Development of Readout Software for Timepix4-based Detectors

Viola Cavallini
on behalf of the 4DPHOTON and INFN - MEDIPIX4 Teams

N.V. Biesuz[2], R. Bolzonella[2,3], P. Cardarelli[2], V. Cavallini[2,3], A. Cotta Ramusino[2,3], M. Fiorini[2,3], M.Guarise[2,3], X. Llopart Cudie[1]

[1]CERN, [2]INFN Ferrara, [3]University of Ferrara

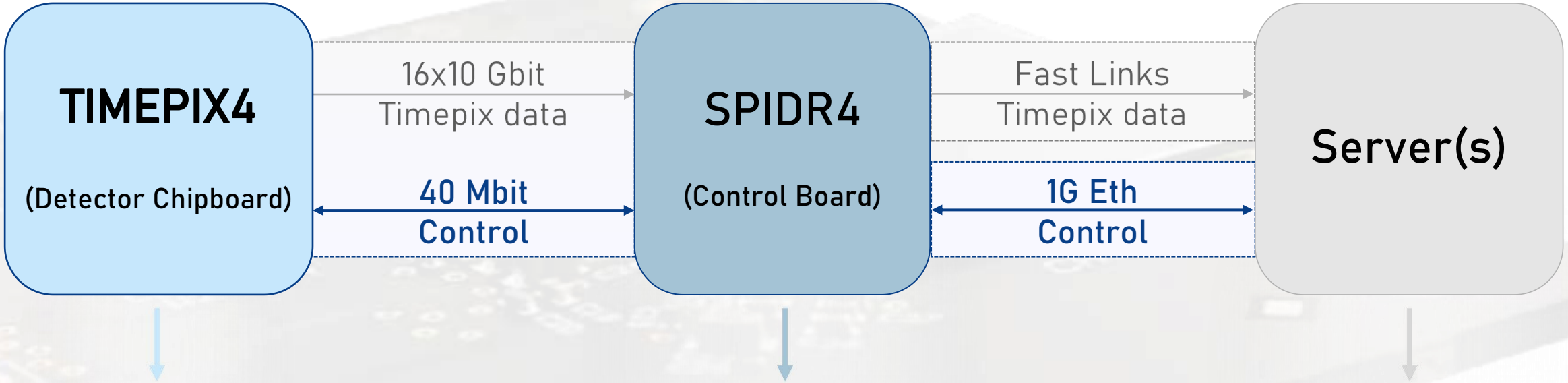
iWoRiD 2022 - Riva del Garda - 27/06/2022

Overview

- Constraints from the hardware architecture.
- The library architecture.
- How to use the library.
- Software adaptability.
- Code distribution.
- First results.

Our setup with Timepix4
and SPIDR4 (from NIKHEF)

The Data Acquisition System

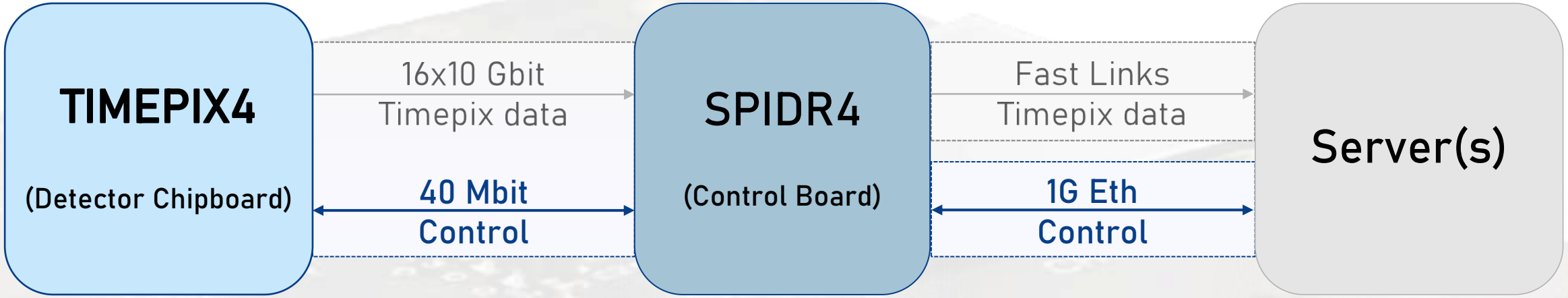


Signal collection and **digitalization**

Data-handling, interface to DAQ servers

System control, **data storage** and **post-processing**

The Data Acquisition System



Signal collection and **digitalization**

Fast enough to get all Timepix4 data

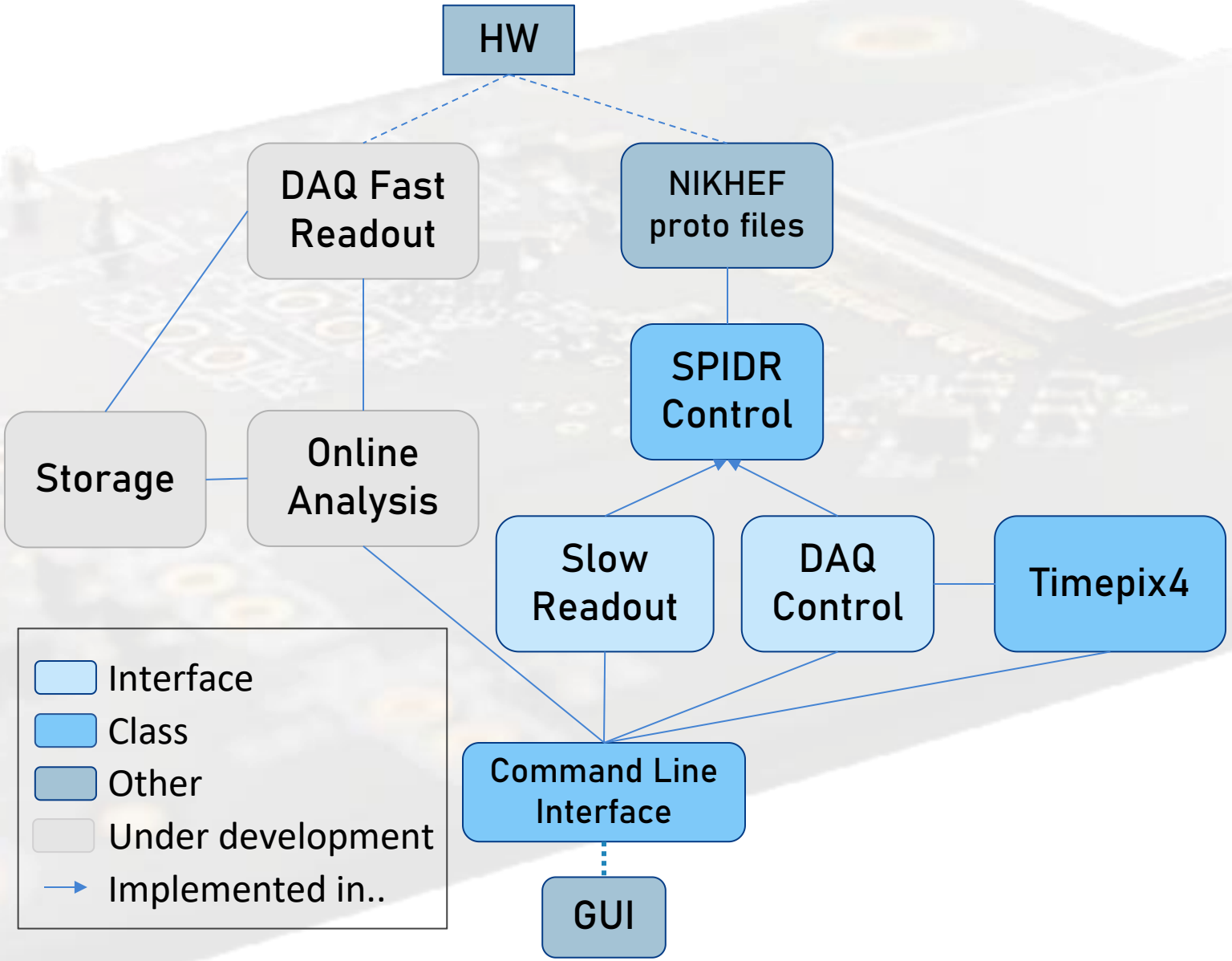
Data-handling, interface to DAQ servers

Adaptable enough to allow for changes in the hardware

System control, **data storage** and **post-processing**

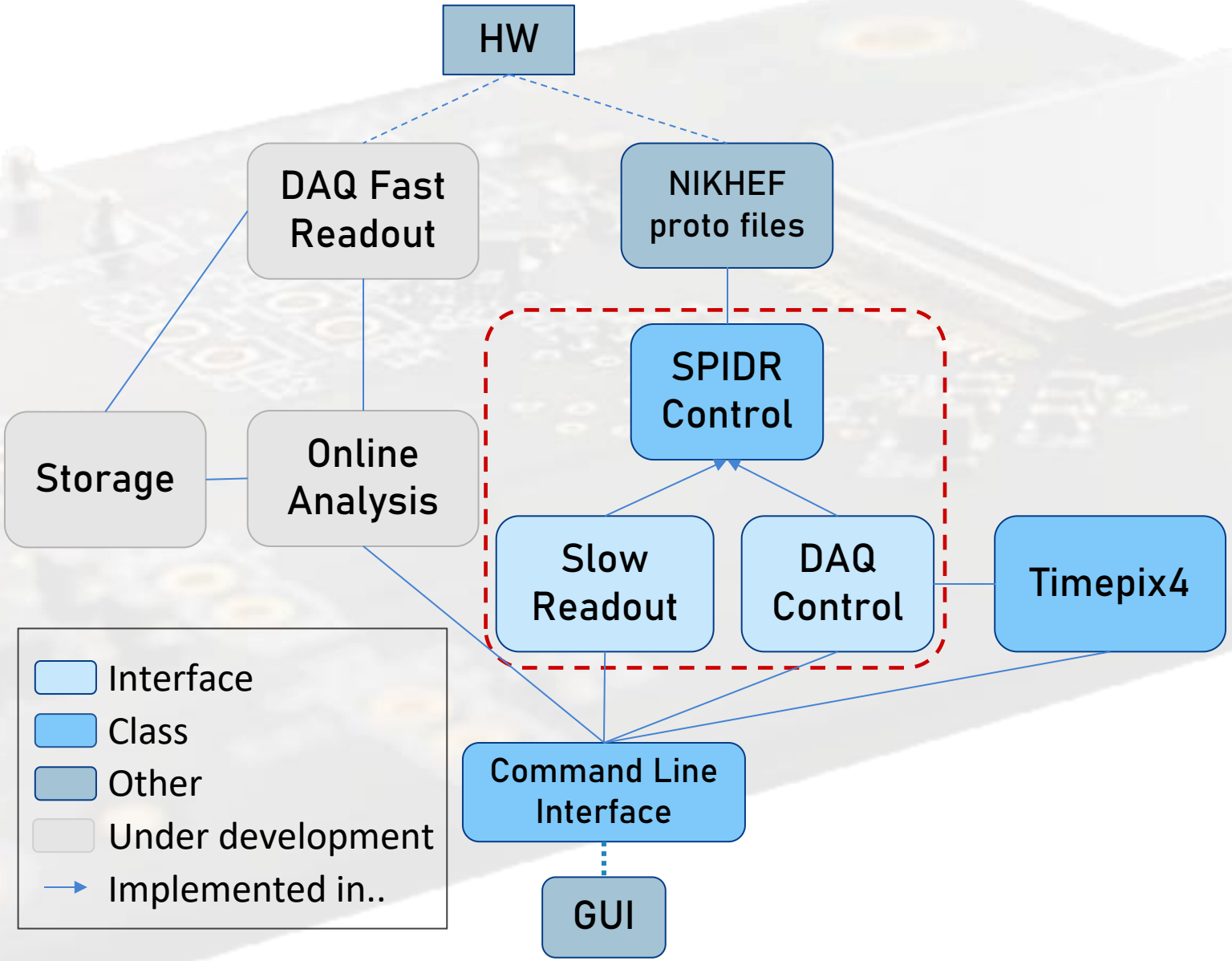
Flexible enough to allow the use of just one server or more

The Library



- **Written in C++**
  Fast, Optimized, Low-Level, Object-Oriented.
- **Readout and Control in unique CLI**
  All you need in a single Command Line Interface program.
- **Object-Oriented**
  Software that maps the hardware architecture.
- **Flexible Architecture**
  Designed to accommodate various DAQ systems with minimal changes.
- **Easy to expand**
  (Optional) bind to Python for fast plotting.

Software Architecture – Control



- **Control Class**

  Configure DAQ and Timepix4.

- **Read and Write Registers**

  Different functions: r/w a register, r/w just a field of the register, r/w using masks, …

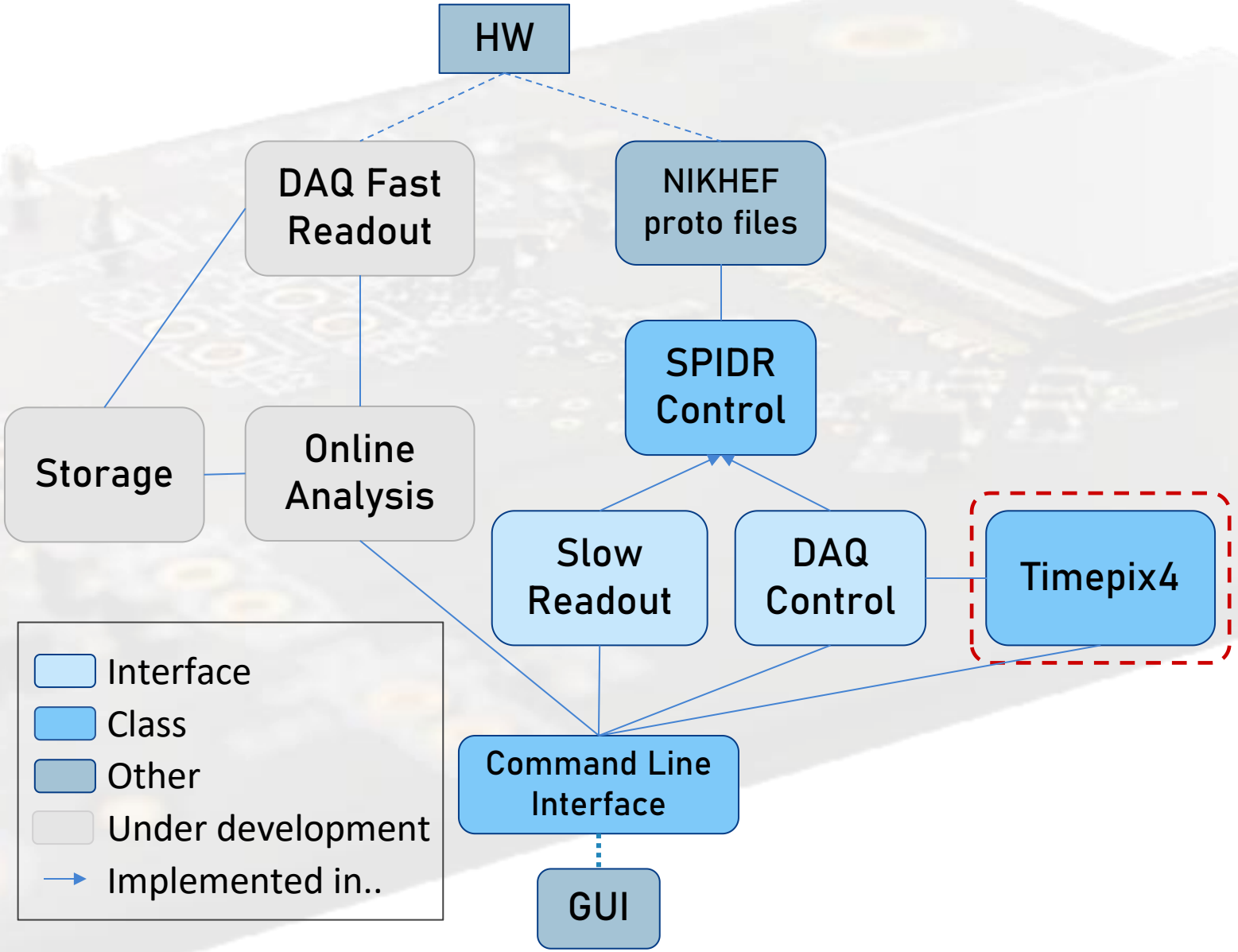
- **Manage Slow Control**

  Everything you can do via slow control is in this class, from configuration to slow readout.

- **Basic APIs for Timepix4**

  Such as reset, matrix configuration or set/get configuration functions.

Software Architecture - Timepix4



- **Class of information**

  Constants, list/structure of registers, list/structure of registers' fields, ...

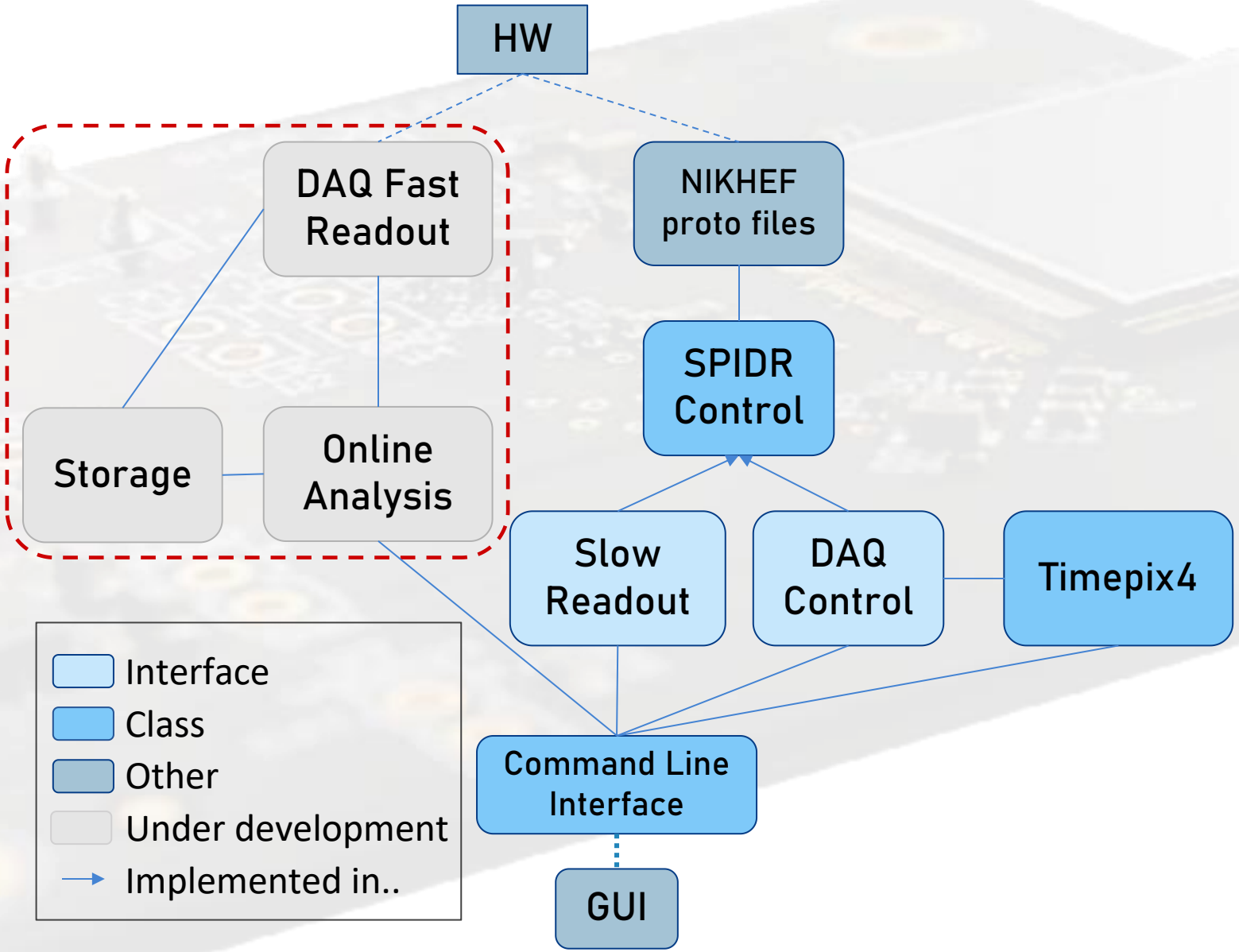
- **Helper functions**

  Functions to get informations about Timepix4 characteristics, such as get static information about a register and its fields or get register's name/ID.

- **Packet decoder**

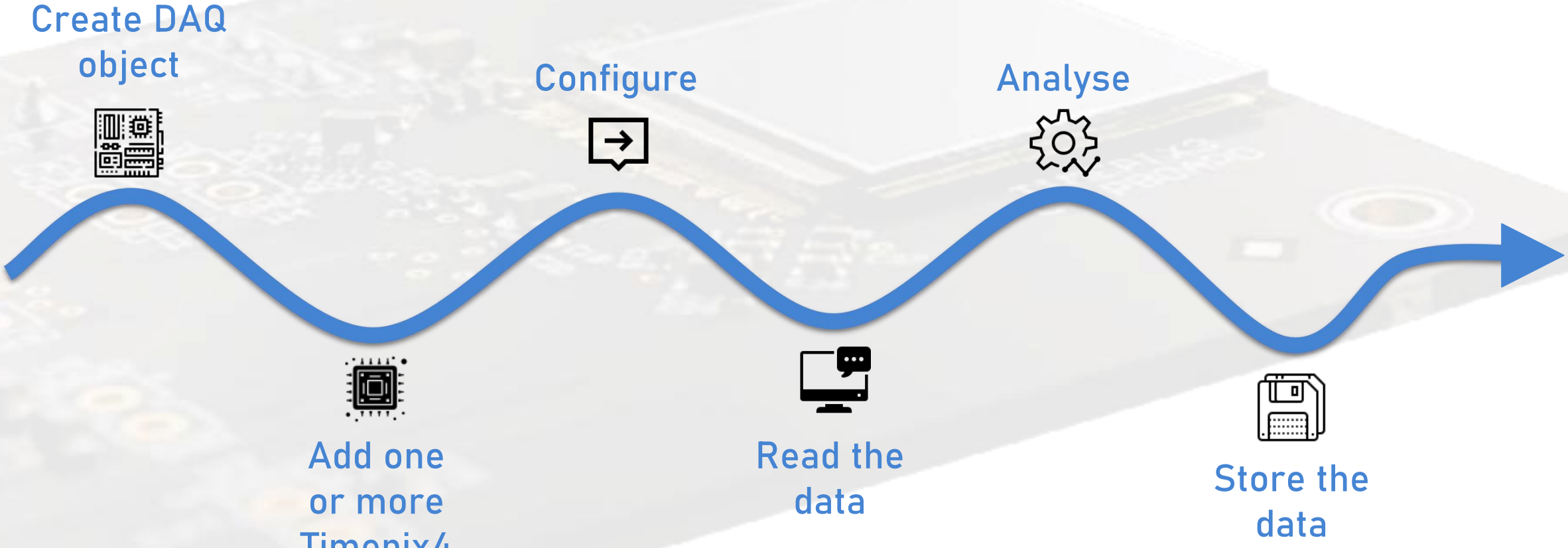
  Function to decode every kind of packet, both in data-driven and in frame-based mode.

Software Architecture - Readout



- **Readout module**
  Functions to capture every packet and store it correctly.
- **Online Analysis**
  Perform online selections and store to disk only relevant information.
- **Plotting**
  Easily implemented using C++ classes (e.g. ROOT) or binding to Python.

What you need to do to use it



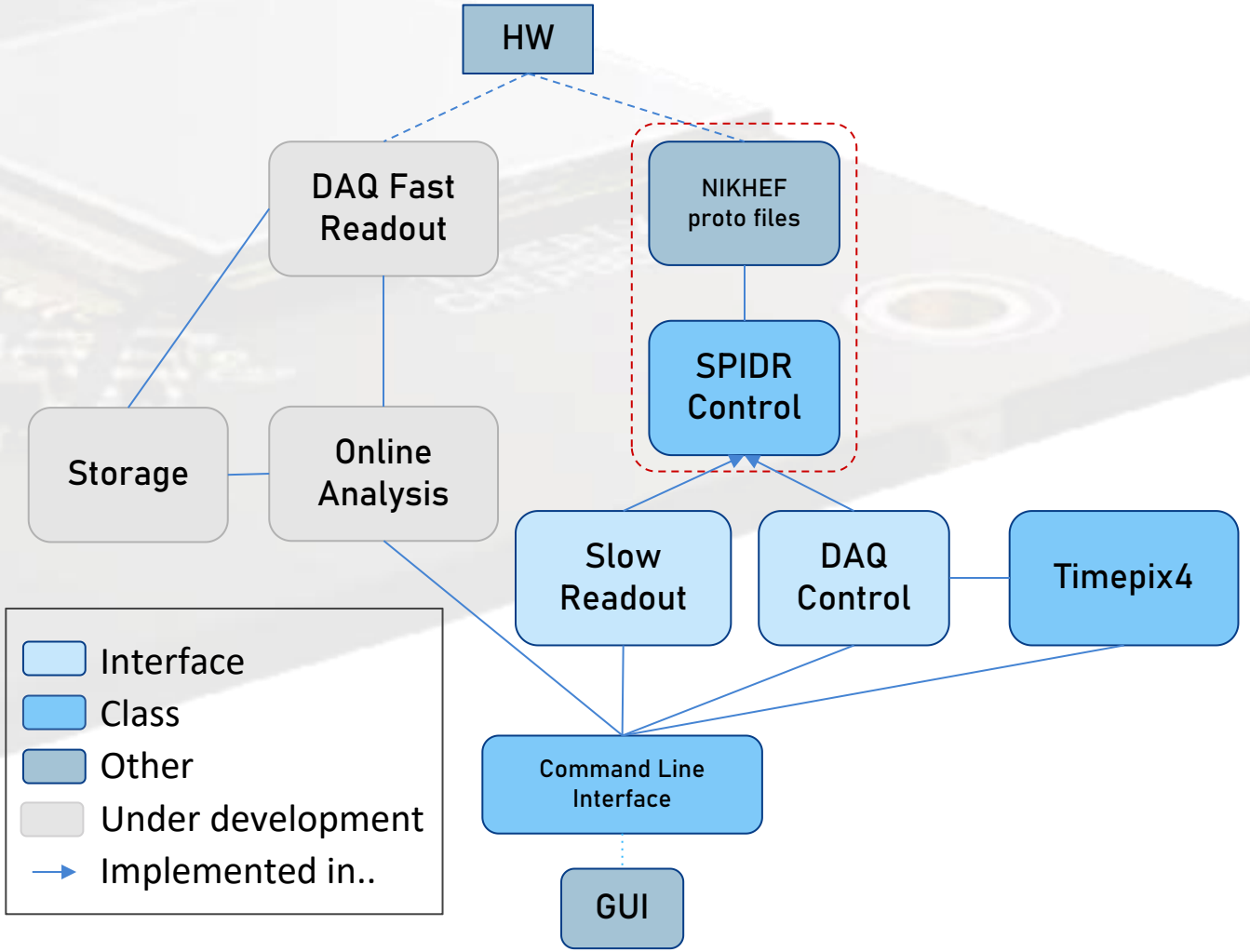
What you need to do to use it

```
#include “SPIDR.h”

// Create one (or more) DAQ object(s)
DAQ daq1 = new SPIDR(ip1);
DAQ daq2 = new SPIDR(ip2);

// Add some Timepix4
vector<uint16_t> tpx1IDs = daq1.addTpx(how_many = 4);
vector<uint16_t> tpx2IDs = daq2.addTpx(how_many = 1);

// Configure
daq1.configure_TP(**args);

// Get Information from Timepix4 Class
uint16_t addr = Timepix4::getRegAddr(“REG_NAME”);

// Read/Write Custom Registrers
daq1.writeTpx(tpx1IDs[0], addr, value);
daq2.readTpx(tpx2IDs[3], addr);

// Create and start readout thread
Thread thread1 = new scThread(daq1);
thread1.startReadout();
```

Build your own DAQ

You can use this software also with a hardware readout system different from SPIDR4.
You only have to write the following 4 methods:

- **Read a Timepix4 Register**
  Operations to read a register through SC/I2C
  Takes register address and returns vector of bytes
- **Write a Timepix4 Register**
  Operations to write a register through SC/I2C
  Takes register address and vector of bytes
- **Configure the DAQ**
  Operations to configure the control board
  Flexibility: desired configuration as JSON string
- **Read DAQ Monitoring Information**
  Operations to read back the control board
  Flexibility: returning configuration as JSON string

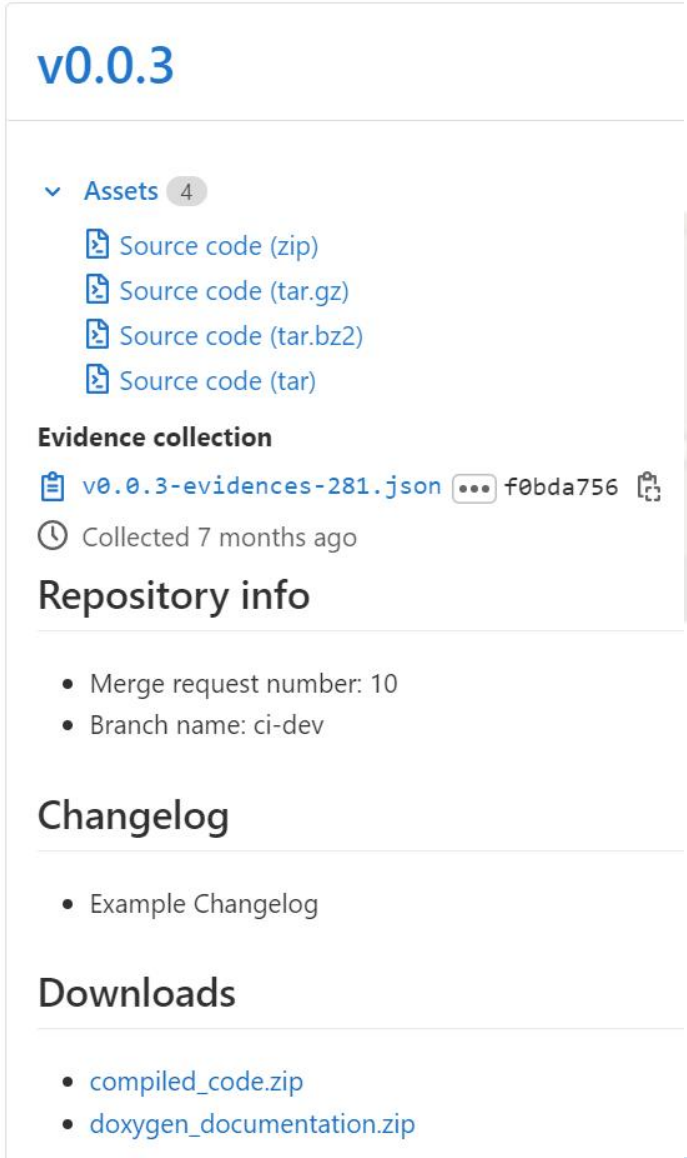


Code distribution

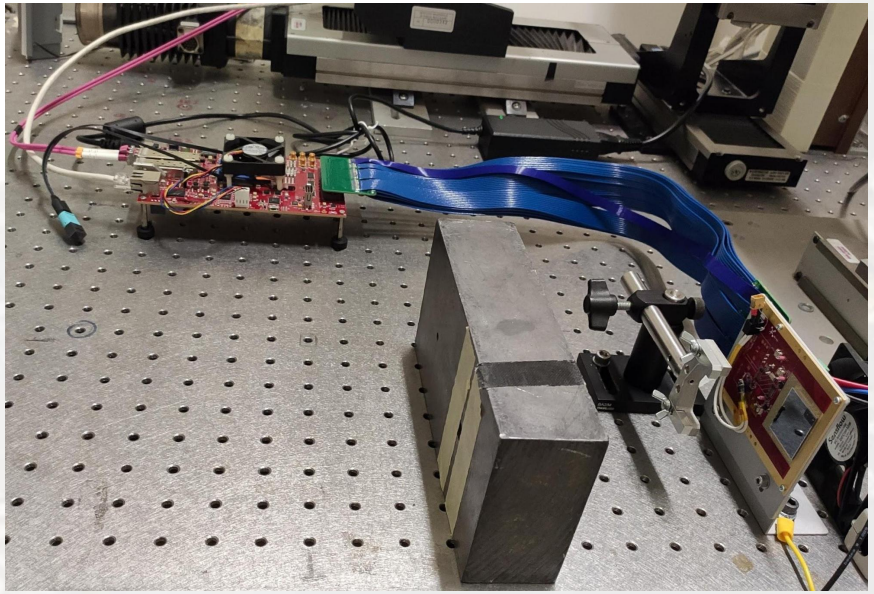
Code will be distributed as GitLab Releases

This will give access to:

- Release code
- Documentation
- Compiled library



Results



Source: Encapsulated ^{241}Am (340 kBq)

Distance: ~5 cm

Acquisition time: ~100s

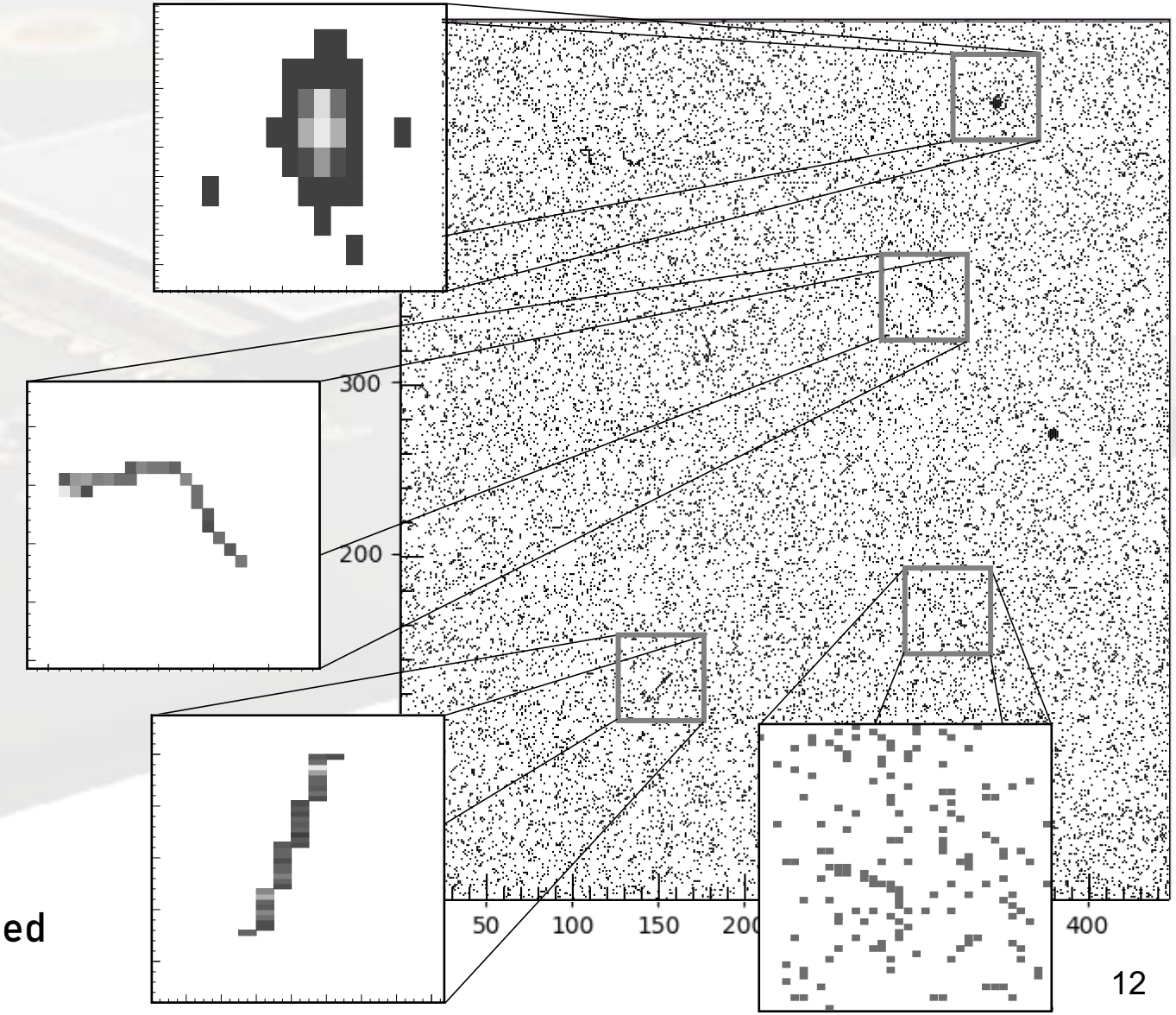


Image captured using Timepix4_v1 bump-bonded to a 300 micron thick Silicon sensor.

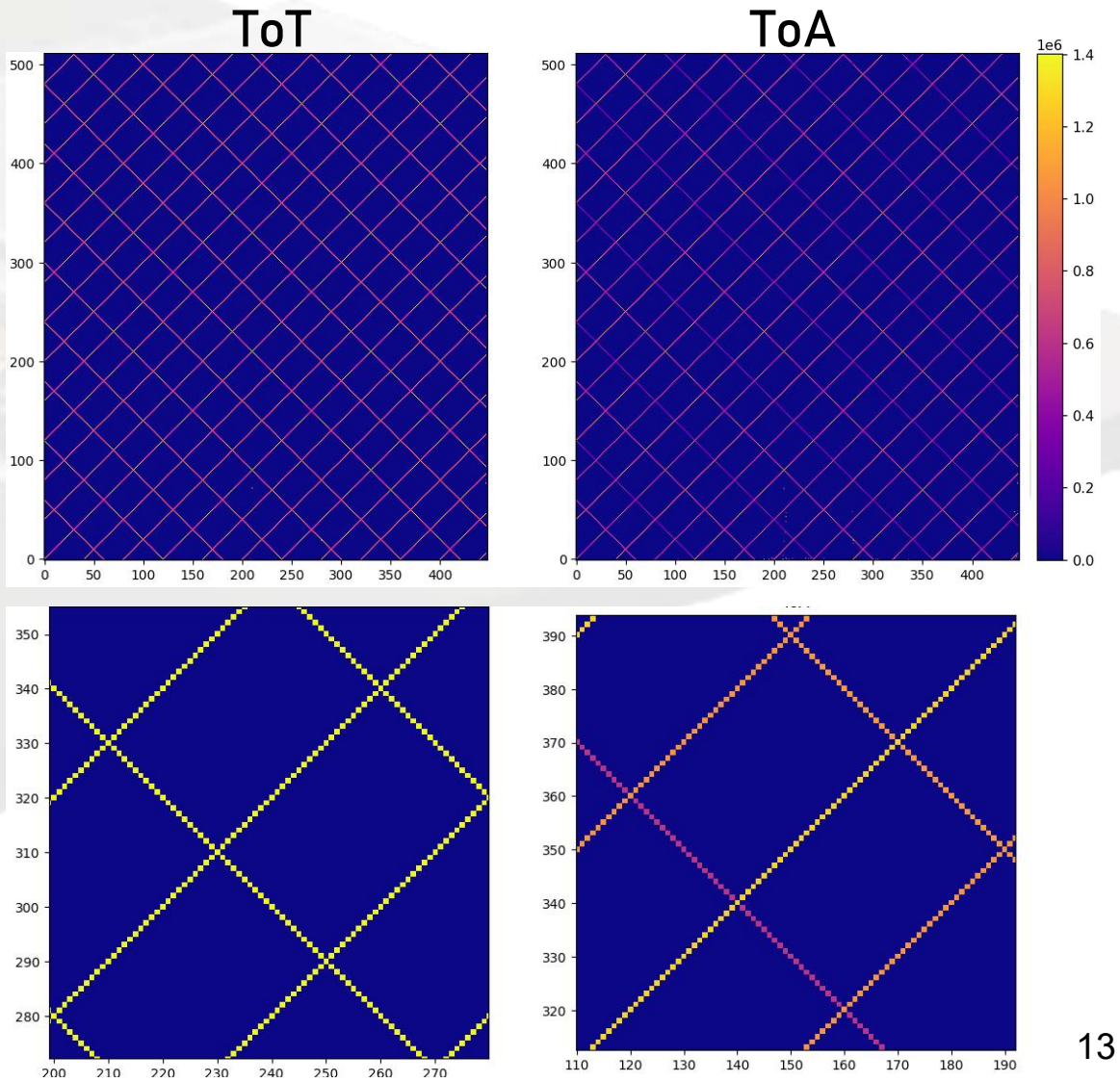
Results

Images obtained using testpulse.

Number of pulses
on different set of pixels: 4

Number of total enabled pixels: 9350

Two plots:
-Time over Threshold
-Time of Arrival



Summary

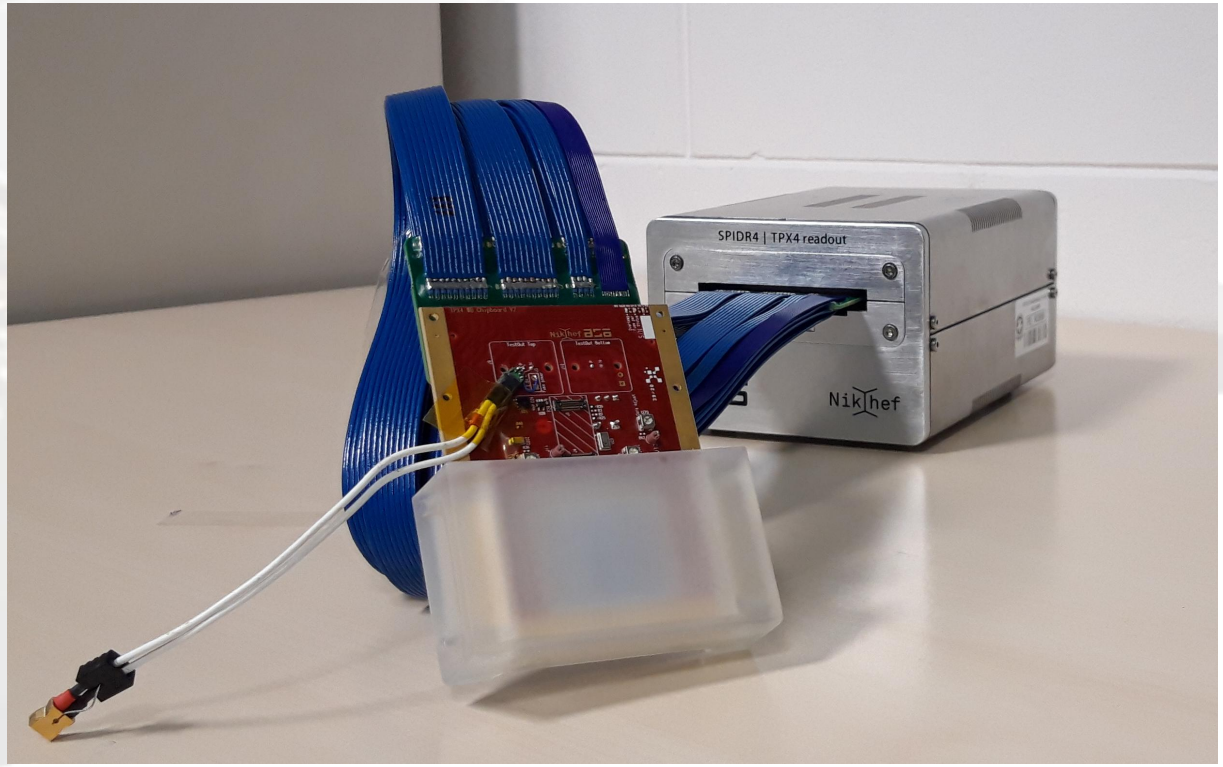
Software C++ under development.

Designed to be flexible enough to accommodate:

- different DAQ systems;
- different detectors based on Timepix4.

Will be released under EUPL-1.2 license:

- open-source;
- derived code needs to be released with compatible licence (copyleft).



You are welcome to use this software for your application!

If interested, please contact us.

Thanks to the CERN and NIKHEF teams for useful discussions, help and support.

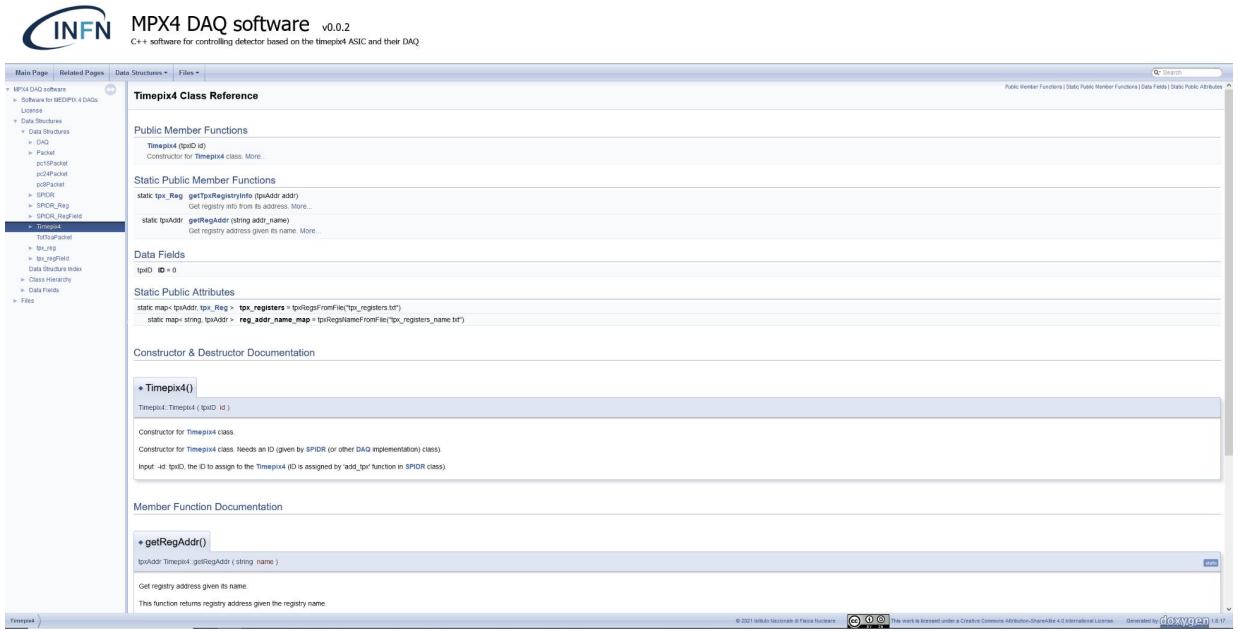
N.V. Biesuz, R. Bolzonella, P. Cardarelli, V. Cavallini, A. Cotta Ramusino, M. Fiorini, M.Guarise, X. Llopart Cudie

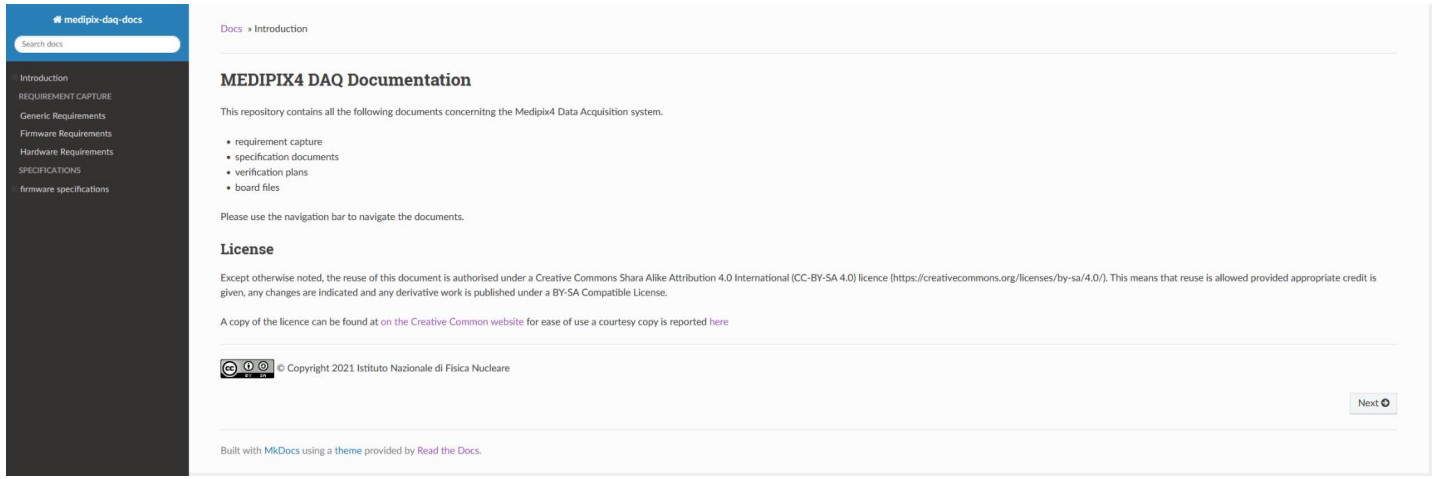
Back-up

Release Content

Software release includes:

- Source code
- Compiled libraries
  - Ubuntu 20.04
  - Others on request
- Full documentation
  - Low level (Doxygen)
    - Fully automated
    - From code
  - High level:
    - mkdocs:
      - How tos
      - Software architecture details
    - Release change-log:
      - Fully automated
      - Form commit messages





Current Status

```
  wc -l include/*
 122 include/DAQ.h
  50 include/SPIDR.h
  32 include/config.h
  27 include/packet.h
  36 include/timepix.h
 138 include/typedef.h
 405 total
  wc -l src/*
1402 src/DAQ.cpp
 299 src/SPIDR.cpp
 141 src/config.cpp
 160 src/packet.cpp
 268 src/timepix.cpp
   3 src/typedef.cpp
2273 total
  wc -l scripts/*
  89 scripts/DD_acquisition.cpp
  35 scripts/buildChangelog.sh
  58 scripts/buildRelease.sh
 105 scripts/helloTpx.cpp
  22 scripts/test_sc_testpulse.cpp
  21 scripts/test_write.cpp
 330 total
```

Configuration part:

- **Read Tpx4 registers**
  + conversions
- **Write Tpx4 registers**
  + variants, + conversions
- **Get external SPIDR4 sample**
- **Some APIs (~ 70)**

Testing scripts:

- **Read/Write regs + get sensors values**
- **Configure Tpx4 from file**
- **Use test-pulse**
- **Read data via slow control + plot (Python)**

To-Do Next:

- Read data via ***fast links***

```
void DAQ::set_conf_io(tpxID tpx_id, tpxIdx idx, int side) {…
void DAQ::enable_dac_out(tpxID tpx_id, tpxIdx idx, bool enable, int side) {…
float DAQ::ADC_Sample(tpxID tpx_id, tpxIdx idx, int OSR, int wait_ms) {…
int DAQ::reverse_Bits(int n, int no_of_bits) {…
void DAQ::setDacOut(tpxID tpx_id, tpxIdx idx, vector<int> selection, int AEOC) {…
float DAQ::PS_monitor_center(tpxID tpx_id, tpxIdx idx, int addr_13, int addr_23, bool printVal, int OSR) {…
uint16_t DAQ::Set_ADC_conf(tpxID tpx_id, tpxIdx idx, int clock_ref, int osr) {…
int DAQ::decodeDLL(int eoc_dll_read) {…
bool DAQ::DLL_is_locked(int eoc_dll_read) {…
void DAQ::power_up_dll(tpxID tpx_id, tpxIdx idx) {…
void DAQ::enable_dll_clk_col(tpxID tpx_id, tpxIdx idx, bool top, int col, bool enable) {…
void DAQ::set_dll_lock(tpxID tpx_id, tpxIdx idx, int window, int threshold) {…
void DAQ::set_dll_clk(tpxID tpx_id, tpxIdx idx, int lock_threshold, int toa_clk_edge, int clk_dll_freq_sel,
void DAQ::column_data_mask(tpxID tpx_id, tpxIdx idx, bool top, int col, bool mask, bool verbose) {…
void DAQ::unmask_all_column_data(tpxID tpx_id, tpxIdx idx) {…
void DAQ::set_pll_settings(tpxID tpx_id, tpxIdx idx) {…
void DAQ::reset_periphery_plls(tpxID tpx_id, tpxIdx idx) {…
int DAQ::read_clock_status_edges(tpxID tpx_id, tpxIdx idx, bool top, bool verbose) {…
void DAQ::reset_matrix(tpxID tpx_id, tpxIdx idx, bool reset_pixel_conf) {…
uint64_t DAQ::data_decode_sc(uint64_t val) {…
void DAQ::configure_TP(tpxID tpx_id, tpxIdx idx, int TP_number, int TP_period_on, int TP_period_off, int tp_
void DAQ::CTPR_enable(tpxID tpx_id, tpxIdx idx, int enable_top, int enable_bot) {…
void DAQ::CTPR_enable_col(tpxID tpx_id, tpxIdx idx, bool top, int eoc, bool enable) {…
void DAQ::mask_col(tpxID tpx_id, tpxIdx idx, bool top, int eoc, bool mask) {…
vector<uint64_t> DAQ::decode_packet(uint64_t packet, int mode, bool gray, int ratio_VCO_CKDLL) {…
uint64_t DAQ::decode_tot(int ftoa_rise, int ftoa_fall, int tot, int ratio_VCO_CKDLL) {…
uint64_t DAQ::decode_toa(int toa, int uftoa_start, int uftoa_stop, int ftoa_rise, int ratio_VCO_CKDLL) {…
int DAQ::decode_uftoa(int uftoa) { // !! Ritorna 0 se non lo trova oppure se è 15: corretto?…
int DAQ::inversegrayCode(uint64_t n) {…
void DAQ::reset_gwt_digital_analog(tpxID tpx_id, tpxIdx idx, bool top, int channels) {…
void DAQ::reset_gwt_fifo(tpxID tpx_id, tpxIdx idx, bool top, int channels) {…
void DAQ::reset_gwt_digital(tpxID tpx_id, tpxIdx idx, bool top, int channels) {…
void DAQ::prbs_pattern(tpxID tpx_id, tpxIdx idx, bool top, bool prbs_on, int channels, int prbs_mode) {…
void DAQ::reset_gwt_PLL_analog(tpxID tpx_id, tpxIdx idx, bool top, int channels, int wait_ms) {…
void DAQ::reset_gwt_DLL_analog(tpxID tpx_id, tpxIdx idx, bool top, int channels, int wait_ms) {…
void DAQ::start_up_links(tpxID tpx_id, tpxIdx idx, bool top, int channels, int high_BW_en, bool enable_cc_pll, int speed
void DAQ::router8x8(tpxID tpx_id, tpxIdx idx, bool top, int channels) {…
uint8_t DAQ::pixel_conf(int dac_bits, int power_enable, int tp_enable, int mask_bit) {…
void DAQ::save_IDACs(tpxID tpx_id, tpxIdx idx, string fbase, vector<string> DAC_I){};
void DAQ::save_VDACs(tpxID tpx_id, tpxIdx idx, string fbase, vector<string> DAC_VB){};
void DAQ::save_DACs(tpxID tpx_id, tpxIdx idx, string fbase, vector<string> DAC_I, vector<string> DAC_VB){};
void DAQ::load_DACs(tpxID tpx_id, tpxIdx idx, string fbase, vector<string> DAC_I, vector<string> DAC_VB, vector<string>
int DAQ::read_fuses(tpxID tpx_id, tpxIdx idx, bool top, bool verbose) {…
string DAQ::decode_chipID(int val) {…
int DAQ::tp_time(int time) {…
vector<int> DAQ::shutter_length(bool on, float time, bool verbose) {…
void DAQ::shutter_on_time(tpxID tpx_id, tpxIdx idx, float time, bool verbose) {…
void DAQ::shutter_off_time(tpxID tpx_id, tpxIdx idx, float time, bool verbose) {…
float DAQ::getTemperature(tpxID tpx_id, tpxIdx idx, int OSR, bool verbose) {…
```