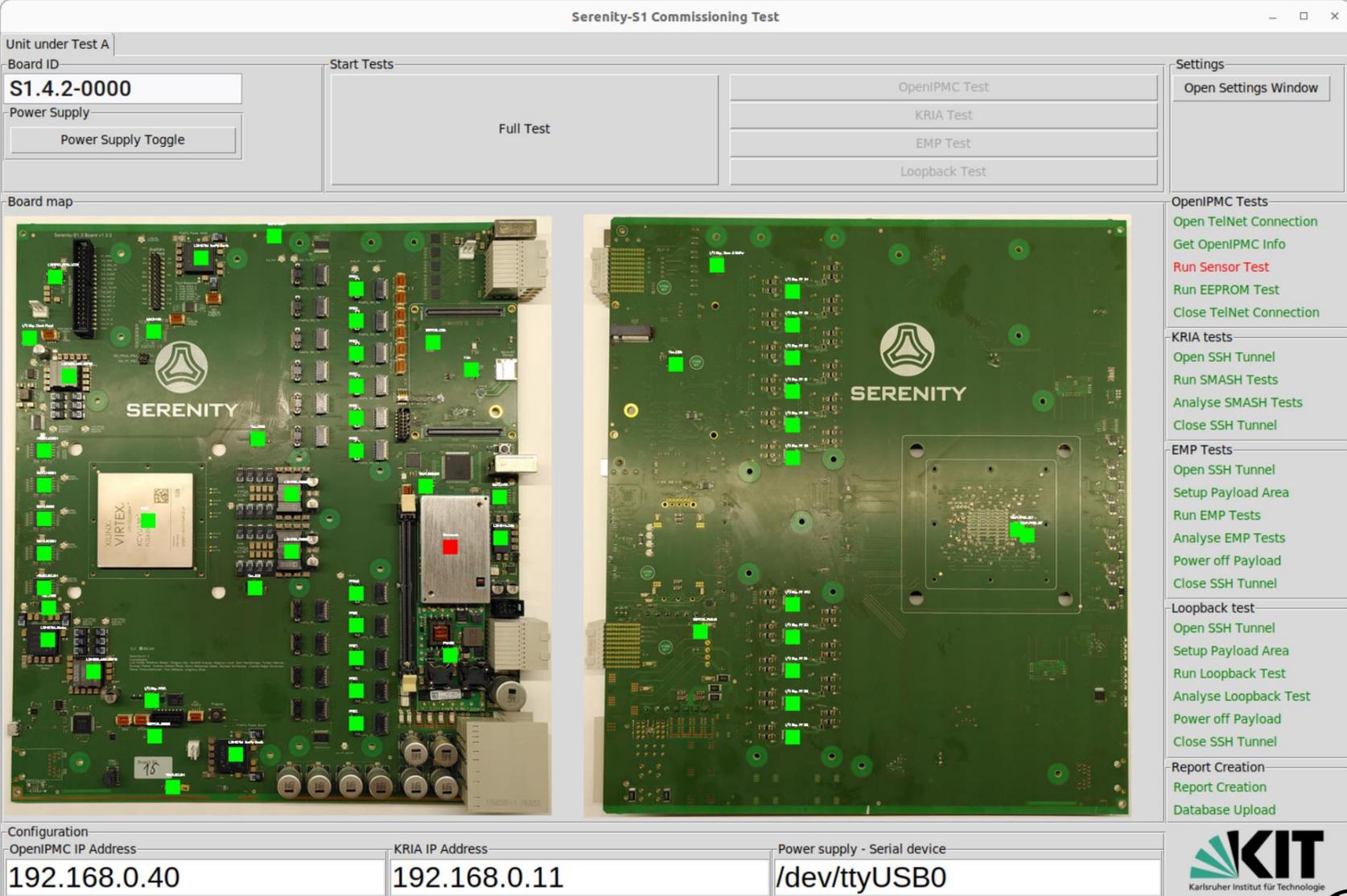
- Simple interface
- Safe to operate
- Automated test flow
- Feedback on test status



# How to Control the FAT?

## Python based GUI

- Simple interface
- Safe to operate
- Automated test flow
- Feedback on test status



Serenity-S1 Commissioning Test

Unit under Test A  
Board ID  
S1.4.2-0000  
Power Supply  
Power Supply Toggle

Start Tests  
Full Test

OpenIPMC Test  
KRIA Test  
EMP Test  
Loopback Test

Open Settings Window

Board map

Configuration  
OpenIPMC IP Address  
192.168.0.40  
KRIA IP Address  
192.168.0.11  
Power supply - Serial device  
/dev/ttyUSB0

OpenIPMC Tests  
Open TelNet Connection  
Get OpenIPMC Info  
Run Sensor Test  
Run EEPROM Test  
Close TelNet Connection

KRIA tests  
Open SSH Tunnel  
Run SMASH Tests  
Analyse SMASH Tests  
Close SSH Tunnel

EMP Tests  
Open SSH Tunnel  
Setup Payload Area  
Run EMP Tests  
Analyse EMP Tests  
Power off Payload  
Close SSH Tunnel

Loopback test  
Open SSH Tunnel  
Setup Payload Area  
Run Loopback Test  
Analyse Loopback Test  
Power off Payload  
Close SSH Tunnel

Report Creation  
Report Creation  
Database Upload

KIT  
Karlsruher Institut für Technologie



# Conclusions & Lessons Learned

## Conclusion (What we achieved):

- Extended QA beyond structural testing
  - Functional tests directly at the manufacturing companies
- Built on on-board controllers and existing board management tools
- Proven usage during extended pilot and pre-series production
  - FAT usage at CERN, KIT, TIFR and three Assembly companies
  - Successfully tested 60 Serenity-S1 boards

## Lessons learned:

- Adding debugging features to a board is very useful
- Having good and flexible board management tools pays off
- Build-In self test is a good approach for functional testing
- Pytest is suitable for testing hardware
- If you are dealing with non experts having a GUI is essential
- Having a clear test specification in the beginning helps a lot to simplify the development





**SERENITY**