

# JDRS for ToPix4

01.03.2016 | Alessandra Lai |

## JDRS: Jülich digital readout system

### Qt user interface

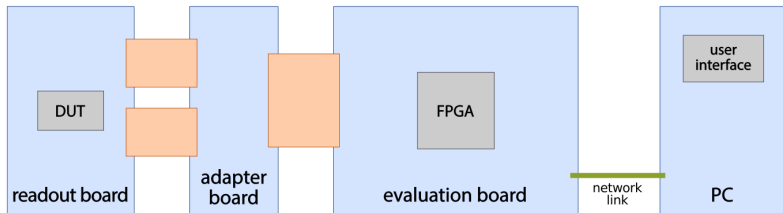
Former version

With new version come new features

### Git repository

### Summary

# JDRS: Jülich digital readout system



Data conversion and communication with the PC:

- ML605 evaluation board (Virtex-6 FPGA)
- firmware (VHDL)

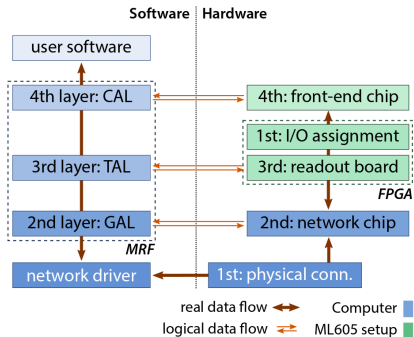
Configuration and data handling:

- PC
- **MVD readout framework (MRF)**
- **Qt-based GUI**

# MRF: MVD readout framework

Four abstraction layers isolate low level from higher level functions:

- physical layer
  - ethernet connection between ML605 and PC
- generic access layer (GAL)
  - data transfer and formatting e.g. open a connection, send and receive data packages...
- transport access layer (TAL)
  - board-specific functions e.g. the clock generation, flush of data buffers...
- chip access layer (CAL)
  - DUT-specific functions e.g. configuration and data readout...



## Qt - 'cute' framework

Qt is a widely used framework for developing application software with graphical user interfaces (GUIs) but not only (e.g. command-line tools).

- open source
- cross-platform (Linux, Windows, Android, Mac, ...)
- uses system resources (i.e. the app gets a native look)
- supports standard C++
- signals and slots mechanism (for event handling)
- supports several compilers (e.g. GCC, Visual studio)
- supports threading for parallel programming
- supports a designer for the layout of the UI
- ...

## Former version

Originally designed for beam test measurements → suffered from strict time constraints, quick implementation and workarounds:

- environment dependency
- lack of modularity and flexibility
- lack of structure
- hard coded settings and magic numbers in the code
- extreme sensitivity to external changes
- cumbersome for inexperienced users
- max reliable readout frequency  $\simeq 50$  MHz
- data from chip handled by *FairMQ*  
→ cumbersome and not necessary for lab measurements

The code to generate the whole GUI in a single file  
 $\approx 2000$  lines!

## Refactoring process: strategy

The idea is to make the existing framework modular by separating the functionalities in independent projects.

Each project consist of, at list:

- a *.pro file*
- a standard C++ class
- a form (i.e. the actual UI)

Rule of thumb: one project per tab.

### Caveat

Communication between projects is needed (e.g. an event that occurs in one tab might trigger an instruction in another tab).

One main window that contains all the other tabs as sub-widgets.

## Main window and widgets

Each widget:

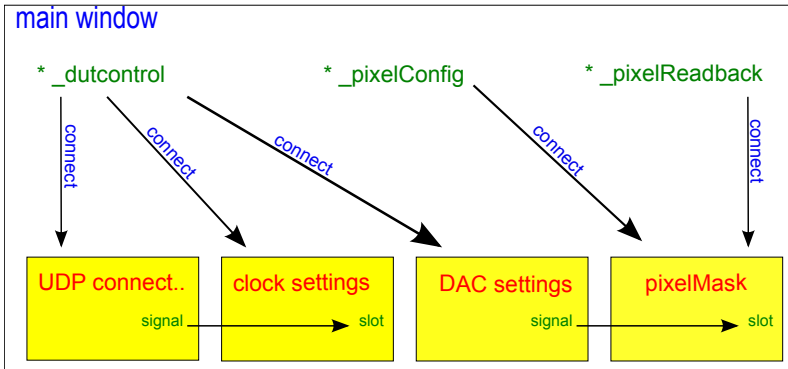
- is an independent project that can be executed standalone
- is included in the main window to build up the full gui
- is connected to the others, if needed, through the main window

The main window:

- holds the sub-widgets
- performs the connections between signals and slots
- initiates and distributes the global information



# GUI structure



## The MRF

The MRF is now a library that can be built independently.  
It has its own *.pro* file but no UI.

→ replace the files 'include' with the library include

```
SOURCES += main.cpp \
mainwindow.cpp \
topix4_fairmq_readout.cpp \
  ../writetofile.cpp \
  ../../MRF/source/mrfddata_chain2ltc2604.cpp \
  ../../MRF/source/mrfddata_chainltc.cpp \
  ../../MRF/source/mrfddataadv2d.cpp \
  ../../MRF/source/mrfddataadvbase.cpp \
  ../../MRF/source/mrfcal_topix4.cpp \
  ../../MRF/source/mrfcal.cpp \
  ../../MRF/source/mrftal_rbttopix4.cpp \
  ../../MRF/source/mrftal_rbbase_v6.cpp \
  ../../MRF/source/mrftal_rbbase.cpp \
  ../../MRF/source/mrfdataregaccess.cpp \
  ../../MRF/source/mrfddataadv1d.cpp \
  ...
HEADERS += mainwindow.h \
topix4_fairmq_readout.h \
  ../writetofile.h \
  ...
```

```
LIBS +=
-L${PROJECTPATH}/JDRS_core/MRF/lib -lMRF
```

## Load and save settings

Settings (e.g. configuration data) are stored in files *.json*

→ Qt offers support for JSON

- easy access to the key-value pairs in the files
- human readable format

```

"CommandCCR0"           : 32,
"CommandCCR1"           : 33,
"CommandCCR2"           : 34,
"CounterMode"           : 1,
"CounterEnable"         : 1,
"ReadoutCycleHalfSpeed" : 1,
"FreezeStop"            : 4,
"Leak_P"                 : 1,
"SelectPol"              : 1,
"PreEmphasisTimeStamp"  : 1,
"PreEmphasisCommands"   : 1,
"CounterStopValue"      : 4095,
"CalLevelDac"           : 5000,
"VCasIlc"                : 45580,
"VCasIfb"                : 40350,
"VRefBaseline"          : 37600,
"notUsed1"               : 0,
"notused2"               : 0,
"VRefD"                  : 37300,
"VCasD"                  : 32450

```

A similar format is available for the pixel configuration as well

- the key is the row number ( $nKeys = nRow = 32$ )
- the value is a 20 entries array ( $nCol = 20$ )

## Data handling

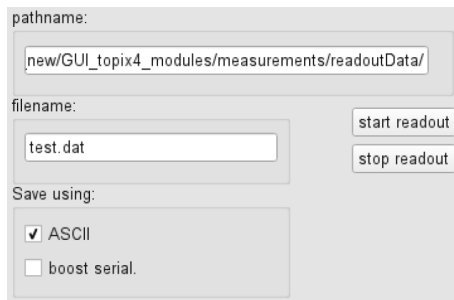
The data coming from the chip at the moment is not handled by *FairMQ*. The data buffer is read (and emptied) when needed and the data is stored in a file.

Two possibilities for data storage:

- ASCII
- boost serializer

From the GUI select:

- filename and path
- the saving method



pathname:

filename:

Save using:  
 ASCII  
 boost serial.

start readout  
 stop readout

The new interface allows to add new methods for storing the data (e.g. *FairMQ*, powerful for beam tests).

## Readback

One way to check if the process of writing to the chip was successful is to read back the data coming from it.

All the data that is written to the chip (configuration values, pixel status, etc) can be read back and visualized in the GUI.

For example the pixel mask status.



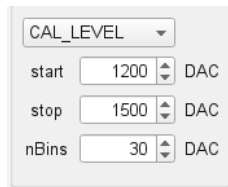
## Measurements

One part of the gui is dedicated to measurements.

The aim is the full qualification of the ToPix chip.

Each pixel has an internal circuit that allows the injection of a certain amount of charge.

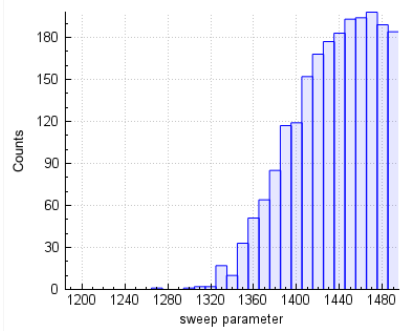
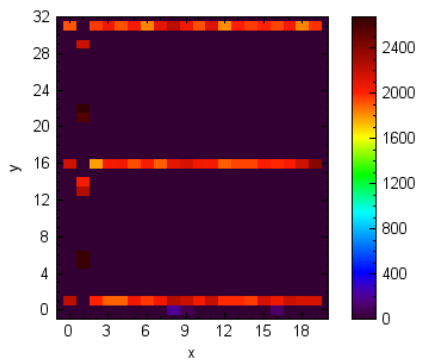
The charge is given in terms of  
DAC values.



A screenshot of a GUI control panel. At the top is a dropdown menu labeled 'CAL\_LEVEL'. Below it are three rows of controls, each consisting of a label, a numeric input field with up/down arrows, and the text 'DAC'. The first row is 'start' with the value '1200'. The second row is 'stop' with the value '1500'. The third row is 'nBins' with the value '30'.

# Measurements: visualization

Row 1, 16, 31 enabled.





## Git repository

The JDRS is under version control on Git (firmware and software)

- JDRS core (git submodule)
- JDRS ToPix

The core repository contains all the JDRS basic functionalities (UDP connection, register access, chip configuration, ...)

⇒ 100% reusable for PASTA

The ToPix repository contains ToPix specific functionalities

⇒ partially reusable for PASTA (adaptations are required)

It is sufficient to checkout the repo and run the JDRS (config file included).

Dependencies:

- boost library
- root

## Summary

- the JDRS is under development
- **before starting with the measurement campaign, the framework needs to be revised and restructured** ✓

Big steps towards modularity, maintainability, usability and flexibility have been achieved by:

- rearranging the code to match the GUI structure
- making the system as independent as possible of the environment
- reducing the dependencies from external libraries/packages although keeping the same functionalities
- improving the data handling and storage

Preparation for full characterization of the chip (at present ToPix, in a later stage PASTA) with automatic routines that can be handle from the GUI.