Karlsruhe Institute of Technology

KAPTURE-2 Guide

KARLSRUHE PULSTAKEING ULTRA-FAST READOUT ELECTRONICS

Documentation by

Matthias Martin
matthias@martin-tue.de

IBPT

July 2019

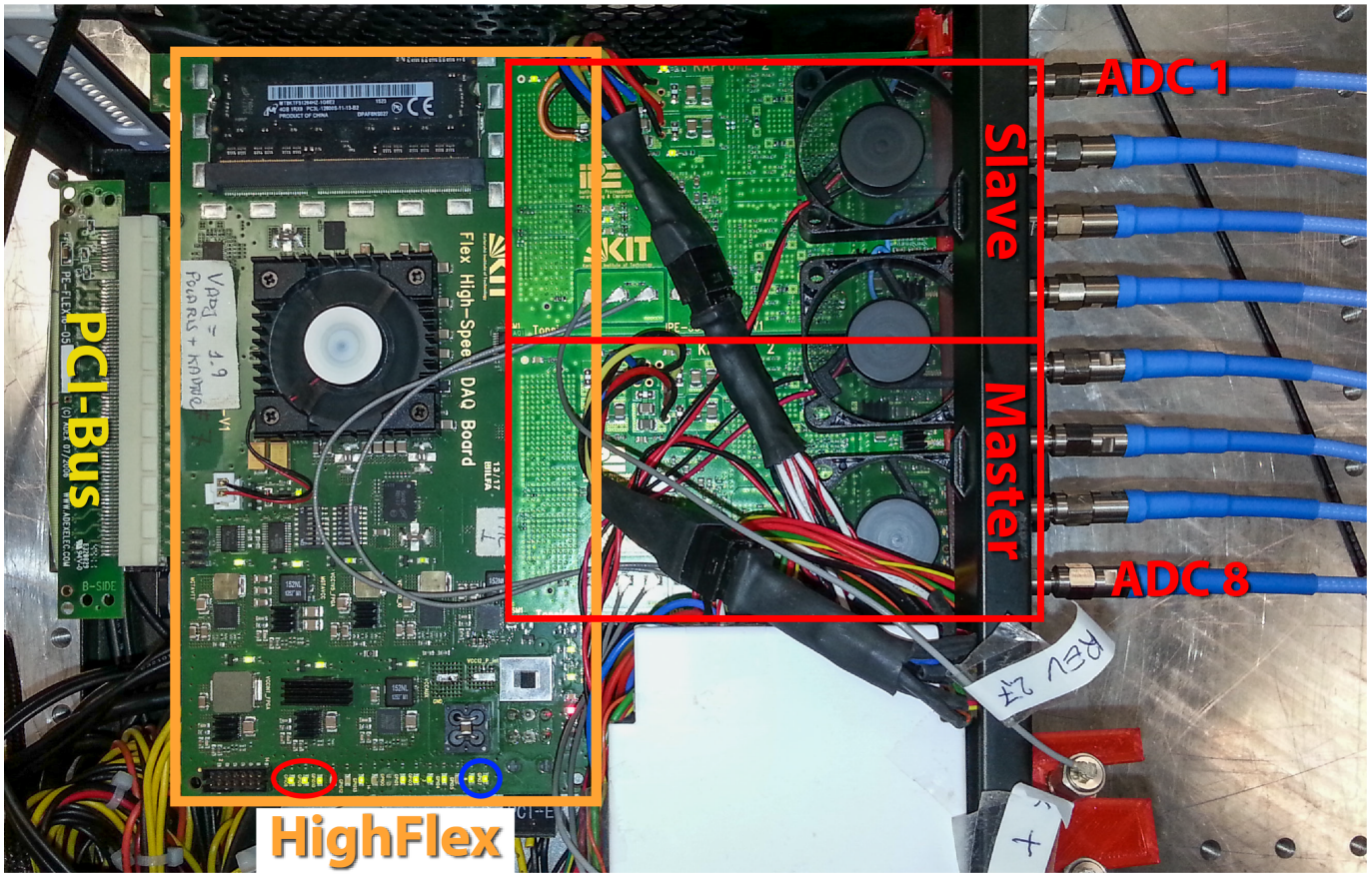
Inhaltsverzeichnis

1 Using Kapture 2

1.1 Measurement in a nutshell

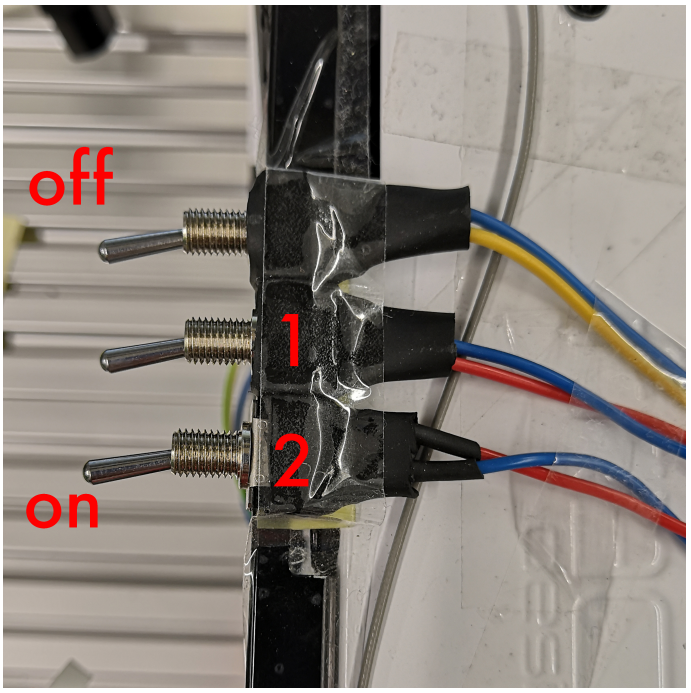
- start-up procedure: Boot PC, switch HighFlex off and on, reboot PC
- Prepare Board can always be used to reset the board
- check that PLL lock LEDs are on
- use internal pilot bunch
- activate HEADER
- T/H FMC2 25ps to 4 is no offset
- Using 500 MHz reference
    1. connect the 125 MHz
    2. do normal init with 125 MHz reference
    3. connect the 500 MHz
    4. run *PLL swap 125 to 500*, *PLL Synch*, *ADC Synch*
    5. now it should be working. If log is missing redo step 4.
    6. do not run *Board Reset* or other *PLL Init*! Then you would need to redo from beginning.
    7. If a DMA error etc happens try *soft reset*
- shutdown: Just turn off the PC and the two switches for the T/H.

1.2 start



The left Ethernet connector is configured for the KARA office network. The right one can be used for other networks.

Currently the Start-up of the system needs some manual workaround. Before turning on the PC put switches 1 and 2 in off state. The unlabeled switch is always on! Then Boot the PC when the Lock-in screen is visible turn the HighFlex of and on with the big switch on the Highflex. Then reboot the PC.



The two switches are to power down the input stage of the KAPTURE Modules. To power them up first turn on the switch 1 and than the switch 2! (if there is a longer period without measurement one can turn them back of to reduce thermal stress.)

Now you are good to go ;)

After starting the PC, start the GUI and hit 'Prepare Board' and you are ready to Measure at 500 MHz with 125 MHz as reference. To reset the Board one can just use Prepare Board. (Soft Reset and Board Off are not necessary for KAPTURE-2). After that DataFlow, Master Control and Data Check should be green. Also take a look on the LEDs of the HighFlex. The 3 inside the red circle indicate that the PCI is working (if one of them is not on - turn the HighFlex off and on and reboot). The 2 inside the blue circle indicate that both PLL are locked. If one is of, redo the prepare Board.

It can be also done manually. For example if the 500 MHz is used as input clock. Than first *Board Reset*, now the correct *PLL init* then *PLL Sync*, then *ADC Calib* and *ADC Autosync*.

On the MultiView Page one can find all the controls.

1.2.1 Basic Settings

1.2.1.1 Configfile

In 'home/user/.kcg2' are two config files. One for the general and one specific for epics. The Configfile is more or less self explaining. The newest option is the "Working Channels" config. It can be used to define which channels are used. It does not affect the RAW-file but it ins used for displaying in some cases. If the file does not exist a window opens where the config can also be edited.

The epics config is best to edit only via the epicsWidget.

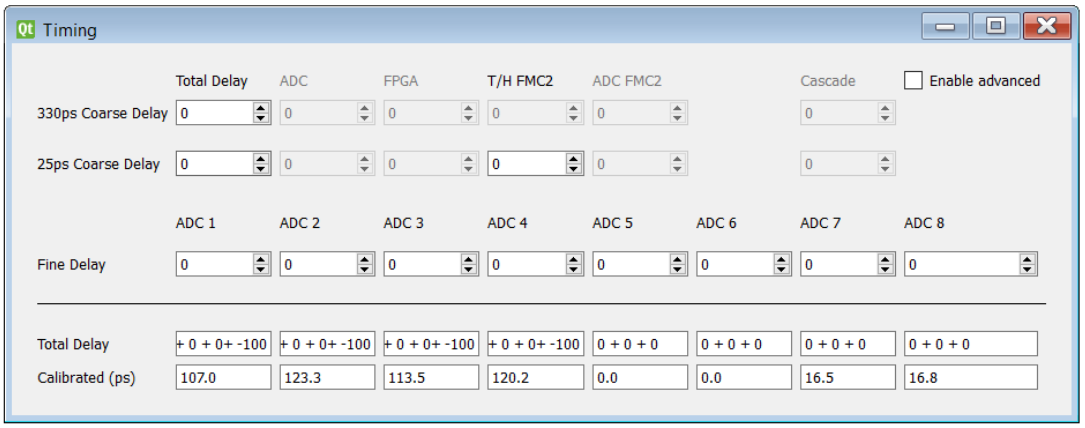
After a change, the GUI needs to be restarted.

1.2.1.2 file/settings

This is a small Settings window to set at runtime the Save Location and the Sub-directory.

1.3 Timing Widget

(Ctrl+T) contains the delay settings. KAPTURE-2 has 3 different delays.



330ps global Delay with 20 possible steps
25ps global Delay with 24 possible steps
3ps channel independent Delay with 32 steps

In addition the second KAPTURE Module can be delayed with the 25ps in the *T/H FMC2* column. The Default is 4! So for example value of 3 means that the second board will sample 25 ps earlier.

The advanced settings for ADC Delay and FPGA Delay are usually not to use. They are mainly for development. In normal user-operation the GUI handles them automatically.

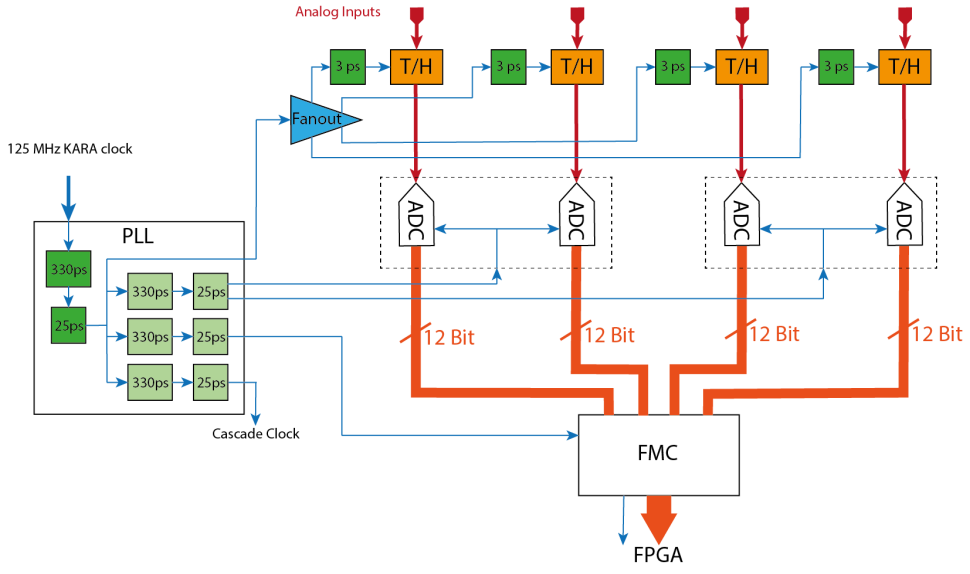
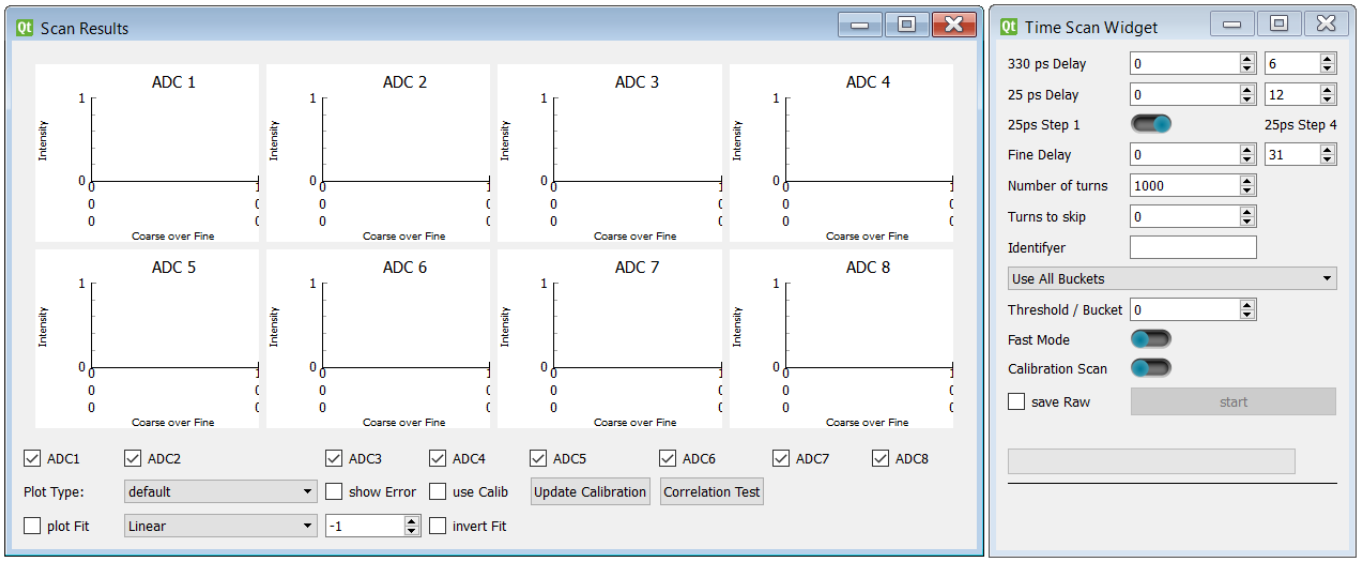


Abbildung 1.1: Timing distribution of a KAPTURE-2 Module. The Cascade Clock output of the first Module is used as input for the second modul. One Important information: The PLL is running with internal loopback to achieve a global Delay. This means that one output of the PLL is looped back to the input and by changeing the delay of this output, the complete System is shifted. But it is shifted in the oposite direction: increasing the delay means sampling earlyer. BUT the GUI takes care of this so that for the user it feels like it is shown in the graphic.

1.4 Time Scan Widget

(Ctrl+Shift+T)



The Time Scan was moved to a new Widget (mostly for code simplification). A TimeScan can be used to find the delays.

Via a drop down menu one can select how the measured data is processed.

- Mean over Everything: It just calculates the mean of the complete data. This is usually not to helpful.
- Threshold: Calculates the mean only for data points with not in 2048+ threshold. If you have a negative peak, enter a negative sign.
- Bucket: Calculates the mean of only on bucket - usually the best way.
- All Buckets: New Feature: It calculates the mean for all buckets individually. Therefore the resulting file is bigger but it allows to compare the buckets.

It offers different Scan Modes. In the standard way (as it is in the Screenshot) it scans step by step through all delays. In this mode usually one uses the 25 ps Step4 Option. If it is set to Step 1 one does over sample the Signal and increases the needed Time. The **Fast Mode** is primary designed to find the region of the Signal inside the 2 ns Bucket. In this case it sets spreads the 8 ADCs over the 100 ps area of the 3 ps. and then goes over the complete 2 ns domain with 25 ps Step 1. It is usually best to use with a threshold of 20. The **Calibration Scan** is only needed to generate the Calibration data with the analog 500 MHz signal and will take a long time. (Details in the Calibration Section)

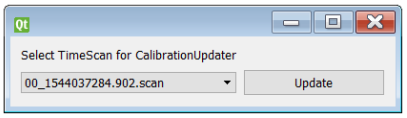
The Scan Results are stored in the sub folder `TimeScan`. In the logfile `scan.info` will the Scan Parameters be stored. The user has the possibility to use the identifier entry to add any text to the Logfile.

The Plotwindow offers multiple view. In **default** the data is plotted for each Channel separately over the delay setting. The other option is the **timecorrected** where the selected ADCs are plotted in one plot over Time. In this view also the Calibration can be activated and the Error can be plotted. In Addition a Fit can be performed.

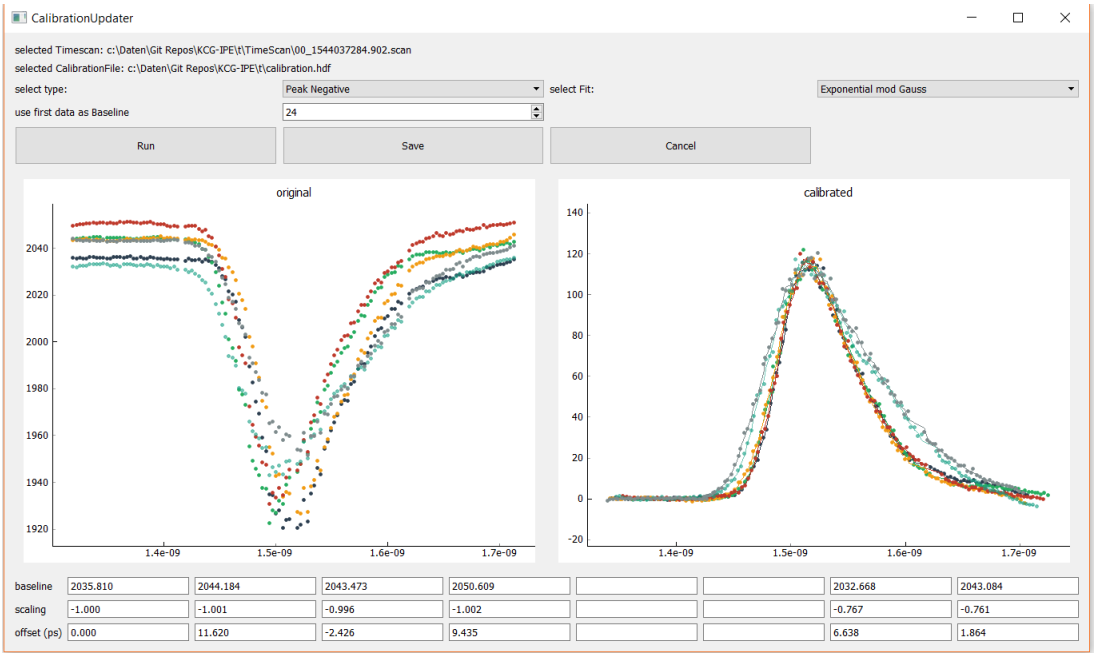
For simpler positioning of the ADC Timings one can plot them inside the Timescan via **show ADC** in the timecorrected view.

1.5 Calibration

 (Ctrl+U)



This offers a Calibration Routine. First a small widget opens where the TimeScan that will be used is to be selected. It is also Possible to open the Calibration directly from the Scan Results Widget. In this case this File selection is skipped.



The main Widget. The Left plot shows the original data with no calibration applied. The Right Plot than with all calibration used. The lower input lines can be used to manually adjust the calibration.

The Calibration is split into two Parts

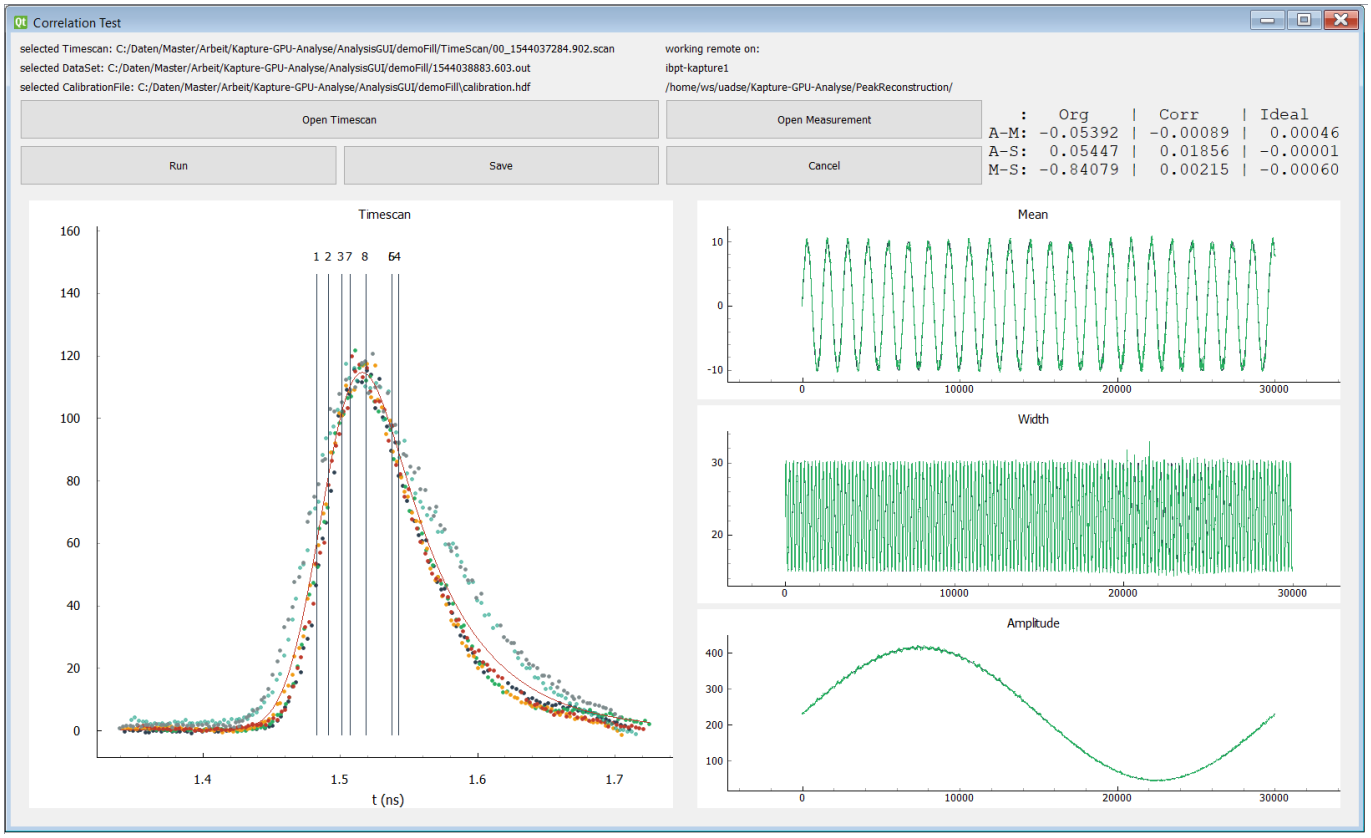
- 500 MHz:
  This needs only to be done once. Maybe it is useful to repeat it after one year or so. To do this one uses the 500 MHz RF clock as Input for the Channels and does a Calibration Scan. And then in this Widget select the type to be *500 MHz*. (the other Options have no effect in this case). It will take a view minutes! It calibrates the width of the Delays.

- Peak:
  This is the more often needed One. It should be done at least every time the Measurement setup is changed. This calibrates the Baseline, Gain and Time offset for the 8 Channels. To do that take a Timescan (usualy with step 4) of the Measurement Signal (as seen in the Screenshot). Now select if it is a Negative Peak or a Positive Peak. If there is a region with the Baseline in front of the Puls (like in the screen-shot) one can force the system to use that for Baseline determination via *use first data as Baseline*. If not or not wanted, set it to 0 (which is also the default).

To run the Calibration hit `Run`. This does not override the old one. If the result is good use `Save` to write it into the file. The old calibration file will than be renamed.

1.6 Correlation Test

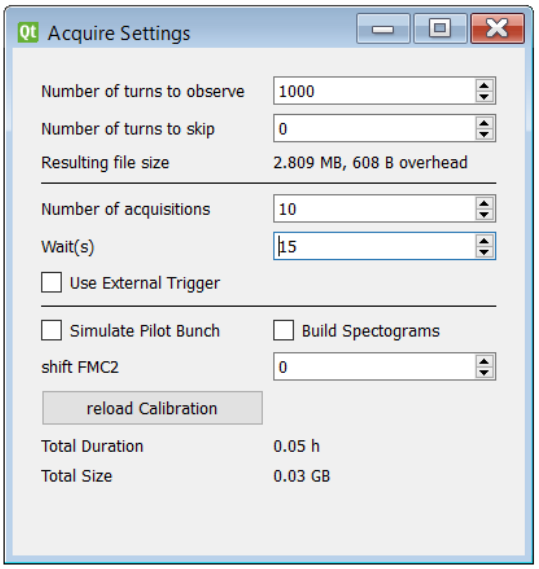
 (Ctrl+Shift+C)



This is only needed if a PeakReconstruction is later to be done. It can also be opened from the Scan Results Widget. With this, one can test if the used distribution of the Channels will give good results. Every time a SingeRead or a ContinuousRead is done the widget updates the distribution. To Perform the test hit Run and wait. On the current KAPTURE-2 System without a CUDA-GPU this is done remotely on ibpt-kapture1. If the result is pleasing one can save the Correlation Correction Parameters via Save so that they can be used later for the ReconstructionAlgorithm.

1.7 Acquisition

 (Ctrl+A)



The first to parameters define each acquisition, with the number of turns to observe and the number to skip. Be aware, that the turns to observe the skipped turns include. (e.g. If observe is 1000 and skip is 1, the file contains 500 data points).

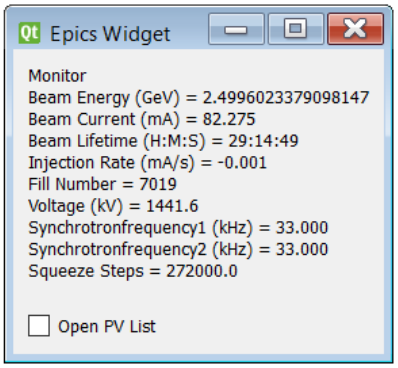
The option *shift FMC2* can be used if there is a shift between ADC1-4 and ADC5-8 witch is with the current firmware usually the case. It does not change anything in the raw file! The shift can change when the 330 ps is changed.

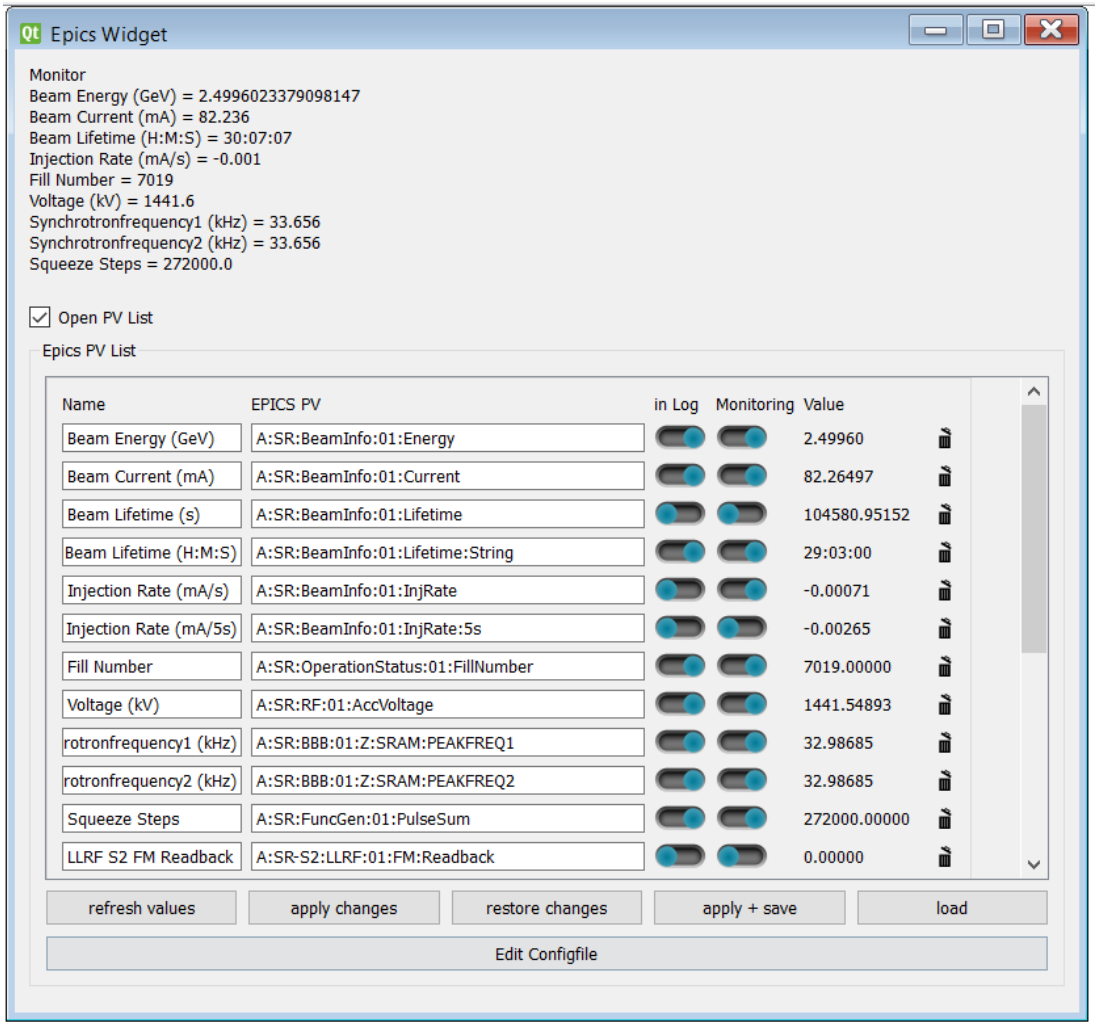
Currently also KAPTURE-2 is most stable without external revolution clock. Therefore *Simulate Pilot Bunch* needs to be checked.

1.8 Ploting

1.9 EPICS

 (Ctrl+E) The epics widget allows to read out parameters of KARA to store them in the Logfile for every acquisition. To change the known PVs open the PV List. Here you can edit the display name and the PV name as well as make them available for the Log and show them in the Monitor. To remove one entry use the garbage can icon. A new PV can be added in the last line of the table. Changes will only be done in the config file by *apply + save*. With *apply* the changes will be only temporally.


Epics Widget
Monitor
Beam Energy (GeV) = 2.4996023379098147
Beam Current (mA) = 82.275
Beam Lifetime (H:M:S) = 29:14:49
Injection Rate (mA/s) = -0.001
Fill Number = 7019
Voltage (kV) = 1441.6
Synchrotronfrequency1 (kHz) = 33.000
Synchrotronfrequency2 (kHz) = 33.000
Squeeze Steps = 272000.0
Open PV List


Epics Widget
Monitor
Beam Energy (GeV) = 2.4996023379098147
Beam Current (mA) = 82.236
Beam Lifetime (H:M:S) = 30:07:07
Injection Rate (mA/s) = -0.001
Fill Number = 7019
Voltage (kV) = 1441.6
Synchrotronfrequency1 (kHz) = 33.656
Synchrotronfrequency2 (kHz) = 33.656
Squeeze Steps = 272000.0
Open PV List
Epics PV List
Name
EPICS PV
in Log
Monitoring
Value
Beam Energy (GeV)
A:SR:BeamInfo:01:Energy
2.49960
Beam Current (mA)
A:SR:BeamInfo:01:Current
82.26497
Beam Lifetime (s)
A:SR:BeamInfo:01:Lifetime
104580.95152
Beam Lifetime (H:M:S)
A:SR:BeamInfo:01:Lifetime:String
29:03:00
Injection Rate (mA/s)
A:SR:BeamInfo:01:InjRate
-0.00071
Injection Rate (mA/5s)
A:SR:BeamInfo:01:InjRate:5s
-0.00265
Fill Number
A:SR:OperationStatus:01:FillNumber
7019.00000
Voltage (kV)
A:SR:RF:01:AccVoltage
1441.54893
rotronfrequency1 (kHz)
A:SR:BBB:01:Z:SRAM:PEAKFREQ1
32.98685
rotronfrequency2 (kHz)
A:SR:BBB:01:Z:SRAM:PEAKFREQ2
32.98685
Squeeze Steps
A:SR:FuncGen:01:PulseSum
272000.00000
LLRF S2 FM Readback
A:SR-S2:LLRF:01:FM:Readback
0.00000
refresh values
apply changes
restore changes
apply + save
load
Edit Configfile

2 GUI Developement

2.1 Pagackelist

- PyQT4
- sip
- pyqtgraph
- numpy
- psutil
- pyepics
- setuptools

2.2 misc

The Code is written to be working on both Python2 and Python3. At the moment this guide mostly only provides infos for the modules I developed or changed.

Before the Idea of KAPTURE2 came to live there was the idea to put multiple Kapture1 boards in on PC. So to the GUI a multi Kapture support was added. Unfortunately not in a perfect OOP fashion and therefore the code is now a little bit inconsistent. The new changes for KAPTURE2 mostly ignore things from the multi kapture implementation and therefore the current GUI version does not really support multi KAPTURE anymore - sry.

Also it should work also on the old KAPTURE-1 System but it has not been tested.

The configfiles and the icons etc are located in *.kcg2* to avoid conflicts with the old KAPTURE-1 GUI.

2.3 Modules

This is a not a complete list of all the modules with more or less small informations.

base/backend/board/communication

This contains the `class PCI`, witch wraps the system-calls for the pci communication to the FPGA. It also creates one instance of the class with the name `pci`. By using

```
from .communication import pci
```

one can then read and write to the FPGA.

There is also a dummy class which is selected when the gui is started with –testing parameter. It does not read or write to the hardware and can be therefore used while developing on a system without a KAPTURE board - even on a windows system.

base/backend/board/board_config

This contains the `class BoardConfiguration`. It is the central control Class for all the KAPTURE settings. (Like Delay, Turns to observe ...) It uses a dictionary to stores all the settings.

It is mainly base on observers. The function `update(key, value)` is used to change a setting. It then calls the observers. There are 3 Types of observers

1. observers_write:
   those are controlled by the class it self. They write the settings to the board.
2. observers_for_all:
   they are called every time one setting is changed - independent of the key
3. observers:
   they are called when the corresponding key changes. Those are mainly used to update the GUI.

The `update` function has an additional parameter `write=True` it controls weather the observers_write will be called or not. By default it is set to true. This is only needed for the function `read_from_board`, wich reads the settings from the FPGA, to prevent it from unnecessary rewrite it.

In widgets one can register a observer via the function `observe(who, callback, key)`
who is used as an identifier when one wants to remove the observer. It can be nearly everything - Object, variable - usually in the GUI it is the Object that will be changed (Like the label that is updated).
callback is the function that will be called and needs to have one parameter by which the new value will be passed.
key is the setting that will be observed.

When the widget will be deleted all corresponding observers need to be removed by `unobserve(who, key)`!

One example from acquiresettings widget

```
def __init__(...):
  self.board_config = board.get_board_config(board_id)
  self.fileSizeOutLabel = self.createLabel("??")
  self.board_config.observe(self.fileSizeOutLabel, self.set_filesize, 'turns_observe')
  .
  .

def set_filesize(self, state):
  .
  .

def closeEvent(self, event):
  self.board_config.unobserve(self.fileSizeOutLabel, 'turns_observe')
  .
  .
```

base/backend/board/sequences

The sequences are used to initialize the board. They are series of commands send to the FPGA. It is controlled by the `board_config` and from the backendinterface. The module contains two function:
`def read_sequence(board_version)`
`def run_sequnce(board_id, sequence, progressbar=None)`

The sequences are stored in json files in *base/backend/board/sequences/sequence_x.json* with x to be the `board_version`. The `board_version` is read by `board_config` from the FPGA.

All sequences in the "sequence_names" list will have a Button.
The "initialization_sequence_order" specifies which sequences will be run for *Prepare Board*

One Sequence is represented like this

```
"demo_sequence": {
      "Comment": "Text␣shown␣on␣Button",
      "status_val": "", #some Sequences set these to enable functions inside the gui
      "sequence": [
          [
              "value", "reg",
              "dialog␣text",  #If not an empty string a popup is shown before sending
              "comment",      #Printed in Logfile
              "key", "value", #Optional: used to update the board_config
              "key", "value"  #          It calls config.update(key, value, write=False)
          ],
          [
              "value", "reg",
              "",             #No popup
              "another␣command␣without␣update␣the␣board_config"
          ],
          [
              "value", "reg",
              "",
              "",
              "key", "value", #only one update of board_config

          ]
      ]
  },
```

base/backend/DataSet

Contains measured data. It has all the needed functions to open files, decode them and prepare them for plotting etc.

This Class is also used outside the KCG.

base/backend/TimeScan

Class to read and generate timescans files.

This Class is also used outside the KCG.

base/backend/CalibrationHandel

This Class handels the Calibration Files and is used by the `DataSet` and `TimeScan`. It contains also an instance of itself which should be used. So don't create a new one.

```
theCalibration = CalibrationHandel()
```

When ever one cals `theCalibration.openFile(...)` it returns a identifier.

base/backendinterface

The most messy module. It contains a vast amount of functions. Including:
wrapper functions for the `board_control`
functions to run the sequences
everything for data acquisition

widgets/

The most of the controlling is in the widgets. They can be understood as some kind of modules. To add new functionality to the GUI - like advanced analytic - one can just add a new Widget.

there are by now:

- AcquireSettingsWidget
- PlotWidget
- SingleReadWidget
- TimeingWidget
- TimescanWidget
- EpicsWidget
  Has one speciality: it is initialized in *base/kcg* via

```
if config.use_epics:
  from ..widgets import EpicsWidget
  EpicsWidget.epicsConfig = EpicsWidget.EpicsConfig()
```

- UpdateCalibrationWidget
- CorrelationWidget

To activate a widget put the module name in widgets/__init__.py

If a widget should be updated everytime a new Data is acquired. register a observer onto `"lastDataSet"`. Like it is done in the `PlotWidget`

```
def initUI(self):
  self.board_config.observe(self, self.observeDataSet, 'lastDataSet')

def observeDataSet(self, data):
  self.plot_live(data=data)

def closeEvent(self, event):
  self.board_config.unobserve(self, 'lastDataSet')
```

Do not forget to unobserve!

config.py

Module to handle the config file. It also provides some helperfunctions for accessing the current working path as well as the installation path. Also is it the place where the colors for the Plotwidget are defined.

2.4 FPGA stuff

Bank Register - KAPTURE-2

9000	Status_read_1	Status_read_2	Status_read_3	Status_read_4
9010	Status_read_5	Status_read_6	Status_read_7	Status_read_8
9020	TOTAL_ORBIT	TOTAL_ACQ	ORBIT_SKIP	ACQ_SKIP
9030	VERSION	ACQ_ACQUIRED	ORBIT_ACQUIRED	
9040	CONTROL	CONTROL_ADC	UPDATE_PATTERN	CONTROL_ADC_2
9050	STATUS_1	STATUS_2	STATUS_3	Number_of_errors
9060	SPI_PLL	SPI_ADC	SPI_FANOUT	DDR_RD_START
9070	DDR_WR_START	DDR_ADDR_WR	DDR_WR_END	DDR_RD_CUR
9080	LVDS_DELAY_CHIP	LVDS_DELAY_CHIP_F2		
9090	DELAY_ADC_1_2	DELAY_ADC_1_2_F	ADC_1_2_CLOCK_DIV	
90A0	DELAY_ADC_3_4	DELAY_ADC_3_4_F	ADC_3_4_CLOCK_DIV	
90B0	DELAY_TH	DELAY_TH_F	TH_CLOCK_DIV	
90C0	DELAY_FPGA		FPGA_CLOCK_DIV	
90D0	CASCADE_DELAY		CASCADE_CLOCK_DIV	
90E0	Harmonic number			
9100	PLL_SPI_DEBUG			
9120	ORBIT_CNT			

Abbildung 2.1: Bank Register