Gotthard-I Documentation

Release 0.3

A. Mozzanica and J. Zhang

November 03, 2016

This is a webpage documenting the Gotthard-I module information. For more details about the sensor, ASIC and readout of Gotthard-I, please refer to A. Mozzanica et al., JINST 7, C01019 (2012): http://iopscience.iop.org/article/10.1088/1748-0221/7/01/C01019.

This document gives a more practical information on the usage and characterization of the detector.

Document version: 0.3

Document contribution and revision:

A. Mozzanica (PSI), A. Parenti (XFEL.EU), D. Thattil (PSI), M. Turcato (XFEL.EU), J. Zhang (PSI)

Document history:

- V03: Add introduction of trigger signal and measurement with trigger
- V02: Formal release after corrections
- V01: Internal release

Contents:

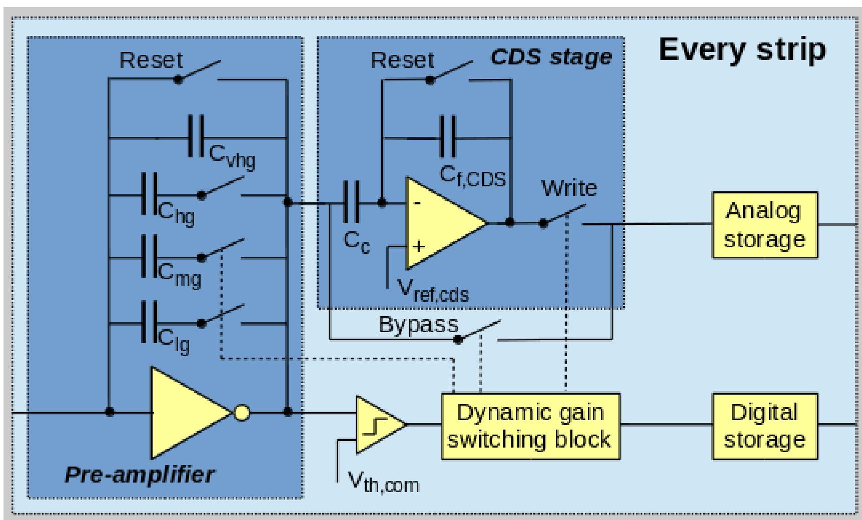
CHAPTER

ONE

INTRODUCTION TO GOTTHARD-I

1.1 Introduction

Gotthard-I is a charge-integrating silicon micro-strip detector with a pitch of 50 um (or optionally 25 um), and 1280 strips in total. It can be operated at < 1 MHz frame rate in burst mode and 40 kHz in continuous mode. The schematic of Gotthard-I ASIC can be seen as below:



Gotthard-I has a dynamic gain switching pre-amplifier to achieve high dynamic range, and a CDS stage to remove reset noise charge of the pre-amplifier. In dynamic gain switching mode, the CDS works before gain switching and is bypassed once gain switching happened. The detector can also work in a "fixed" gain mode, in which case only a constant gain applies. In "fixed" gain mode, the feedback capacitance of the pre-amplifier is fixed according to the input by users for detector operation, and the CDS stage is activated all the time.

In the following, the detailed information about how to configure and set-up the detector module, how to perform measurements and get data, and how to perform data analysis with basic routines will be introduced.

CHAPTER

TWO

SOFTWARES

The SLS detectors software is intended to control the detectors developed by the SLS Detectors group. It provides a command line interface (text client), a graphical user interface(GUI) as well as an API that can be embedded in your acquisitions system, some tools for detector calibration and the software to receive the data from detector with high data throughput (e.g. Gotthard).

2.1 The software package

The SLS detector software (slsDetectorPackage) can be downloaded through: https://www.psi.ch/detectors/users-support. The complete software package is composed of several programs which can be installed (or locally compiled) depending on the needs:

- The slsDetector shared and static libraries which are necessary for all user interfaces.
- The command line interfaces which are provided to communicate with the detectors using the command line and eventually to the data receiver
- The data receiver (slsReceiver), which can be run on a different machine, receives the data from the detector and interfaces to the control software via TCP/IP for defining e.g. the file name, output path and return status and progress of the acquisition
- The graphical user interface (slsDetectorGUI) which provides a user friendly way of operating the detectors with online data preview
- The calibration wizards (energyCalibrationWizard, angularCalibrationWizard) to analyze the data and produce the energy or angular calibration files (only for photon-counting detector and thus not an interest for Gotthard users)
- The Gotthard virtual servers to simulate the detectors behavior (however only control commands work, not the data acquisition itself)

2.2 Install softwares

1. Prerequisites for using the softwares

   The software is written in C/C++. It needs to be able to access the shared memeory of the control PC and communicate to the detectors over TCP/IP. Therefore the detector should receive a proper IP address (either DHCP or static) and no firewall should be present between th control PC and the detector.

   For installing the slsDetector shared and static libraries and the slsDetectorClient software, any Linux installation with a working gcc should be fine. The slsDetectorGUI is based on Qt4 with Qwt libraries. The calibration wizards are based on the CERN Root data analysis framework.

To compile the software you will need the whole Qt4, Qwt and Root installation, including the header files. To run the software, it is enough to have the Qt4, Qwt or Root libraries appended to the LD_LIBRARY_PATH. CERN Root is not mandatory if users perform data analysis with another program language.

For detector configuration and data acquisition, the minimal requirements can be summarized below:

- slsDetectorPackage: All detector related executables and libraries
- Qt-4.8.2, qwt-6.0.1 and Qwt3D: Necessary for the GUI

In addition to slsDetectorPackage, the Qt-4.8.2 software can be downloaded: ftp://ftp.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.8.1.tar.gz (or alternatively at http://doc.qt.io/qt-4.8), qwt-6.0.1: https://svn.code.sf.net/p/qwt/code/branches/qwt-6.0/, and Qwt3D: http://qwtplot3d.sourceforge.net/.

Installation of Qt-4.8.2:

```
> gunzip [qt_file_name].tar.gz
> tar xvf [qt_file_name].tar
> ./configure
> make
> make install
```

Installation of Qwt-related packages:

```
> svn co https://svn.code.sf.net/p/qwt/code/branches/qwt-6.0
> cd qwt-6.0
> qmake
> make
> make install
```

More information about the software installation can be found at the following link: https://www.psi.ch/detectors/UsersSupportEN/slsDetectorInstall.pdf

PS: If there are repositories including Qt-4.8.2 and Qwt existing, instead of using the fore-mentioned standard installation, simply try “yum install qt-devel qwt-devel root” for Scientific Linux, “apt-get install libqt4-dev libqwt4-dev root-system” for Ubuntu.

2. Export the libraries and executables through command line after software installation:

   - Qt library:

```
> export QTDIR=[.../.../]Qt-4.8.2
> export LD_LIBRARY_PATH=$QTDIR:$LD_LIBRARY_PATH
> export PATH=$QTDIR/bin:$PATH
```

   - g++ directory:

```
> export QMAKESPEC=$QTDIR/mkspecs/linux-g++
```

   - qwt directory:

```
> export QWTDIR=[.../.../]qwt-6.0.1
> export LD_LIBRARY_PATH=$QWTDIR:$LD_LIBRARY_PATH
```

   - Qwt3D:

```
> export QWT3D=[.../]qwtplot3d
> export LD_LIBRARY_PATH=$QWT3D:$LD_LIBRARY_PATH
```

   It is also recommended to put them into the ”.bashrc” file so that they do not have to be input for each start.

3. Compile slsDetectorPackage

   The slsReceiver and slsDetectorGui executables should be compiled before using:

```
> cd [.../.../]slsDetectorPackage
> make clean; make
```

Then export the libraries and executables:

```
> cd bin
> export LD_LIBRARY_PATH=$PWD:$LD_LIBRARY_PATH
> export PATH=$PWD:$PATH
```

The method mentioned above is the minimal effort to compile the slsDetectorPackage. Other compilation methods:

- make -> compile the library, the command line interface and the receiver
- make lib -> compile only the library
- make slsDetectorClient -> compile the command line interface (and the library, since it is required)
- make slsDetectorClient_static -> static compile the command line interface statically linking the library (and the library, since it is required)
- make slsReceiver -> compile the data reciever (and the library, since it is required)
- make slsReceiver_static -> compile the data reciever statically linking the library (and the library, since it is required)
- make slsDetectorGUI -> compile slsDetectorGUI - requires a working Qt4 and Qwt installation
- make calWiz -> compile the calibration wizards - requires a working root installation
- make doc -> compile documentation in pdf format
- make htmldoc -> compile documentation in html format
- make install_lib -> installs the libraries, the text clients, the documentation and the includes for the API
- make install -> installs all software, including the gui, the cal wizards and the includes for the API
- make confinstall -> installs all software, including the gui, the cal wizards and the includes for the API, prompting for the install paths
- make clean -> remove object files and executables
- make help -> lists possible targets
- make gotthard_virtual -> compile a virtual GOTTHARD detector server (works for control commands, not for data taking)

The path where the files binaries, libraries, documentation and includes will be installed can either be defined interactively by sourcing the configure script (not executing!) or during compilation using make confinstall or defined on the command line deifning one (or all) the following variables (normally INSTALLROOT is enough):

- INSTALLROOT -> Directory where you want to install the software. Defaults to PWD
- BINDIR -> Directory where you want to install the binaries. Defaults to bin/
- INCDIR -> Directory where you want to pute the header files. Defaults to include
- LIBDIR -> Directory where you want to install the libraries. Defaults to bin/
- DOCDIR Directory where you want to copy the documentation. Defaults to doc/

To be able to run the executables, append the BINDIR directory to your PATH and LIBDIR to the LD_LIBRARY_PATH. To run the GUI, you also need to add to your LD_LIBRARY_PATH the Qt4 and Qwt libraries, without the need to install the whole Qt and Qwt developer package:

- libqwt.so.6

- libQtGui.so.4
- libQtCore.so.4
- libQtSvg.so.4

More options and information about software installation can be found in https://www.psi.ch/detectors/UsersSupportEN/slsDetectorInstall.pdf.

2.3 Upgrade softwares

The softwares are released through the following webpage: https://www.psi.ch/detectors/users-support. It is recommended to check the new release there. The client, receiver and detector server have to be updated at the same time. To update the detector server, follow the instruction below:

1. Server binary preparation

   First, check the location of the tftp directory:

```
> more /etc/xinetd.d/tftp
```

   The tftpboot directory will be shown after the "server_args". Here in the test machine, it is "/tftpboot". If no tftp exists, download and install it: http://askubuntu.com/questions/201505/how-do-i-install-and-run-a-tftp-server.

   The, copy the server binary to tftp directory:

```
> cp [.../.../]slsDetectorPackage/slsDetectorSoftware/gotthardDetectorServer/
/gotthardDetectorServer /tftpboot/
```

2. Update the detector server

   After powering on the detector, do the following in the command line:

```
> telnet bchip050   (either the hostname or the IP address of the detector)
> ps   (list the running processes)
> killall gotthardDetectorServer   (stop the currently running server)
> tftp -r pc_name gotthardDetectorServer -g
> chmod 777 gotthardDetectorServer
> ./gotthardDetectorServer &   (start the new server)
```

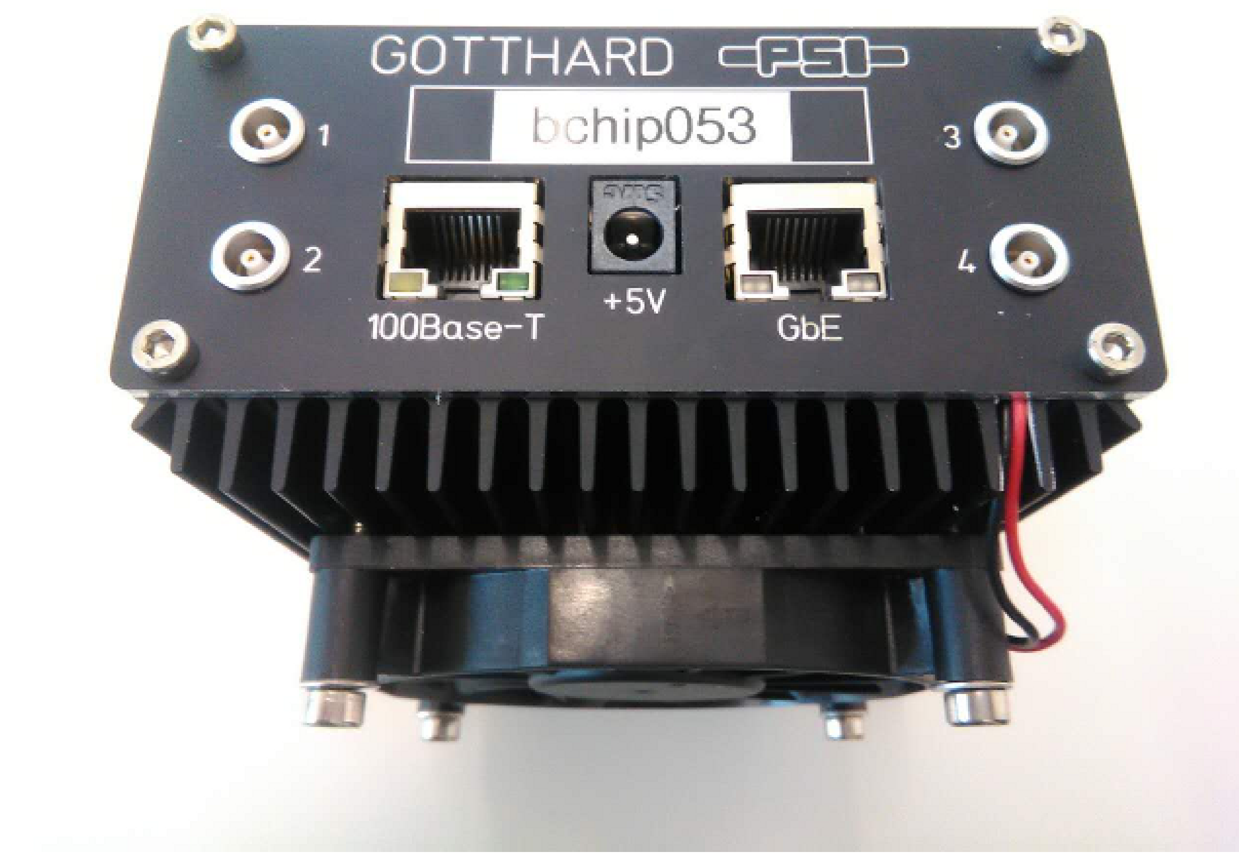
   Note that the bchip050 should be replaced by the hostname of the specific detector module.

DETECTOR SET-UP AND CONFIGURATION

The information about how to set-up the Gotthard-I detector and configure the detector has been summarized as below.

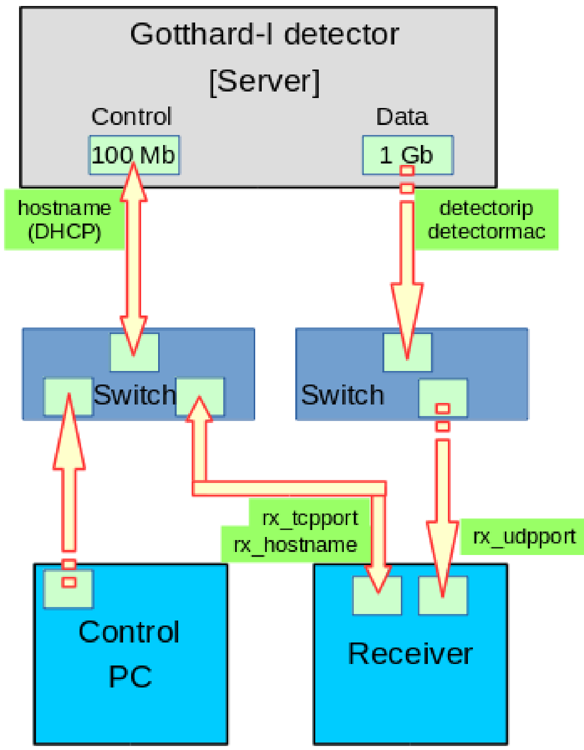
3.1 Connect the detector

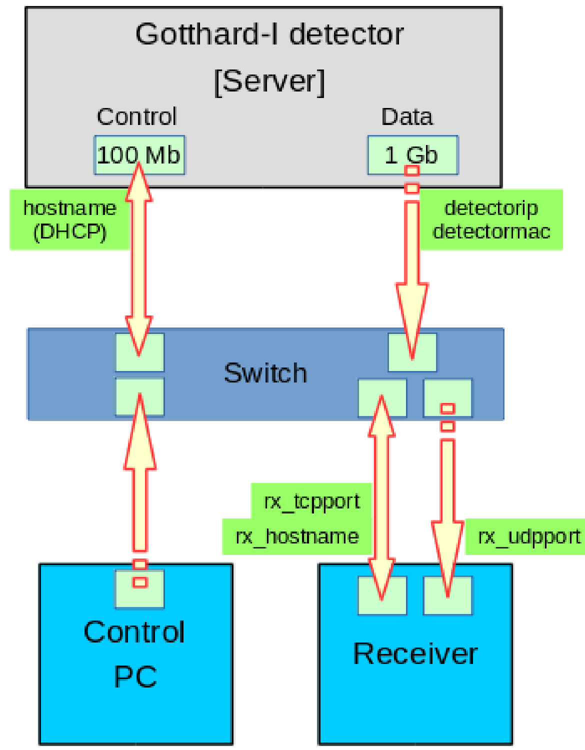
There are one power plug, two Ethernet ports, and four lemo connectors.



The suply power requires +5 V as input.

The two Ethernet ports: One for detector control, the other for data transmission. Two options of connection between the detector, control PC and receiver can be found below.





The command can be sent through a control PC to the detector directly or through a receiver(refer to the figures above) and the data received by the receiver through 1 GbEthernet link under UDP protocal.

The four lemo connectors (labeled 1-4): 1 is used to receive triggers for the detector, 2 is the trigger sent-out from the detector. The lemo connectors 2 always generates trigger signals by the detector in case the other devices need to synchronize with it. In order to trigger the detector by an external signal, refer to the section "Edit the configuration file". Connectors 3 and 4 are normally not in use.

For the input external trigger for connector 1, it should be 3.3 V LVTTL signal with ~100 ns pulse width.

3.2 Configure the system

1. Edit the configuration file

   The configuration file ends with an extension of ".config". In the whole text, the file name "bchip.config" is used.

```
type Gotthard+
0:hostname 10.42.0.35
#0:port 1952
#0:stopport 1953
#0:rx_tcpport 1954
0:settingsdir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
0:angdir 1.000000
0:moveflag 0.000000
0:lock 0
0:caldir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
```

```
11 0:ffdir /home/wp74diag
12 0:extsig:0 off
13 0:extsig:1 off
14 0:extsig:2 off
15 0:extsig:3 off
16 0:detectorip 10.42.0.2
17 0:detectormac 00:aa:bb:cc:dd:ee
18 0:rx_udpport 50004
19 0:rx_hostname 10.42.0.1
20 0:outdir /home/wp74diag/data/gotthard
21 0:vhighvoltage 120
22 0:frames 1000
23 0:exptime 0.0001
24 0:period  0.0100
25 master -1
26 sync none
27 outdir /home/wp74diag/data/gotthard
28 ffdir /home/wp74diag
29 headerbefore none
30 headerafter none
31 headerbeforepar none
32 headerafterpar none
33 badchannels none
34 angconv none
35 globaloff 0.000000
36 binsize 0.001000
37 threaded 1
```

The following line should be changed accordingly for each detector module or PC connection:

- L2: hostname or IP address for the detector
- L5: the communication port between client and receiver, 1954 by default
- L6 & L10: setting directory based on the location of slsDetectorsPackage folder
- L11, L20, L27 & L28: output directory of data
- L12: set to "trigger_in_rising_edge" if a trigger will be used. It is suggested to set it all the time. With this, it is also possible to work with "Auto" mode without triggers. The setting of different modes, e.g. "Auto" or "Trigger Exposure Series", can be done from the "Timing Mode" box inside "Measurement" tab of the SLS Detector GUI. The details for setting GUI can be found in the Section "Usage of GUI" of Chapter "Use GUI to perform measurement".
- L16: the ip address of the detector for the UPD interface with the receiver
- L17: the mac address of the detector upd interface to mac is configurable; any unique mac address can be set
- L18: the udp port of the receiver for data receiving
- L19: host name or IP address of the receiver for the TCP/IP interface with the client
- L21: bias voltage of the sensor; 200 V is recommended for operation

In addtion, the "rx_udpip" can also be set if several internet connections exist:

- rx_udpip: the ip address of the receiver for the UDP interface with the detector; it has to be on the same internet as L16

2. Power on the detector module first and start the detector server

   The detector server will automatically start for users. If not, type the following in the command line:

```
> ping bchip050
> telnet bchip050
> killall gotthardDetectorServer
root:>./gotthardDetectorServer
```

Note that “bchip050” is the name of specific module! Try to ping the module and see whether it has been connected and then start the server.

3. Start receiver

   To start the receiver on the PC, enter the “slsDetectorsPackage/bin” folder and type the following in the command line:

```
> which slsReceiver (if path is configured correctly, it's in the right bin folder)
> slsReceiver (Note: or ./slsReceiver)
```

   The libSlsDetector.so and libSlsReceiver.so project libraries, and project executatbles should be added before starting receiver server. See chapter section-2.2.

4. Start GUI

   To start GUI for detector control and data display, type the following in the command line:

```
> slsDetectorGui --f [.../.../]bchip.config
```

   The -f option is only needed if the detector and receiver are not configured.

   In the GUI pop-up, the “developer” tab will not be activated and thus the DAC values cannot be changed in the GUI (only in command line in this case). If a DAC value needs to be changed, the “developer” tab should be activated and the following command should be used:

```
> slsDetectorGui -df [.../.../]bchip.config
```

   In such case, the DAC values can be changed in-situ and the temperature of the FPGA can be readout. Only do this if DAC values need to be changed.

   Note that “export” has to be done to run any project executables.

3.3 Exit after measurements

1. Stop the receiver first:

```
> CTRL + C
```

2. Stop the detector server

   This automatically done when powering off the detector.

3.4 CLI mode

The detector can also be ran and controlled without using GUI. In this case, the make file needs to be modified. This is useful if the QWT is not available on the system. To compile without GUI using: “make client; “make receiver; make” in the command line.

Some useful executables in such mode:

```
> sls_detector_put   (Note: set a value of a parameter)
> sls_detector_get   (Note: get a value of a parameter)
> sls_detector_help  (Note: get help on something)
> sls_detector_acquire   (Note: acquire images)
```

The syntax of commands is:

```
> sls_detector_put [id]:command [argument]   (for using sls detector class)
> sls_detector_put [id]-command [arguments]  (for using multi-detector class)
```

Initialization commands:

```
* free          : frees the shared memory in the 1st detector's slot
   get            free -  frees shared memory
* hostname      : hostname
   put            hostname [name] - sets hostname
   get            hostname - gets hostname
* settingsdir   : the path to the settings folders
   put            settingsdir [fname] - sets path
   get            settingsdir - gets path
* caldir        : the path to the calibration folders
   put            caldir [fname] - sets path
   get            caldir - gets path
* outdir        : the path to the output files
   put            outdir [fname] - sets path
   get            outdir - gets path
* settings      : the settings for the detector.
  [Options]       [value]- lowgain, mediumgain, highgain, veryhighgain,
dynamicgain
   put            settings [value] - sets settings
   get            settings - gets settings
* config        : configuration files. fname:multidetector parameters.
[fname].det[id]:detector specific parameters(clientip, servermac etc.)
   put            config [fname] - reads configuration file and detector
specific file and sets the values from it
   get            config [fname] - writes configuration file. Detector specific
file is created automatically
```

Acquisition commands:

```
* extsig         : to use the trigger for acquisition.
  [Options]         [signal_id]- 0, 1, 2, 3
                    [value]- off, trigger_in_rising_edge, gate_in_active_high
   put              extsig:[signal_id] [value] - sets settings
   get              extsig:[signal_id] - gets settings
* frames         : number of frames
   put              frames [value] - sets number
   get              frames - gets number
* cycles         : number of trains
   put              cycles [value] - sets number
   get              cycles - gets number
* exptime        : exposure time in seconds
   put              exptime [value] - sets time
   get              exptime - gets time
* period         : acquisition period in seconds
   put              period [value] - sets time
   get              period - gets time
* delay          : delay after trigger in seconds
   put              delay [value] - sets time
   get              delay - gets time
* status         : acquisition status
  [Options]         [value]- start, stop
   put              status [value]- starts/stops acquisition
   get              status - gets acquisition status
* fname          : sets the output file name
   put              fname - the output file name. By default, it is
run_[index].raw
* index          : sets the start index of the output files
   put              index [value] - the frame number. By default, it is the next
index number
* frame          : writes the next frame to outdir(output files directory)
   get              frame - writes the frame to [fname]_[index].raw
* data           : writes all the frames to outdir(output files directory)
   get              data - writes all the frames to outdir [fname]_[index].raw
```

Debugging commands:

```
* reg            : read/write register
   put              reg [address_in_hex] [value_in_hex] - writes to register
   get              reg - reads register value
```

General commands:

```
* temp_adc      : adc temperature
   get             temp_adc - gets temperature
* temp_fpga     : fpga temperature
   get             temp_fpga - gets temperature
* darkimage     : loads dark image from file to the detector
   put             darkimage [fname] - loads image from file to detector.
* gainimage     : loads gain image from file to the detector
   put             gainimage [fname] - loads image from file to detector.
* resetctr      : stops acquisition, reset counter block memory
   put             resetctr [value] - if value=1, starts acquisition after
resetting counter block memory
* readctr       : stops acquisition, reads counter block memory to a file
   put             resetctr [value] [fname] - if value=1, starts acquisition
after reading counter block memory
```

Commands for configuring network (these are normally not used independently, since a configuration file is loaded):

```
* rx_hostname   : the ip/hostname of the receiver
   put             rx_hosname [ip/hostname] - sets receiver hostname or IP address

   get             rx_hosname - gets the receiver hostname
* rx_udpip      : the udp ip of receiver
   put             rx_udpip [ip] - sets receiver udp ip in xxx.xxx.xxx.xxx format
   get             rx_udpip - gets the receiver udp ip
* rx_udpmac     : the mac address of the receiver
   put             rx_udpmac [mac_address] - sets receiver mac address in
xx:xx:xx:xx:xx:xx format. Normally automatically retrieved from receiver using
rx_hostname.
   get             rx_udpmac - gets the receiver mac address
* detectormac   : the mac address of the detector
   put             detectormac [mac_address] - sets mac address of detector in
xx:xx:xx:xx:xx:xx format
   get             detectormac - gets the detector mac address
* detectorip    : the ip address of the detector
   put             detectorip [ip] - sets ip address of detector in
xxx.xxx.xxx.xxx format
   get             detectorip - gets the detector ip address
* rx_tcpport    : the tcp port of the receiver
   put             rx_tcpport [port] - sets receiver tcp port
   get             rx_tcpport - gets the receiver tcp port
* rx_udpport    : the udp port of the receiver
   put             rx_udpport [port] - sets receiver udp port
   get             rx_udpport - gets the receiver udp port
* configuremac  : configure mac
   put             configuremac [value] - configures mac; value=-1 for all adc
or 1..5 for a specific adc
```

Example of using commands:

```
./sls_detector_get free
./sls_detector_put hostname bchip001
./sls_detector_put settingsdir  ../settings
./sls_detector_put caldir  ../settings
./sls_detector_put outdir ~/scratch
./sls_detector_put settings highgain
./sls_detector_put extsig:0 off
./sls_detector_put extsig:1 off
./sls_detector_put extsig:2 off
./sls_detector_put extsig:3 off

./sls_detector_put frames 2
./sls_detector_put cycles 1
./sls_detector_put exptime 0.001
./sls_detector_put period 0.002
./sls_detector_put status start
./sls_detector_get status

./sls_detector_put index 0
./sls_detector_get frames
./sls_detector_get data

./sls_detector_get reg 0x50
./sls_detector_put reg 0x50 0x23

./sls_detector_get temp_adc
./sls_detector_get temp_fpga

./sls_detector_get config ~/scratch/trial.config
./sls_detector_put config ~/scratch/trial.config


./sls_detector_put 0:detectorip 129.129.202.46
./sls_detector_put 0:detectormac 00:aa:bb:cc:dd:ee
./sls_detector_put 0:rx_udpport 50004
./sls_detector_put 0:rx_udpip 129.129.202.120
./sls_detector_put 0:rx_hostname pc6898
./sls_detector_put 0:outdir ~/scratch


./sls_detector_put digibittest 1
./sls_detector_put configuremac 1

./sls_detector_put darkimage ~/scratch/image.txt
./sls_detector_put gainimage ~/scratch/image.txt
./sls_detector_put resetctr 1
./sls_detector_get readctr 0 ~/scratch/counter.raw
```

More useful commands can be found in https://www.psi.ch/detectors/UsersSupportEN/slsDetectorClientHowTo.pdf.

In this mode, the configuration file should be loaded manually at least once:

```
> sls_detector_put config [.../.../]bchip.config
```

3.5 Setup file

This is a set-up file for setting a specific measurement. It can be loaded from "Utilities->Load setup file" inside SLS Detector GUI. However, all these settings can be input and changed in the GUI.

```
fname run
index 0
dr 16
settings veryhighgain
exptime 0.000002990
period 0.000025
delay 0.999999968
gates 1
frames 3000000
cycles 1.000000000
timing auto
fineoff 0.000000
flatfield none
badchannels none
```

3.6 Configure two detectors

Here below is an example of configuration file for running two modules in parallel, which have been successfully tested with a laptop of EU.XFEL.

- Two-module configuration file:

```
detsizechan 2560 1
hostname 10.42.0.35+10.42.0.37+
#0:port 1952
#0:stopport 1953
#0:rx_tcpport 1956 must also have this in receiver config file
0:settingsdir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
0:angdir 1.000000
0:moveflag 0.000000
0:lock 0
0:caldir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
0:ffdir /home/wp74diag
0:extsig:0 off
0:extsig:1 off
0:extsig:2 off
0:extsig:3 off
0:detectorip 10.42.0.2
0:detectormac 00:aa:bb:cc:dd:ee
0:rx_udpport 50004
0:rx_udpip 10.42.0.1
0:rx_hostname 10.42.0.1
0:outdir /home/wp74diag/data/gotthard
0:vhighvoltage 120
```

```
0:frames 1000
0:exptime 0.0001
0:period  0.0100

#1:port 1952
#1:stopport 1953
#1:rx_tcpport 1956 must also have this in receiver config file
1:settingsdir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
1:angdir 1.000000
1:moveflag 0.000000
1:lock 0
1:caldir /home/wp74diag/slsDetectorsPackage/settingsdir/gotthard
1:ffdir /home/wp74diag
1:extsig:0 off
1:extsig:1 off
1:extsig:2 off
1:extsig:3 off
1:rx_tcpport 1720
1:detectorip 10.42.0.3
1:detectormac 01:aa:bb:cc:dd:e1
1:rx_udpport 50005
1:rx_udpip 10.42.0.1
1:rx_hostname 10.42.0.1
1:outdir /home/wp74diag/data/gotthard
1:vhighvoltage 120
1:frames 1000
1:exptime 0.0001
1:period  0.0100


master -1
sync none
outdir /home/wp74diag/data/gotthard
ffdir /home/wp74diag
headerbefore none
headerafter none
headerbeforepar none
headerafterpar none
badchannels none
angconv none
globaloff 0.000000
binsize 0.001000
threaded 1
```

CHAPTER FOUR

USE GUI TO PERFORM MEASUREMENT

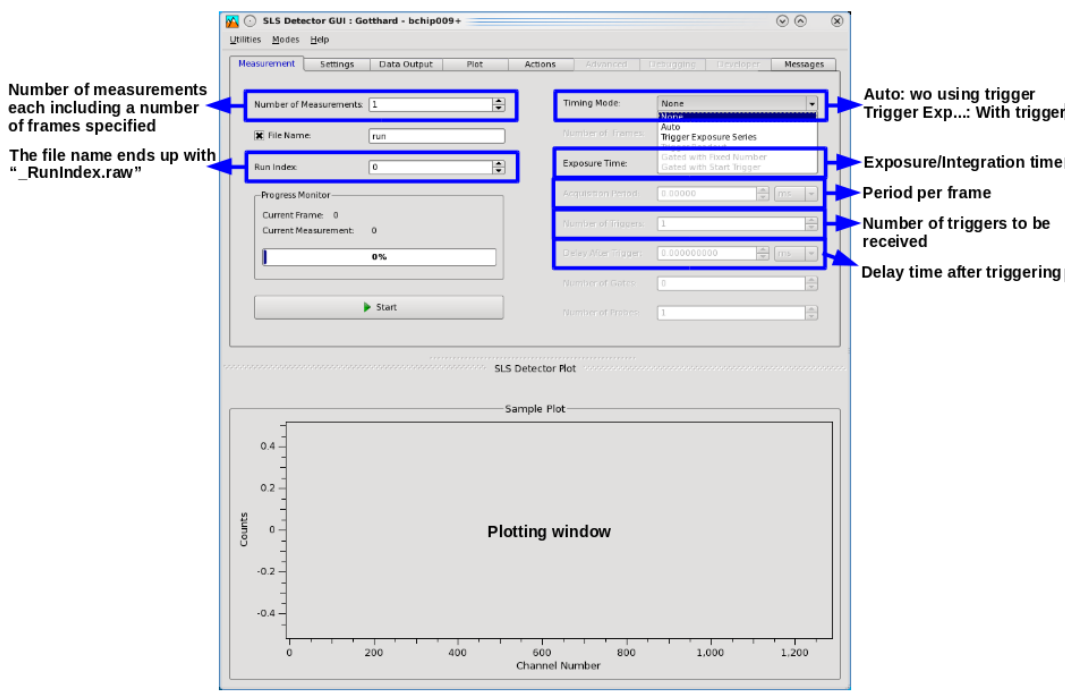
4.1 Usage of GUI

As introduced in previous chapter, the GUI can be started with the following in the command line:

```
> slsDetectorGui --f [.../.../]bchip.config
```

The GUI includes several tabs for detector control and data acquisition.

- The “Measurement” tab:



In this tab, it is possible to specify the following parameters:

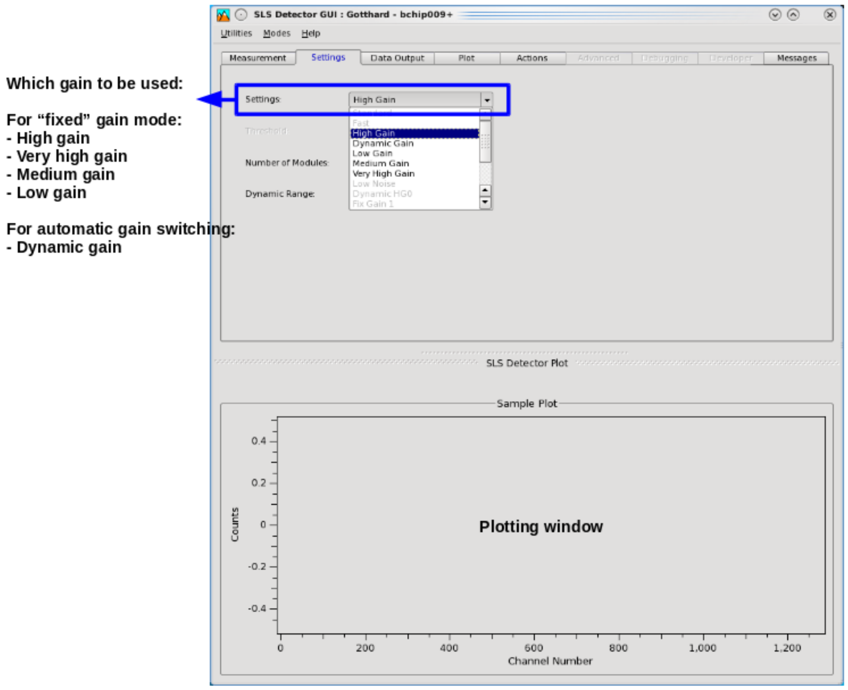
- Number of measurements: Each measurement includes a number of frames input on the right of the window
- Run index: The file name ends up with the index number, for example “run_f000000000000_RunIndex.raw”

- Number of frames: The total number of frames to be measured for each measurement
- Timing mode: "Auto" or "Trigger Exposure Series". The former does not trigger whereas the latter uses a trigger.
- Exposure time: Time of integration
- Acquisition period: Period per frame. The frame rate is given by 1 divided by the acquisition period, for example 1 ms input gives a frame rate of 1 kHz. In burst mode with trigger it refers to the period of two burst frames: it should be > 1.25 us (1/800 kHz); in continuous mode without trigger, it has to be larger than the exposure time at least and > 23-25 us (1/40 kHz); in continuous mode with trigger, it is suggested to be 23-25 us to make sure the triggers will not be overlooked in case the trigger rate is higher than the acquisition frequency (1/acquisition period).
- Number of triggers: Perform a number of frames for each trigger and total frames given by number of frames multiplying number of triggers for each measurement. For continous mode with trigger, the number of frames has to be set 1; for burst mode with trigger, the number of frames refer to the number of burst images for each trigger and maximal at 128 due to the limit of the memory in FPGA!
- Delay after trigger: Delay time to start taking data after receiving a trigger signal. For any numnber < 32 ns, the setting cannot work; thus a setting value > 32 ns is necessary for the delay of acquisition after receiving trigger.

Note that in order to use the trigger mode, the line "0:extsig:0 off" in the configuration file has to be changed to "0:extsig:0 trigger_in_rising_edge". The #1 of the lemo connectors is used to receive trigger signals. #2-4 are the trigger signals from the detector which can be used to synchronize the other devices when they need to be triggered by the detector, as explained in the previous chapter.

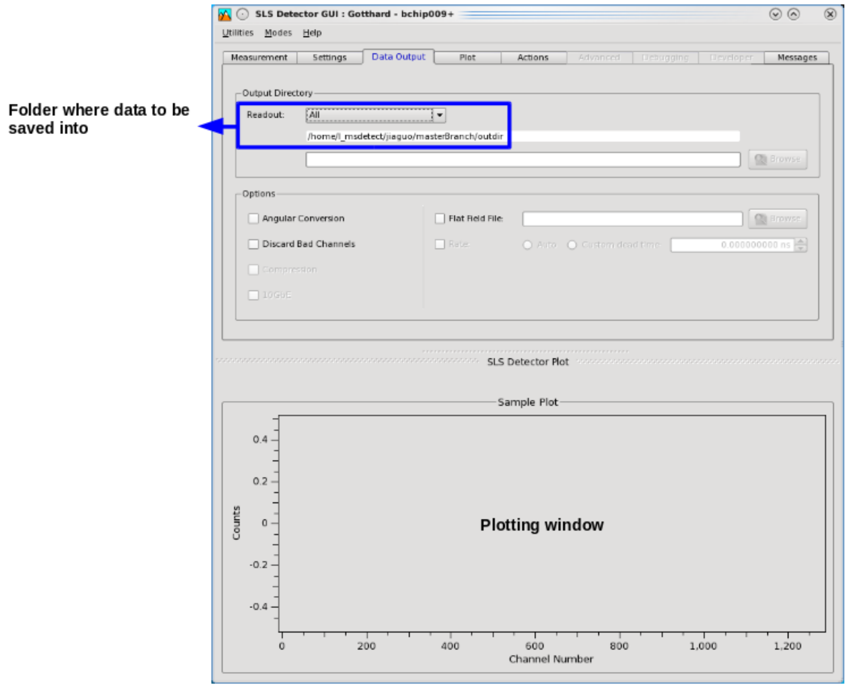
Only when the trigger mode is selected from the "Timing Mode" block, the input for "Number of triggers" and "Delay after trigger" area can be activated.

- The “Setting” tab:

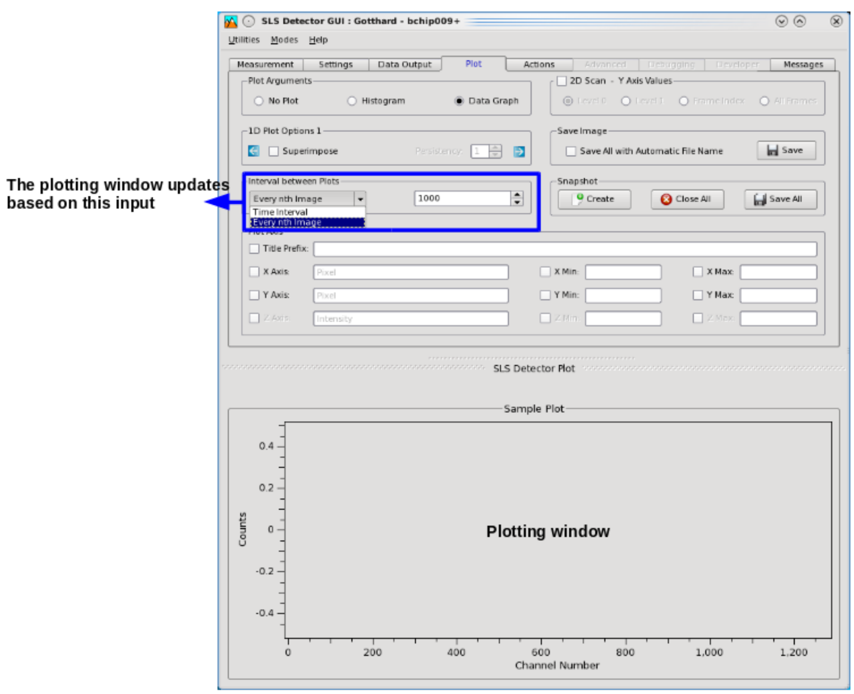


  - Settings: Select operating mode either “fixed” gain or dynamic gain switching
    - High gain: Single photons regime low noise, working up to a few tens of 12 keV photons
    - Very high gain: Single photons regime and very low noise, working up to a couple tens of 12 keV photons
    - Medium gain: No single photon sensitivity, working for photons between a few tens to hundreds
    - Low gain: No single photon sensitivity, working for photons from a few hundreds to ten thousand
    - Dynamic gain: Dynamically switch gain, working for single photon up to ten thousand photons

- The “Data Output” tab: Choose the folder for output data



- The “Plot” tab:



The refresh rate of the plot in “Plotting window” can be set here. In addtion, pedestal subtracted results can be shown in the "Plotting window" by perform an on-line pedestal subtration through "1D plot option 1" dialog.

- The other tabs:

  Since the other tabs are irrelevant for users' setting, they will not be discussed here.

4.2 Examples to set-up measurements

- Measurements without trigger:
  - Choose "Auto" in "Timing Mode" of "Measurement" tab
  - In "Settings", select an operation gain in measurement: "Very High Gain", "High Gain", "Medium Gain", "Low Gain", or "Dynamic Gain"
  - Enter the "Exposure Time" in "Measurement" tab: With "Very High Gain" and "High Gain" mode, the "Exposure Time" should not exceed a few tens of microsecond, otherwise the ADU saturates due to leakage current.
  - Enter the "Acquision Period" in "Measurement" tab: 100 microseciond or 1 milisecond and so on, depending on the required frame rate.
  - Set "Number of frames" in "Measurement" tab and "Number of Measurements": The total frames given by "Number of frames" multiplied by "Number of Measurements".
  - Set the "File Name" and "Run Index"
  - Press the "Start" button in "Measurement" tab to start taking data
- Measurements with trigger:
  - In the configuration file, change "0:extsig:0 off" to "0:extsig:0 trigger_in_rising_edge"; and then load the configuration file again from GUI: "Utilities" -> "Load Configuration File"
  - Choose "Trigger Exposure Series" in "Timing Mode" of "Measurement" tab
  - Input "Number of Triggers" to be received by the detector in the "Measurement" tab
  - Input the delays through "Delay After Trigger" in the "Measurement" tab: The measurement starts with this delay after receiving trigger signal
  - In "Settings", select an operation gain in measurement: "Very High Gain", "High Gain", "Medium Gain", "Low Gain" or "Dynamic Gain"
  - Enter the "Exposure Time" in "Measurement" tab: With "Very High Gain" and "High Gain" mode, the "Exposure Time" should not exceed a few tens of microsecond, otherwise the ADU saturates due to leakage current.
  - Enter the "Acquision Period" in "Measurement" tab: 100 microseciond or 1 milisecond and so on, depending on the required frame rate and running mode. For continuous mode with trigger, the setting is suggested to be 23-25 us and longer setting may overlook pulses with repetition rate higher than acquisition rate (1/acquisition period); however for burst mode, this input defines the period between two burst images and can be shorter than 23-25 us but has to be > 1.25 us (800 kHz maximum).
  - Set "Number of frames" in "Measurement" tab and "Number of Measurements": The total frames given by "Number of frames" multiplied by "Number of Measurements" and "Number of Triggers". For burst mode, it refers to the number of burst images per trigger and thus should not be exceeded 128; for continuous mode, it should be 1 (1 trigger gives 1 image).
  - Set the "File Name" and "Run Index"

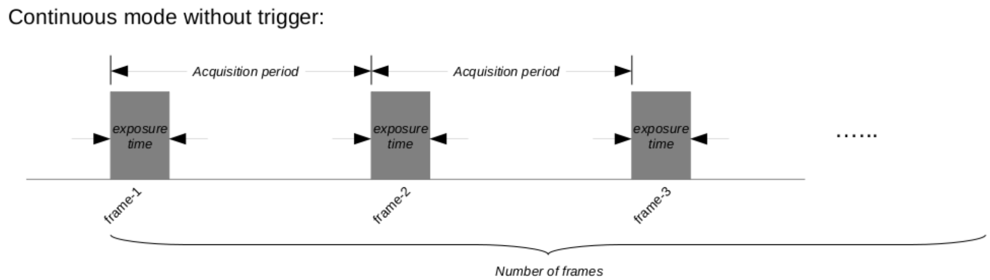
- Press the "Start" button in "Measurement" tab to start taking data

- Note that the dark measurement, X-ray fluresence measurement with lab X-ray source can be done with "Auto" mode; whereas measurements with the single shot laser and synchrotron beam/FEL should be done with "Trigger" mode.

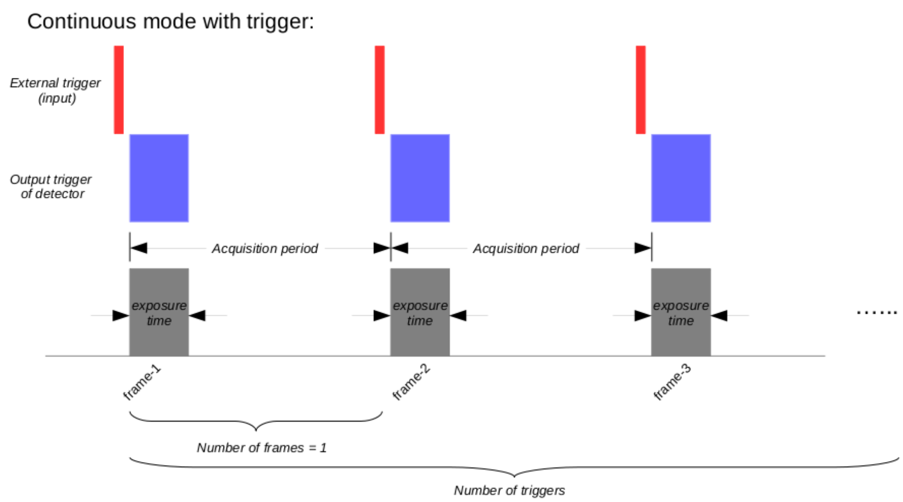
4.3 Examples to set-up timings

The explanation of setting up time related input can be summarized below.

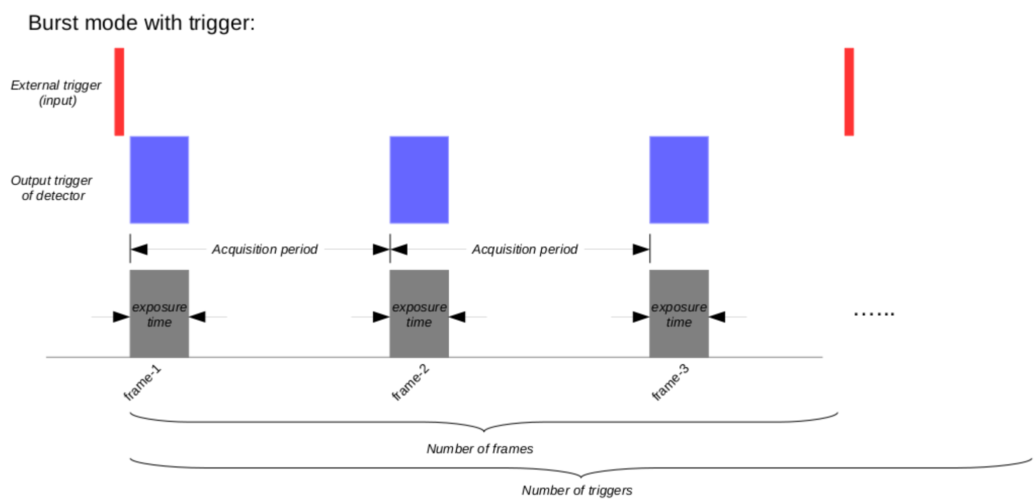
For continous mode without external trigger ("Auto" option in "Timing Mode" of "Measurement" tab):



For continous mode with external trigger ("Trigger Exposure Series" option in "Timing Mode" of "Measurement" tab):



For burst mode with external trigger ("Trigger Exposure Series" option in "Timing Mode" of "Measurement" tab):


Burst mode with trigger:
External trigger (input)
Output trigger of detector
Acquisition period
Acquisition period
exposure time
exposure time
exposure time
……
frame-1
frame-2
frame-3
Number of frames
Number of triggers

CHAPTER

FIVE

CHARACTERIZATION AND CALIBRATION

To get correct photon energy from measurements with a charge-integrating detector (Gotthard), proper characterization and calibration is necessary. This chapter will introduce the basic concept of detector calibration.

Usually, the following need to be characterized/calibrated:

- Gains and offsets in "fixed" gain mode (HG0, G0, G1 and G2)
- Gains and offsets in dynamic gain mode (G0, G1 and G2)
- Noise

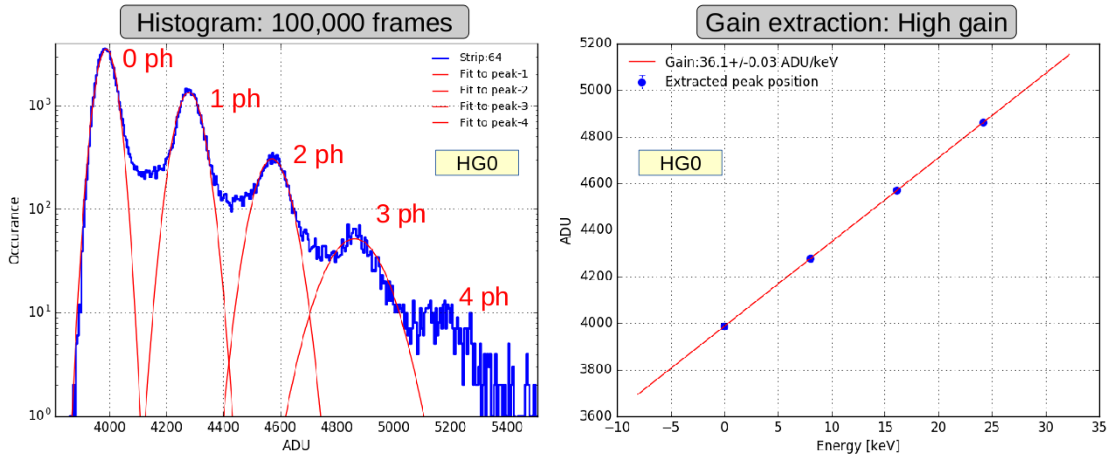
The conversion of measured ADU to photon energy in a measurement will be based on the calibration results mentioned above.

5.1 Gains & offsets in "fixed" gain mode

The gain and offset (also called "pedestal" sometimes) for very high gain (HG0) and high gain (G0) can be measured with X-ray fluorescence from an X-ray tube. However the gains for medium gain (G1) and low gain (G2) can only be measured with synchrotron/FEL beam instead of a lab X-ray source.

For example, in case a lab X-ray tube is used, the X-ray fluorescence from a Cu, Mo or other targets can be measured using Gotthard detector by putting it in front of the target. For this measurement, an exposure time (also called "integration time" sometimes) of a few microsecond, an acquisition period of 1 ms and "fixed" gain with either high gain (G0) or very high gain (HG0) shall be set. 2 us, 5 us and 10 us are commonly used as exposure time and >500 000 frames recommended to obtain enough data.

After the measurement, the histogram/occurance of ADU values for each strip can be plotted and peak positions can be extracted. As seen below, it is the histogram from a strip (Strip-64) in ameasurement using X-ray fluorescence from a Cu-target (Ka line at 8.05 keV). The 0, 1, ..., up to 4 photon peaks can be seen and their peak positions extracted and plotted as function of energy from different number of coincident photons.



The straight line fit gives the slope (gain in a unit of ADU/keV) and offset (in a unit of ADU). The gains for HG0 and G0 are different.

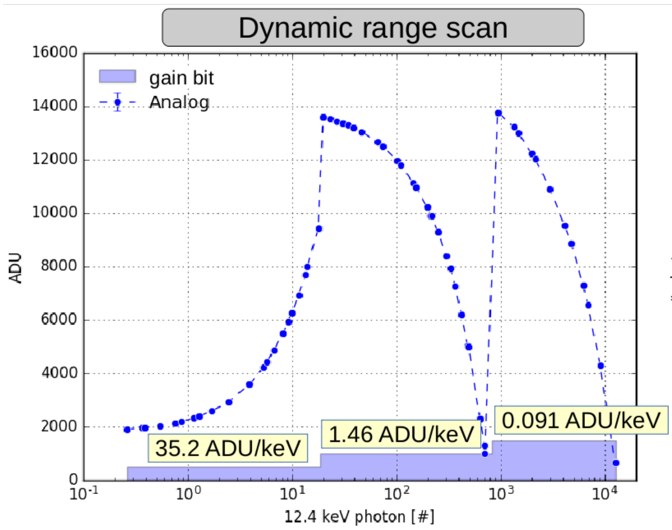
Since the medium gain and low gain are very small, it is not possible to get separated peak in the histogram using X-ray fluorescence. In this case, multiple coincident photons from synchrotron/FEL beam should be used to calibrate G1 and G2.

The offsets (pedestals) for HG0, G0, G1 and G2 can be obtained from measurements using the same settings but without any X-rays. The mean or the center of a gaussian fit to the hitogram represents the offset (pedestal) for the specific gain setting used in the measurement.

5.2 Gains & offsets in dynamic gain mode

In dynamic gain mode, the high gain (G0) is used as the initial gain stage. The gain [ADU/keV] and offset [ADU] of high gain stage in dynamic gain mode are identical to the ones in "fixed" gain mode. That is, with X-ray fluorescence, the histogram for dynamic range mode and "fixed" gain mode using "high gain" are the same; however, the gains and offsets of medium and low gains are different between dynamic gain mode and "fixed" gain mode. Thus, it is necessary to calibrate the medium gain and low gain in dynamic range mode independent of the calibration of medium gain and low gain in "fixed" gain mode. Similarly, these can be measured with either strong X-ray source (synchrotron or FEL) or laser.

For lab tests using a laser, one can select the dynamic gain in the setting. By scanning the laser intensity, it is possible to obtain the dynamic range curve of a strip into which laser injects. The laser intensity can be converted to number of photons or keV based on a conversion rate between the slope in high gain region and the high gain (G0) measured with X-ray fluorescence. The dynamic range curve from laser measurement is shown below:



Based on the fore-mentioned conversion, the medium and low gain region can be fit by staight lines separated and then gains and offsets extracted as indicated in the figure.

Note that all numbers indicate in the figures are from a prototype instead of a detector module and thus it can be different from the results obtained with a detector module.

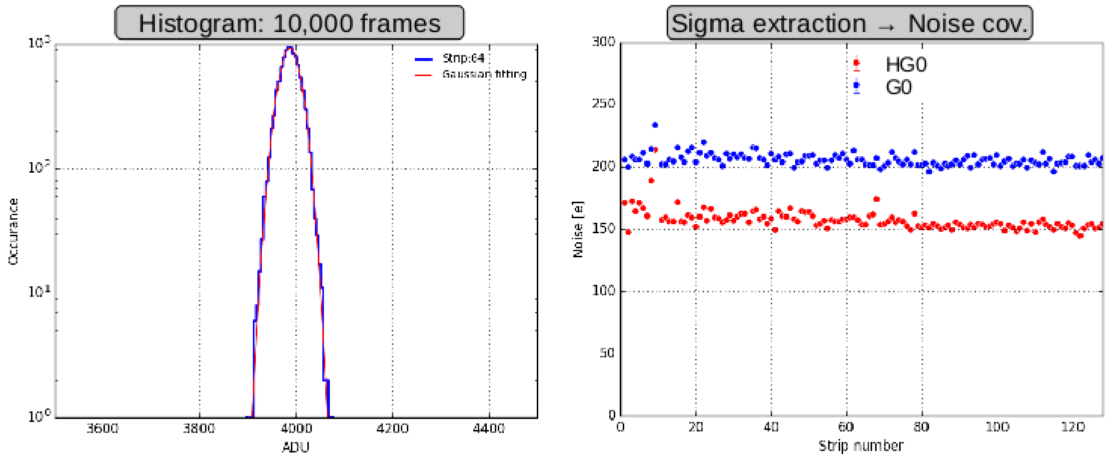
5.3 Noise

Noise is a key factor indicating the best separation of two different photon energies in a measurement. For example, for a noise of 300 e- (corresponding to 1 keV), a good energy separation can be achieved for 5 keV when counting 5 sigma. The noise is related to the exposure time and temperature. For a "fixed" experimental condition where exposure time and temperature do not change, the noise can also be measured through "dark" measurement: Operating the detector in a light-tight environment without X-rays. The settings for exposure time and acquisition period can be identical to the ones in X-ray measurement but not mandatory. The number of frames can be less, for example ~ 10 000 frames in total.

After the measurement, the histogram/occurance of ADU values for each strip shall be calculated. The distribution of histogram is fited by a gaussian function with mean value the offset (pedestal) as mentioned before and the sigma (unit: ADU) the noise related parameter. Then the noise can be calculated based on the following formula:

> Noise[E.N.C.]=Sigmal[ADU]/gain[ADU/keV]*1000/3.6[eV]

Here below is an example of noise measurement with very high gain and high gain:



It is recommended to perform this measurement with the same settings used for an experiment.

5.4 Energy conversion

Once the gains and offsets calibrated, the conversion can be done with:

photon_energy=(Analog[ADU]-offset[ADU])/gain[ADU/keV]

using offset (pedestal) and gain values for specific gain stages.

CHAPTER

SIX

DATA PROCESSING

For data processing, a few routines and functions have been prepared as a starting point.

For data analysis with provided routines, the following are essential:

- python 2.7 or 3.3
- numpy
- scipy
- matplotlib
- peakutils
- h5py
- lmfit

For python related routines, one solution is to install ANACONDA: https://www.continuum.io/downloads. It includes all necessary python-related packages for scientific calculation except lmfit module. For Anaconda python users, the lmfit module can be installed through:

```
> conda install -c conda-forge lmfit
> conda install -c newville lmfit
```

Standard installation from XFEL.EU calibration package is also enough.

To show the examples in this text, Jupyter Notebook is used. It is also fine to run the code in python script (".py" file).

6.1 Data structure

The data structure for each frame can be summarize as below, each data point is 16-bit:

> | a | b | c | c | data for 1280/2-1 channles | a+1 | b | data for 1280/2+1 channels |

Here "a" refers to index number, "b" time related number, "c" flag. The data for 1280 channels/strips will divide into two parts, thus it is necessary to make sure the continuity of index number for per 1280 channels in order to avoid data misalignment due to data packets loss.

The 16-bit data for each channel include both the gain bit and analog information: The first 2 bits give the gain stage used and the last 14 bits analog ADU value. For the first 2 bits, "00" for high (G0) or very high gain (HG0), "01" for medium gain (G1) and "11" for low gain (G2).

The gain bit information is particularly important when using dynamic gain mode.

6.2 Conversion gain

The gain for very high gain (HG0) and high gain (G0) can be calculated based on X-ray fluorescence data with the function: *calGain_Xray(Folder, Run_index, binsize=5, E_xray=8.05, half_region=10, thres=0.2, min_dist=40, channels=linspace(1,1280,1280), common_correction=”No”)*. The input parameters are:

- Folder: where the data file located
- Run_index: the input index number in GUI. This number is in front of ”.raw” in the name of measurement files.
- binsize: the bin size to generate the histogram
- E_xray: energy of the X-ray fluorescence
- half_region: half of the fitting region per peak
- thres: threshold to consider as a peak in its histogram
- min_dist: minimal distance between two peaks
- channels: input channels to be calcuated
- common_correction: whether a common mode correction is mode or not

It calls function *index_peaks()* using “thres” and “min_dist” as input defined in PeakUtils module. More information about this module can be found at http://pythonhosted.org/PeakUtils/.

Below is an example about how to use the function *calGain_Xray()*. Download the example at https://desycloud.desy.de/index.php/s/E2n1uRYXogabLhA.

jupyter gainExtraction_Xray Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Markdown CellToolbar

Use functions in "func_GotthardI.py" to calculate gain

Created on 2016-07-22

Changes:

```
- 2016-07-22: First creation
```

by Jiaguo Zhang, jiaguo.zhang@psi.ch

Import packages and modules

In [1]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import *
from func_GotthardI import *
```

Give folder of data files and Run_index

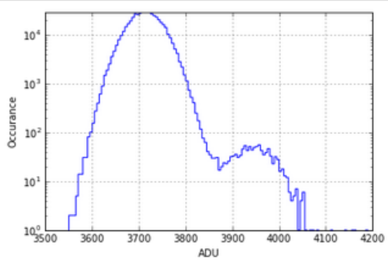
In [2]:
```
Folder = "/scratch/Data/20160720/"
Run_index = 6
```

Take a look at the histogram for a specific channel/strip

In [3]:
```
bins, occurance = getHist(Folder, Run_index, i_strip=500, binsize=5)
```

Take a look at the histogram

In [4]:
```
plot(bins, occurance, drawstyle="steps-pre", linewidth=1.0)
grid(True)
xlim(3500,4200)
ylim(1,)
yscale("log")
xlabel("ADU")
ylabel("Occurance")
show()
```



Calculate the gain for this specific channel/strip

Call the function: getGain_Xray(bins, occurance, E_xray=8.05, half_region=10, thres=0.2, min_dist=40)

In [5]:
```
gain_i_strip, offset_i_strip, gain_error_i_strip, peak_pos_i_strip, E_peaks = getGain_Xray(bins, occurance, E_xra
print "The gain for this input strip:", gain_i_strip, " ADU/keV."
```

```
The gain for this input strip: 28.8660451111  ADU/keV.
```

Calculate conversion gain for all strip and save them in a file

Call the function: with certain guess of input parameters For Cu-target: calGain_Xray(Folder, Run_index, binsize=5, E_xray=8.05, half_region=10, thres=0.2, min_dist=40) For In-target: calGain_Xray(Folder, Run_index, binsize=10, E_xray=24.2, half_region=10, thres=0.01, min_dist=140)

In [6]:
```
# Calculate gain for all strips
gain, offset, gain_error = calGain_Xray(Folder, Run_index, binsize=5, E_xray=8.05, half_region=10, thres=0.01, m:
savetxt("gain_In_vhg.txt",gain)
savetxt("offset_In_vhg.txt", offset)
```

```
Channel: 128 , gain: 29.7 ADU/keV...
Channel: 256 , gain: 28.4 ADU/keV...
Channel: 384 , gain: 27.9 ADU/keV...
Channel: 512 , gain: 27.6 ADU/keV...
No proper value find by peakutils! A dead channel?
Channel: 640 , gain: 29.8 ADU/keV...
Channel: 768 , gain: 29.6 ADU/keV...
Channel: 896 , gain: 30.0 ADU/keV...
Channel: 1024 , gain: 29.5 ADU/keV...
Channel: 1152 , gain: 29.9 ADU/keV...
Channel: 1280 , gain: 27.7 ADU/keV...
```

Another function can also be called for gain calculation using lmfit module with a method of multi-peak fitting: *calGain_Xray_lmfit(Folder, Run_index, binsize=5, Exray=8.05, gain_guess=10.0, sigma_guess=15, prob_1ph=0.4, channels=linspace(1,1280,1280), common_correction="No")*. The input parameters are:

- Folder: where the data file located
- Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.
- binsize: the bin size to generate the histogram
- E_xray: energy of the X-ray fluorescence
- gain_guess: initial guess of gain value in terms of ADU/keV
- sigma_guess: initial guess of sigma value of gaussian fitting to noise and single photon peak
- prob_1ph: initial guess of single photon probability per channel per frame
- channels: input channels to be calcuated
- common_correction: whether a common correction is mode or not

It is recommended to run both functions separately and merge the gain data, especially for the channels with failed fitting. If the convergence is not good enough, the program should run a few times till satisfication reached. The example for gain data merging can be downloaded at https://desycloud.desy.de/index.php/s/vBIZ2hZqkKHVksP.

6.3 Pedestal and noise

The pedestal (offset) and noise can be calculated for dark measurement in a light-tight box with the function: *calNoise_ADU(Folder, Run_index, binsize=5, common_correction="No", nbits=14)*. The input parameters are:

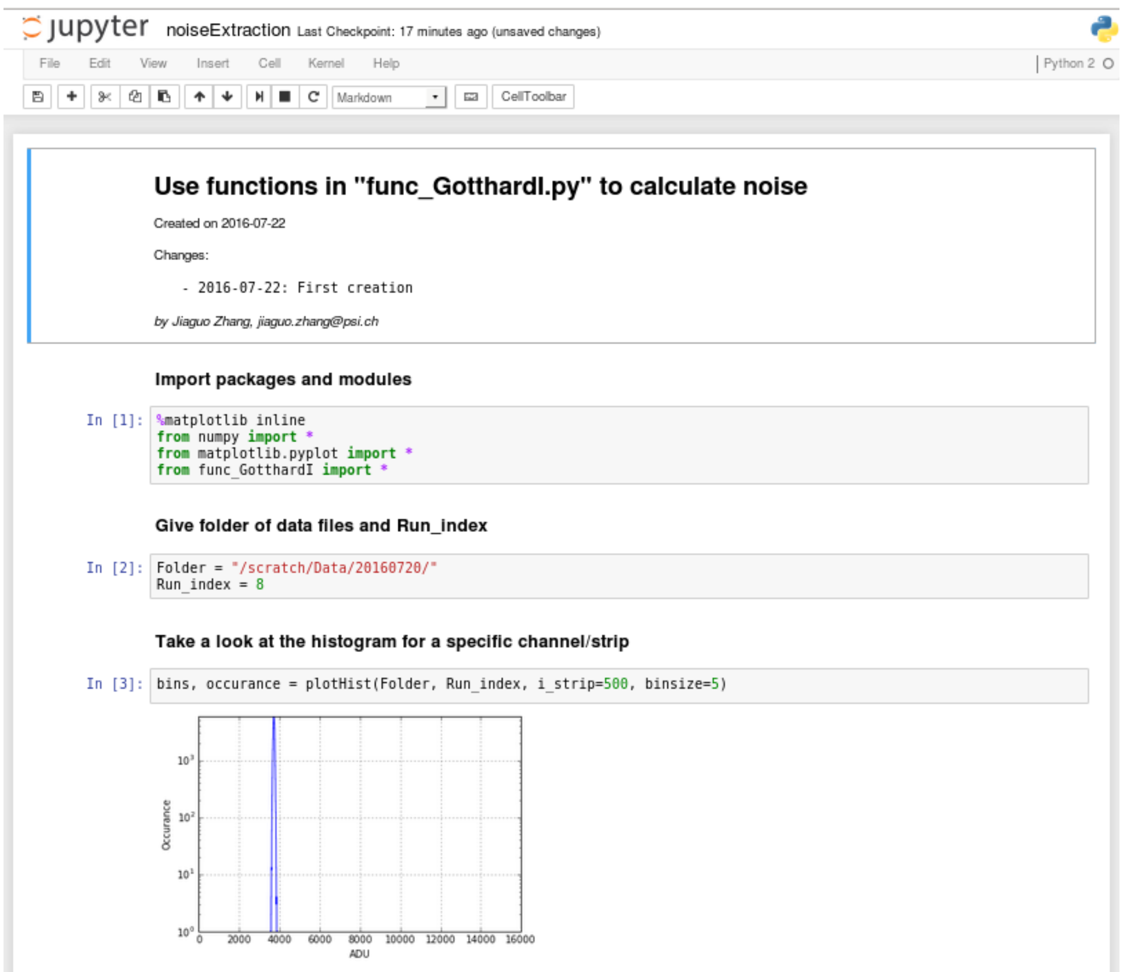
- Folder: where the data file located
- Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.

- binsize: the bin size to fill in the histogram
- common_correction: whether a common correction is mode or not
- nbits: the number of bits for analogue ADU value. Use 14 for the consistent gain in a measurement and 16 for a changed gain in a measurement.

The calculated noise in terms of ADU can be converted to equivalent noise charge (E.N.C.) with the function *convert-Noise(noise_ADU, gain)*. The input:

- noise_ADU: the output from calNoise_ADU() function
- gain: the output from calGain_Xray() function

Below is an example about how to use the function *calNoise_ADU()* and *convertNoise()*. Download the example at https://desycloud.desy.de/index.php/s/T2XCi9orhNv1MDl.



Calculate the noise in a unit of ADU for this specific channel/strip

Call the function: getNoise_ADU(bins, occurance)

In [4]:
```
sigma, pedestal, sigma_error, pedestal_error = getNoise_ADU(bins, occurance)
print "The noise in ADU:", sigma
```

```
The noise in ADU: 33.512242131
```

Calculate noise in ADU for all strip and save them in a file

Call the function: calNoise_ADU(Folder, Run_index, binsize=5)

In [ ]:
```
sigma, pedestal, sigma_error, pedestal_error = calNoise_ADU(Folder, Run_index, binsize=5)
savetxt("noise_ADU.txt", sigma)
savetxt("pedestal.txt", pedestal)
```

Convert the noise in ADU to electrons and save it in a file

Call the function: convertNoise(noise_ADU, gain)

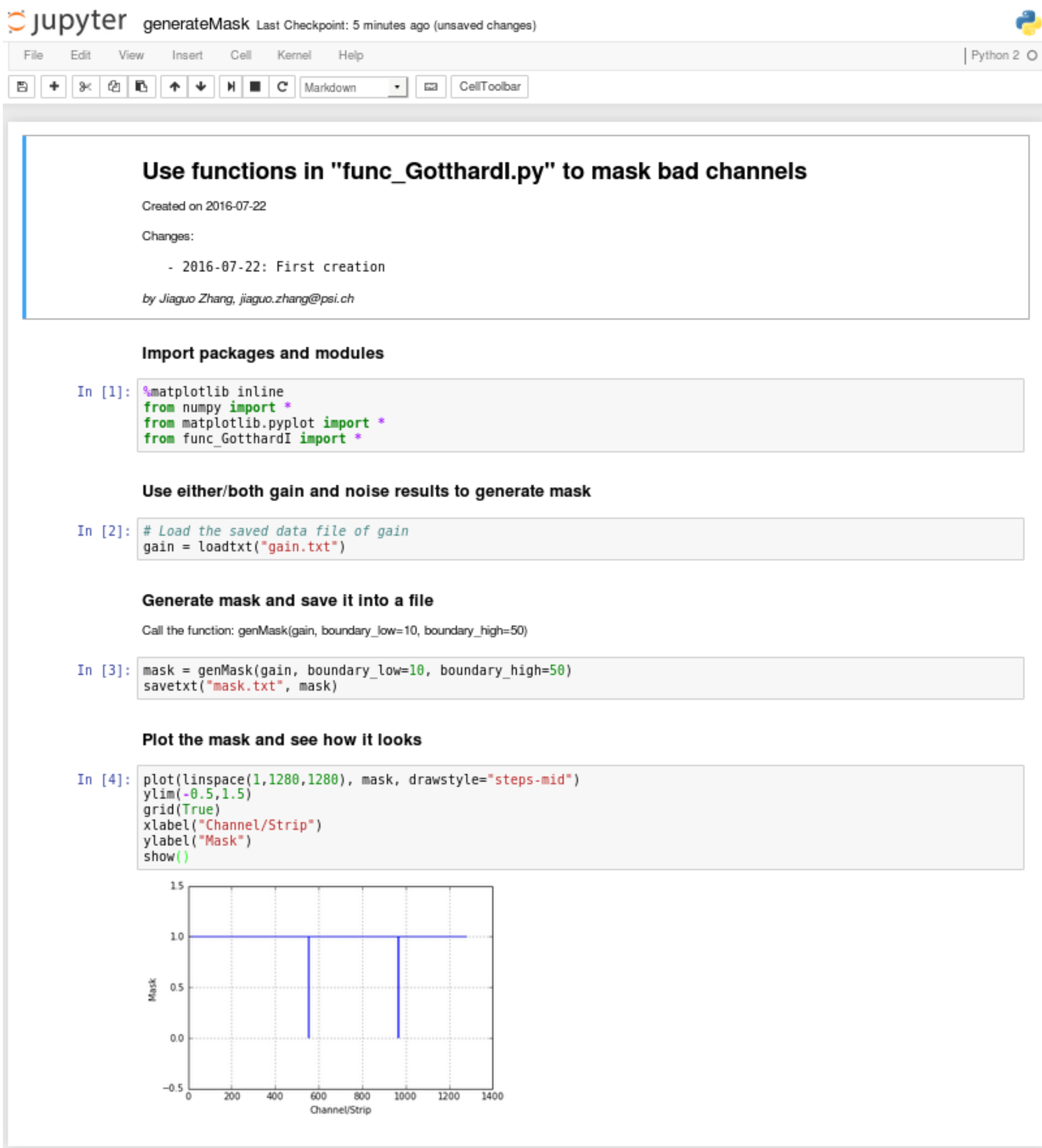
In [ ]:
```
# Load the saved gain and noise in terms of ADU
gain = loadtxt("gain.txt")
noise_ADU = loadtxt("noise_ADU.txt")
pedestal = loadtxt("pedestal.txt")

noise_e = convertNoise(noise_ADU, gain)
savetxt("noise_e.txt", noise_e)
```

6.4 Mask generation

For dead channels and noisy channels, it is possible to generate a mask which can be used to mask out the bad data when converting measurement to photon energy. The function for generating mask: *genMask(data, boundary_low, boundary_high)*, with

- data: either gain result or noise result
- boundary_low: channel with data below this boundary to be considered as a bad channel to be masked
- boundary_high: channel with data above this boundary to be considered as a bad channel to be masked

Below is an example about how to use the function *genMask()*. Download the example at https://desycloud.desy.de/index.php/s/gVJQ49QinjpXf0Q.



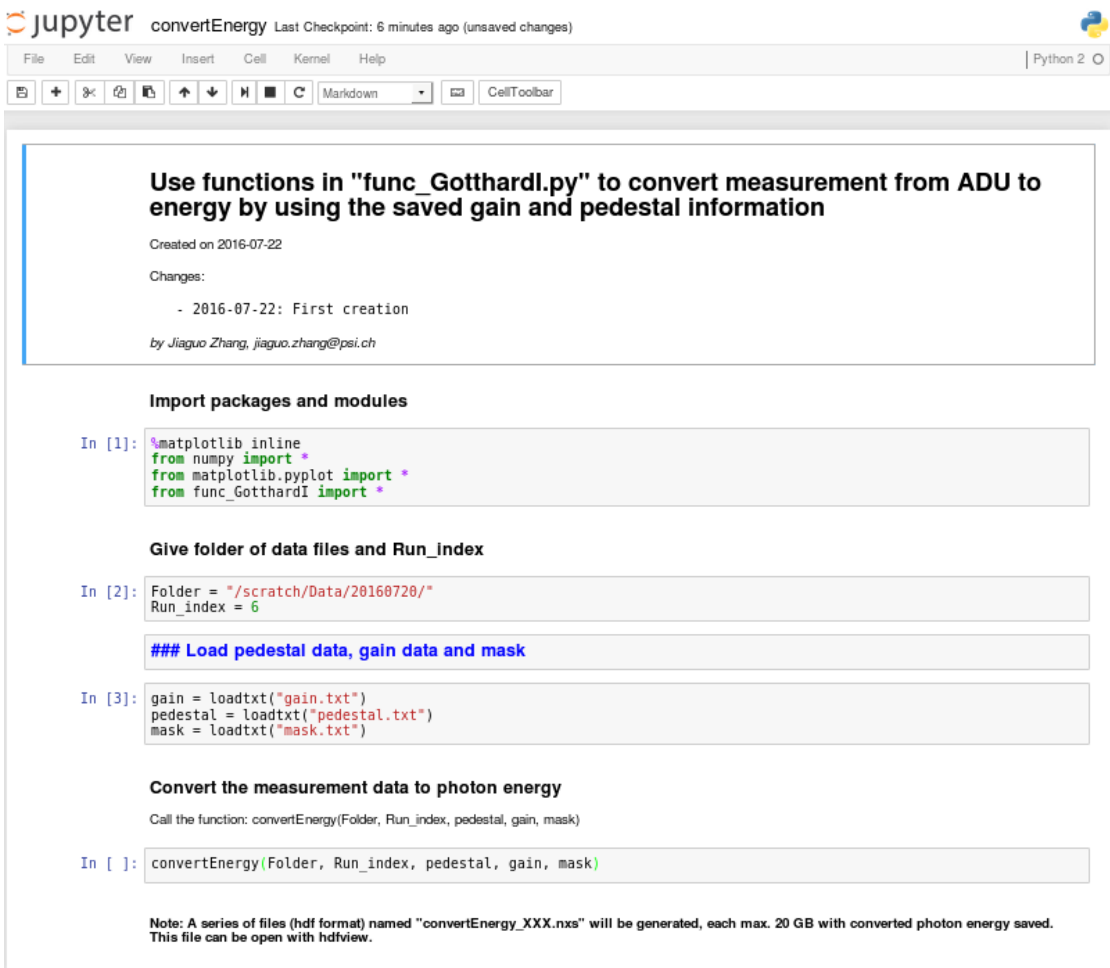
6.5 Energy conversion

Measurement data can be converted to photon energy [keV] using the pedestal data and gain data. The conversion is done with the function: *convertEnergy(Folder, Run_index, pedestal, gain, mask)*, with

- Folder: where the data file located
- Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.
- pedestal: pedestal data for all strips
- gain: conversion gain data for all strips
- mask: mask to be used to discard bad channels

By applying this function, a series of files in hdf5 format named "convertEnergy_XXX.nxs" will be generated, each max.20 GB with converted photon energy saved. The saved file with photon energy can be open with hdfview.

Below is an example about how to use the function *convertEnergy()*. Download the example at https://desycloud.desy.de/index.php/s/hss5kkxhtHN9JO8.



6.6 Callables

Here below will list the functions written in python, which can be called directly after importing the "func_GotthardI.py" module.

- *getHist(Folder, Run_index, i_strip, binsize, common_correction, nbits)*
  - This function will return the histogram for a specific strip.
  - Input:

    * Folder: where the data file located
    * Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files
    * i_strip: get the histogram for which strip
    * binsize: bin size in terms of ADU to get the histogram
    * common_correction: whether a common mode correction is preferred
    * nbits: number of bits for measurement data
  - Return:
    * bins
    * occurance

- *plotHist(Folder, Run_index, i_strip, binsize, common_correction, nbits)*
  - This function will run getHist first and plot the histogram
  - Input:
    * Folder: where the data file located
    * Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files
    * i_strip: get the histogram for which strip
    * binsize: bin size in terms of ADU to get the histogram
    * common_correction: whether a common correction is preferred
    * nbits: number of bits for measurement data
  - Return:
    * bins
    * occurance

- *getGain_Xray(bins, occurance, E_xray, half_region, thres, min_dist)*
  - This function will calculate the gain according to the input histogram and energy of X-ray
  - Input:
    * bins: the bins from output of getHist() or plotHist()
    * occurance: the occurance for each bin from the output of getHist() or plotHist()
    * E_xray: energy of X-ray characteristic line
    * half_region: half of peak region to be fit with Gaussian
    * thres: the threshold counts as a peak
    * min_dist: minimal distance between two peaks
  - Return:
    * gain: calculated gain from the input histogram
    * offset: offset/pedestal of the histogram
    * gain_error: error of calculated gain
    * peak_pos: the peak positions

        - E_peaks: the corresponding peak energies

- *getGain_Xray_lmfit(bins, occurance, Exray, gain_guess, sigma_guess, prob_1ph)*
    - This function will calculate the gain according to the input histogram and energy of X-ray using lmfit module
    - Input:
        - bins: the bins from output of getHist() or plotHist()
        - occurance: the occurance for each bin from the output of getHist() or plotHist()
        - Exray: energy of X-ray characteristic line
        - gain_guess: initial guess of conversion gain
        - sigma_guess: initial guess of sigma for a gaussian fitting
        - prob_1ph: probability of seeing 1 photon per channel per frame
    - Return:
        - gain: calculated gain from the input histogram
        - offset: offset/pedestal of the histogram
        - gain_error: error of calculated gain, 0 given at the moment
        - peak_pos: the peak positions
        - E_peaks: the corresponding peak energies
- *calGain_Xray(Folder, Run_index, binsize, E_xray, half_region, thres, min_dist, common_correction)*
    - This function will calculate the gains for all strips/channels
    - Input:
        - Folder: where the data file located
        - Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.
        - binsize: the bin size to generate the histogram
        - E_xray: energy of the X-ray fluorescence
        - half_region: half of the fitting region per peak
        - thres: threshold to consider as a peak in its histogram
        - min_dist: minimal distance between two peaks
        - common_correction: whether a common mode correction is preferred
    - Return:
        - gain: calculated gain for all strips
        - offset: offset/pedestal for all strips
        - gain_error: error of gain for each strip
- *calGain_Xray_lmfit(Folder, Run_index, binsize, Exray, gain_guess, sigma_guess, prob_1ph, common_correction)*
    - This function will calculate the gains for all strips/channels based on lmfit module
    - Input:

- Folder: where the data file located
- Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.
- binsize: the bin size to generate the histogram
- Exray: energy of X-ray characteristic line
- gain_guess: initial guess of conversion gain
- sigma_guess: initial guess of sigma for a gaussian fitting
- prob_1ph: probability of seeing 1 photon per channel per frame
- common_correction: whether a common mode correction is preferred

– Return:

  - gain: calculated gain for all strips
  - offset: offset/pedestal for all strips
  - gain_error: error of gain for each strip

- *getNoise_ADU(bins, occurance)*
  - This function will calculate the noise in terms of ADU for a specific input histogram
  - Input:
    - bins: the bins from output of getHist() or plotHist()
    - occurance: the occurance for each bin from the output of getHist() or plotHist()
  - Return:
    - sigma: the noise in terms of ADU
    - pedestal: offset/pedestal from the noise measurement
    - sigma_error: error of noise in terms of ADU
    - pedestal_error: error of offset/pedestal
- *calNoise_ADU(Folder, Run_index, binsize, common_correction, nbits)*
  - This function will calculate the noise in ADU for all strips
  - Input:
    - Folder: where the data file located
    - Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files
    - binsize: bin size in terms of ADU to get the histogram
    - common_correction: whether a common mode correction is preferred
    - nbits: number of bits for measurement data
  - Return:
    - sigma: the noise in terms of ADU
    - pedestal: offset/pedestal from the noise measurement
    - sigma_error: error of noise in terms of ADU
    - pedestal_error: error of offset/pedestal

- *convertNoise(noise_ADU, gain)*
  - This function will convert the noise in ADU to equivalent electron charge (E.N.C.)
  - Input:
    * noise_ADU: the calculated noise in [ADU]
    * gain: the calculated gain in [ADU/keV]
  - Return:
    * noise_e: noise E.N.C. [e-]
- *genMask(data, boundary_low, boundary_high)*
  - This function will generate a mask according to input data and boundaries
  - Input:
    * data: either gain or noise
    * boundary_low: the low boundary for reliable data
    * boundary_high: the high boundary for reliable data
  - Return:
    * mask: the mask for strips. 1 -> good strip/channel, 0 -> masked strip/channel
- *convertEnergy(Folder, Run_index, pedestal, gain, mask)*
  - This function will convert the measurement into photon energy and save the data into a file with hdf format
  - Input:
    * Folder: where the data file located
    * Run_index: the input index number in GUI. This number is in front of ".raw" in the name of measurement files.
    * pedestal: pedestal data for all strips
    * gain: conversion gain data for all strips
    * mask: mask to be used to discard bad channels
  - Return:
    * No data return but all saved into a hdf file automatically
- *file_merger_const(Path_open, File_base_in, index_i, N_files, Path_save)*
  - This function will merge different measurement files into one
  - Input:
    * Path_open: where the data files located
    * File_base_in: the file base in front of the index and the file extension
    * index_i: the index of the first file
    * N_files: number files following an increment of the index
    * Path_save: the path or folder to save the merged data
  - Return:
    * No data return but a merged file saved automatically into the specified folder

6.7 Last words

All routines and functions provided are only served as a basis for understanding the data process and analysis. The choise of programming language and software is up to the users.

CHAPTER SEVEN

ROUTINES

7.1 Python routines

There is a python module called "func_GotthardI.py", in which the basic functions are defined and can be called by importing this module. The file can be downloaded at https://desycloud.desy.de/index.php/s/VbhSuBmlIM18GHx.

```
"""
# A collection of functions for Gotthard-I module
Changes:
    - 2016-11-02 Function file_merger_const() added to merge binary files together
    - 2016-10-18 Function getHist() does not separate the gain bit and analog info any more, both wi
    - 2016-10-18 Function histogram_array(data_array, binsize=5, nbits=14) implemented with new inpu
    - 2016-08-09 Common mode correction with Gaussian fit to get the mean value per frame
    - 2016-08-05 Add common mode correction for function getHist() and plotHist() and calNoise_ADU()
    - 2016-08-05 Add functions calGain_Xray_lmfit() and getGain_Xray_lmfit() using lmfit module for g
    - 2016-08-04 Add error handling for function calNoise_ADU()
    - 2016-08-03 Solve the run_index non-umbiguous problem for function getHist() and convertEnergy(
    - 2016-08-03 Merge with the modification by Andrea
    - 2016-08-03 Correct the threshold input error for function index_peaks()
    - 2016-07-28 Change the way of file sorting
    - 2016-07-25 Decode gain bit in fixed gain mode for functions: getHist() and convertEnergy()
    - 2016-07-21 First creation

# Created by jiaguo.zhang@psi.ch
"""

#!/usr/bin/env python
# ld elements in base 2, 10, 16.

import os,sys
from numpy import *
from matplotlib.pyplot import *
from pylab import *
from scipy.optimize import curve_fit
import peakutils
from lmfit.models import GaussianModel, ExponentialModel
from lmfit import minimize, Minimizer, Parameters, Parameter, report_fit



############### FUNCTION COLLECTIONS FOR DATA CONVERSION ###############
# global definition
# base = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F]
base = [str(x) for x in range(10)] + [ chr(x) for x in range(ord('A'),ord('A')+6)]

```

```
# bin2dec
def bin2dec(string_num):
    return str(int(string_num, 2))

# hex2dec
def hex2dec(string_num):
    return str(int(string_num.upper(), 16))

# dec2bin
def dec2bin(string_num):
    num = int(string_num)
    mid = []
    while True:
        if num == 0: break
        num,rem = divmod(num, 2)
        mid.append(base[rem])

    return ''.join([str(x) for x in mid[::-1]])

# dec2hex
def dec2hex(string_num):
    num = int(string_num)
    mid = []
    while True:
        if num == 0: break
        num,rem = divmod(num, 16)
        mid.append(base[rem])

    return ''.join([str(x) for x in mid[::-1]])

# hex2tobin
def hex2bin(string_num):
    return dec2bin(hex2dec(string_num.upper()))

# bin2hex
def bin2hex(string_num):
    return dec2hex(bin2dec(string_num))
########################################################################



################### DEFINE FUNCTIONS FOR FITTING #######################
# Define fitting functions
# Gaussian
def func_gauss(x, A, xbar, sigma):
    return A*exp(-(x-xbar)**2/(2*sigma**2))
# Linear
def func_lin(x, k, b):
    return k*x+b

# Sqrt
def func_sqrt(x, k, b):
    return k*sqrt(x)+b

# Sqrt all component
def func_sqrt_all(x, k, b):
    return sqrt(k*x+b)

```

```
# Log10
def func_log10(x, k, b):
    return k*log10(x+1)+b

# Linear + exponential
def func_lin_exp(x, k, b, A, tau):
    return k*x+b-A*exp(-x/tau)

# Pure exponential
def func_exp_pure(x, A, tau):
    return A*exp(-x/tau)

# Exponential with shift
def func_exp_shift(x, A, tau, xbar):
    return A*exp((x-xbar)/tau)
########################################################################


############################## Histogram related ######################
# Plot the distribution of ADU in histogram
def plot_histogram(data_array, binsize=5, style="-", label=""):
    """
    # Use hist from matplotlib function
    n, bins, patches = hist(data_array, bins=16385/binsize, range=(0,16385), histtype="step", align =
    print(len(n), len(bins))
    """
    # Use histogram from numpy function
    range_ADU = linspace(0, 16385, 16386)
    bins = range_ADU[0::binsize]
    occurance = histogram(data_array, bins)[0]
    plot(bins[1:], occurance, style, label=label, drawstyle="steps-mid", linewidth=2.0)
    legend(loc=1, frameon=False, fontsize=10)
    grid(True)
    xlim(0,16000)
    xlabel("ADU")
    ylabel("Occurance")
    return bins[1:], occurance, std(data_array), mean(data_array)

# Generate the histogram for input data array
def histogram_array(data_array, binsize=5, nbits=14):
    nint = 2**nbits/binsize
    if mod(2**nbits,binsize) != 0:
        nint = nint + 1
    range_ADU = linspace(0, nint*binsize, nint*binsize+1)
    bins = range_ADU[0::binsize]
    occurance = histogram(data_array, bins)[0]
    return bins[1:], occurance

# Generate the histogram for input data array
def histogram_array_ADC(data_array, nbits=12, binsize=5):
    range_ADU = linspace(0, 2**nbits, 2**nbits+1)-binsize/2.0
    bins = range_ADU[0::binsize]
    occurance = histogram(data_array, bins)[0]
    return bins[1:], occurance

# Generate the histogram for input data array
def histogram_array_range(data_array, bins):
    occurance = histogram(data_array, bins)[0]
```

```
    return bins[1:], occurance
########################################################################



######################## Fitting related func ############################
# Get mean and sigma from a gaussian fitting
def para_gauss_fit(xdata, ydata, p0, thr=0):
    ydata_thr = ydata[where(ydata>=thr)[0]]
    xdata_thr = xdata[where(ydata>=thr)[0]]
    if len(ydata_thr) > 1:
        try:
            popt, pcov = curve_fit(func_gauss, xdata_thr, ydata_thr, p0=p0)
        except RuntimeError:
            popt = zeros(3, dtype="uint32")
            popt[0] = max(ydata_thr)
            popt[1] = xdata_thr[where(ydata_thr==max(ydata_thr))[0]]
            popt[2] = 0
            pcov = zeros((3,3))
            return popt, pcov
    else:
        popt = ydata_thr[0], xdata_thr[0], 0
        #popt = 0, 0, 0
        pcov = zeros((3,3))
    return popt, pcov

# Get mean and sigma from a gaussian fitting, the curve_fit function with errors input for data
def para_gauss_fit_error_in(xdata, ydata, p0, thr=0, sigma=None, absolute_sigma=False):
    ydata_thr = ydata[where(ydata>=thr)[0]]
    xdata_thr = xdata[where(ydata>=thr)[0]]
    sigma_thr = sigma[where(ydata>=thr)[0]]
    if len(ydata_thr) > 1:
        try:
            popt, pcov = curve_fit(func_gauss, xdata_thr, ydata_thr, p0=p0, sigma=sigma_thr, absolute
        except RuntimeError:
            popt = zeros(3, dtype="uint32")
            popt[0] = max(ydata_thr)
            if len(where(ydata_thr==max(ydata_thr))[0])==1:
                popt[1] = xdata_thr[where(ydata_thr==max(ydata_thr))[0]]
            else:
                popt[1] = xdata_thr[where(ydata_thr==max(ydata_thr))[0][0]]
            popt[2] = 0
            pcov = zeros((3,3))
            return popt, pcov
    else:
        popt = ydata_thr[0], xdata_thr[0], 0
        pcov = zeros((3,3))
    return popt, pcov

# Get parameters from a linear fit, the curve_fit function with errors input for data
def para_lin_fit_error_in(xdata, ydata, p0, sigma=None, absolute_sigma=False):
    if len(ydata) > 1:
        try:
            popt, pcov = curve_fit(func_lin, xdata, ydata, p0=p0, sigma=sigma, absolute_sigma=absolut
        except RuntimeError:
            popt = zeros(2)
            pcov = zeros((2,2))
            return popt, pcov
```

```
    else:
        popt = zeros(2)
        pcov = zeros((2,2))
    return popt, pcov


# Peak indexes of distributions using peakutils packages
# Input: thres - in percentage of peak value
#        min_dist - in difference of index numbers
def index_peaks(data_array, thres=0.01, min_dist=40):
    return peakutils.indexes(data_array, thres=thres, min_dist=min_dist)


# Moving average
def moving_average(a, n=2) :
    ret = cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n


# Moving sum
def moving_sum(a, n=2) :
    ret = cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:]
########################################################################


######################## High level functions ##########################
# Get histogram from X-ray measurement
def getHist(Folder, Run_index, i_strip=500, binsize=5, common_correction="No", nbits=14):

    # Get all files with this index
    files = []
    files += [each for each in os.listdir(Folder) if each.endswith("_"+str(Run_index)+'.raw')]
    # Sort files by generation time
    #files.sort(key=lambda x: os.path.getmtime(x))    # Opt-1
    files.sort()    # Opt-2
    #print(files)

    N_channels = 1280    # Number of channels
    Header = 6           # x 16 bits (2 bytes)
    Header_odd = 4
    Header_even = 2

    Frame_length = Header + N_channels
    Frame_halflength = int(Frame_length/2)

    # loop all files
    for i in range(len(files)):
        #print("It is processing", i+1, "th file...")
        Filepath = Folder + files[i]

        Data = fromfile(Filepath, dtype=uint16, count=-1)

        nPackets = len(Data)/Frame_halflength
        #print("The number of packets received:", nPackets)
        #print(Data[:Frame_halflength])
```

```
        #print(Data[Frame_halflength:2*Frame_halflength])

        # Index of packets
        Index_packets = Data[0::Frame_halflength]
        #print(Index_packets)

        # Index for odd and even number of indexes
        Index_packets_odd = where(mod(Index_packets,2)==1)[0]
        Index_packets_even = where(mod(Index_packets,2)==0)[0]
        #print(Index_packets_odd, Index_packets_even)

        # Give a specific strip
        if i_strip <= N_channels/2 - 2 + 1:   # data in odd number of index
            Data_i_strip_file = Data[Index_packets_odd*Frame_halflength+i_strip+Header_odd-1]
        else:   # data in even number of index
            Data_i_strip_file = Data[Index_packets_even*Frame_halflength+i_strip-int(N_channels/2-2)-

        # Get rid of the gain bit of data
        # 2016-10-18 Comment out to take the gain bit info together with the real data
        #Data_i_strip_file[where(Data_i_strip_file<2**14)[0]] = Data_i_strip_file[where(Data_i_strip_
        #Data_i_strip_file[concatenate((where(Data_i_strip_file>=2**14)[0], where(Data_i_strip_file<2
        #Data_i_strip_file[where(Data_i_strip_file>=2**15+2**14)[0]] = Data_i_strip_file[where(Data_i_

        # Accumulate the data for each file
        #Data_i_strip.append(Data_i_strip_file)
        #print(i, len(files))

        # Check whether the common mode correction is on or not
        if common_correction == "No":
            Data_i_strip_file = Data_i_strip_file
        elif common_correction == "Yes":
            if len(Index_packets_odd) != len(Index_packets_even):
                print("Packets lost! Watch out the correctness of the common mode correction!")

            CM_corr = []
            for iPacket in range(len(Index_packets_odd)):
                data_1Packet = Data[Index_packets_odd[iPacket]*Frame_halflength+Header_odd:Index_pack
                data_2Packet = Data[Index_packets_even[iPacket]*Frame_halflength+Header_even:Index_pa
                data_frame = concatenate((data_1Packet, data_2Packet))

                # Opt-1: Use a mean value of each frame to make common correction
                CM_corr = append(CM_corr, mean(data_frame))
                """
                # Opt-2: Use a gaussian fitting to get the mean of gaussian
                bins_frame, occurance_frame = histogram_array(data_frame, binsize=binsize)
                sigma_frame, pedestal_frame, sigma_error_frame, pedestal_error_frame = getNoise_ADU(b
                CM_corr = append(CM_corr, pedestal_frame)
                """

            # Take the first frame as reference
            if len(CM_corr) != 0:
                CM_corr = CM_corr - CM_corr[0]
                Data_i_strip_file = Data_i_strip_file - CM_corr
        else:
            print("The input for common_correction should be either Yes or No.")

        # Accumulate the accurrance
        bins, occurance_file = histogram_array(Data_i_strip_file, binsize=binsize, nbits=nbits)
```

```
        if i == 0:
            occurance = occurance_file
        else:
            occurance = occurance + occurance_file

    return bins, occurance


# Plot the histogram from Xray
def plotHist(Folder, Run_index, i_strip=500, binsize=5, common_correction="No", nbits=14):

    bins, occurance = getHist(Folder, Run_index, i_strip=i_strip, binsize=binsize, common_correction=

    # plot it
    figure("histogram")
    plot(bins, occurance, drawstyle="steps-pre", linewidth=1.0)
    grid(True)
    #xlim(0,16000)
    xlim(bins[where(occurance>1)[0]].min(), bins[where(occurance>1)[0]].max())
    ylim(1,)
    yscale("log")
    xlabel("ADU")
    ylabel("Occurance")
    show()

    return bins, occurance


# Calculate gain from X-ray fluorescence measurement using peakutils and curve_fit of scipy
def getGain_Xray(bins, occurance, E_xray=8.05, half_region=10, thres=0.2, min_dist=40):

    # Do peak finding and linear fitting here
    # Peak finding
    peak_index_guess = index_peaks(where(occurance>1, occurance, 1), thres=thres, min_dist=min_dist)
    #print("The index of peak position:", peak_index_guess)
    n_peaks = len(peak_index_guess)

    if n_peaks!= 0:
        E_peaks = linspace(0, n_peaks-1, n_peaks)*E_xray
        E_plot = linspace(-1, n_peaks, 100)*E_xray   # Generate a series energy points for plotting
        peak_pos_i_strip = zeros(n_peaks)
        peak_pos_error_i_strip = zeros(n_peaks)
        # Peak fitting
        for i in range(n_peaks):
            popt, pcov = para_gauss_fit_error_in(bins[peak_index_guess[i]-half_region:peak_index_gues
            #popt, pcov = para_gauss_fit_error_in(bins[peak_index_guess[i]-half_region:peak_index_gue
            peak_pos_i_strip[i] = popt[1]
            peak_pos_error_i_strip[i] = sqrt(pcov[1][1])


        # Calculate gain for the input strip from one ADC
        popt, pcov = para_lin_fit_error_in(E_peaks, peak_pos_i_strip, p0=[35.0, peak_pos_i_strip[0]],
        gain_i_strip = popt[0]
        offset_i_strip = popt[1]
        gain_error_i_strip = sqrt(pcov[0][0])

        return gain_i_strip, offset_i_strip, gain_error_i_strip, peak_pos_i_strip, E_peaks
```

```
    else:  # For the channel dead
        print("No proper value find by peakutils! A dead channel?")
        gain_i_strip = 0
        offset_i_strip = 0
        gain_error_i_strip = 0
        peak_pos_i_strip = 0
        E_peaks = 0

        return gain_i_strip, offset_i_strip, gain_error_i_strip, peak_pos_i_strip, E_peaks


# Calculate gain from X-ray fluorescence measurement using lmfit package
# Exray: X-ray energy
# gain_guess: guess value of gain in unit of ADU/keV
# sigma_guess: guess value of sigma in Gaussian fit
# prob_1ph: probably of seeing a single photon in a frame for peak value guess
def getGain_Xray_lmfit(bins, occurance, Exray=8.05, gain_guess=10.0, sigma_guess=15, prob_1ph=0.4):

    ADU_guess = gain_guess*Exray

    # Get the noise peak location
    if len(where(occurance==max(occurance))[0]) > 1:
        noise_peak_loc = bins[where(occurance==max(occurance))[0][0]]
    else:
        noise_peak_loc = bins[where(occurance==max(occurance))[0]]
    # Get the occurance of noise peak
    noise_peak_val = max(occurance)

    #print(noise_peak_loc, noise_peak_val)

    gauss1  = GaussianModel(prefix='g1_')
    pars = gauss1.make_params()
    #pars.update( gauss1.make_params())

    pars['g1_center'].set(noise_peak_loc, min=noise_peak_loc-3*sigma_guess, max=noise_peak_loc+3*sigm
    pars['g1_sigma'].set(sigma_guess)
    pars['g1_amplitude'].set(noise_peak_val)

    gauss2  = GaussianModel(prefix='g2_')

    pars.update(gauss2.make_params())

    pars['g2_center'].set(noise_peak_loc+ADU_guess, min=noise_peak_loc+3*sigma_guess)
    pars['g2_sigma'].set(sigma_guess)
    pars['g2_amplitude'].set(noise_peak_val*prob_1ph)

    mod = gauss1 + gauss2

    # Get the fit
    out = mod.fit(occurance, pars, x=bins)

    # Calculate gain
    gain = abs(out.best_values["g1_center"] - out.best_values["g2_center"])/Exray
    offset = min(out.best_values["g1_center"], out.best_values["g2_center"])
    gain_error = 0.0
    peak_pos = array([min(out.best_values["g1_center"], out.best_values["g2_center"]), max(out.best_v
    E_peaks = array([0.0, Exray])

```

```
    return gain, offset, gain_error, peak_pos, E_peaks



# Calculate conversion gains for all strips
def calGain_Xray(Folder, Run_index, binsize=5, E_xray=8.05, half_region=10, thres=0.2, min_dist=40, c

    #N_channels = 1280   # Number of channels
    N_channels = len(channels)
    Header = 6           # x 16 bits (2 bytes)
    Header_odd = 4
    Header_even = 2

    gain = zeros(N_channels)
    gain_error = zeros(N_channels)
    offset = zeros(N_channels, dtype="uint16")
    # Loop all strips by calling getHist() and getGain_Xray() functions
    i = 0
    for i_strip in channels.astype("uint16"):
        bins, occurance = getHist(Folder, Run_index, i_strip=i_strip, binsize=binsize, common_correct
        gain[i], offset[i], gain_error[i], dummy1, dummy2 = getGain_Xray(bins, occurance, E_xray=E_xr
        if mod(i,N_channels/10) == 0:
            print("Channel:", i_strip,", gain:", int(gain[i]*10)/10.0, "ADU/keV...")
        i = i + 1
    return gain, offset, gain_error


# Calculate conversion gains for all strips based on lmfit method
def calGain_Xray_lmfit(Folder, Run_index, binsize=5, Exray=8.05, gain_guess=10.0, sigma_guess=15, pro

    #N_channels = 1280   # Number of channels
    N_channels = len(channels)
    #print(N_channels)
    Header = 6           # x 16 bits (2 bytes)
    Header_odd = 4
    Header_even = 2

    gain = zeros(N_channels)
    gain_error = zeros(N_channels)
    offset = zeros(N_channels, dtype="uint16")
    # Loop all strips by calling getHist() and getGain_Xray() functions
    i = 0
    for i_strip in channels.astype("uint16"):
        #print(i)
        bins, occurance = getHist(Folder, Run_index, i_strip=i_strip, binsize=binsize, common_correct
        gain[i], offset[i], gain_error[i], dummy1, dummy2 = getGain_Xray_lmfit(bins, occurance, Exray
        if mod(i,N_channels/10) == 0:
            print("Channel:", i_strip,", gain:", int(gain[i]*10)/10.0, "ADU/keV...")
        i = i + 1
    return gain, offset, gain_error


# Calculate noise in terms of ADU from dark measurement
# Run getHist() first to get bins and occurance for the specific channel and run getNoise_ADU()
def getNoise_ADU(bins, occurance):
    # Fit the histogram
    # The initial guess
    occ_max = max(occurance)
```

```
    pos_max = bins[where(occurance==max(occurance))[0]]
    if len(pos_max) > 1:
        popt, pcov = para_gauss_fit(bins, occurance, p0=[occ_max, pos_max[0], 10.0], thr=0.1)
    else:
        popt, pcov = para_gauss_fit(bins, occurance, p0=[occ_max, pos_max, 10.0], thr=0.1)
    pedestal = popt[1]
    sigma = popt[2]
    pedestal_error = sqrt(pcov[1][1])
    sigma_error = sqrt(pcov[2][2])

    return sigma, pedestal, sigma_error, pedestal_error

# Calculate noise in terms of ADU for all strips
def calNoise_ADU(Folder, Run_index, binsize=5, common_correction="No", nbits=14):

    N_channels = 1280   # Number of channels
    Header = 6          # x 16 bits (2 bytes)
    Header_odd = 4
    Header_even = 2

    sigma = zeros(N_channels)
    sigma_error = zeros(N_channels)
    pedestal = zeros(N_channels, dtype="uint16")
    pedestal_error = zeros(N_channels, dtype="uint16")

    # Loop all strips by calling getNoise_ADU() and getHist() functions
    for i in range(N_channels):
        bins, occurance = getHist(Folder, Run_index, i_strip=i+1, binsize=binsize, common_correction=
        try:
            sigma[i], pedestal[i], sigma_error[i], pedestal_error[i] = getNoise_ADU(bins, occurance)
        except ValueError:
            sigma[i], pedestal[i], sigma_error[i], pedestal_error[i] = [0, 0, 0, 0]
        if mod(i+1, 1) == 0:
            print("Channel:", i+1, ", noise in ADU:", int(sigma[i]*10)/10.0, "ADU...")

    return sigma, pedestal, sigma_error, pedestal_error

# Convert noise in ADU to noise in electrons
# Keep the noise_ADU and gain the same dimension
def convertNoise(noise_ADU, gain):
    noise_e = noise_ADU/gain*1000/3.6
    return noise_e

# Generate a mask for data correction
# Noise or gain data can be input, the lower and upper boundary defined for good data, others masked
def genMask(data, boundary_low=-1, boundary_high=inf):
    # define an intial mask: 1 representing good strip, 0 for bad
    mask = ones(len(data))
    mask[concatenate((where(data<boundary_low)[0],where(data>boundary_high)[0]))] = 0

    return mask


# Convert the measurement data into photon energy on a basis of per frame and save a copy into a hdf
def convertEnergy(Folder, Run_index, pedestal, gain, mask=ones(1280)):

    # Get all files with this index
    files = []
```

```
    files += [each for each in os.listdir(Folder) if each.endswith("_"+str(Run_index)+'.raw')]
    # Sort files by generation time
    files.sort(key=lambda x: os.path.getmtime(os.path.join(Folder, x)))

    N_channels = 1280    # Number of channels
    Header = 6           # x 16 bits (2 bytes)
    Header_odd = 4
    Header_even = 2

    Frame_length = Header + N_channels
    Frame_halflength = int(Frame_length/2)


    import h5py
    # The output file creation
    #f5 = h5py.File(Folder + "/convertEnergy.hdf", "w")
    f5 = h5py.File(Folder + "/"+ "convertEnergy_%03i.nxs", "w", driver="family",memb_size=20*1024**3)
    grp_frames = f5.create_group("Frames")
    dset = grp_frames.create_dataset('data', (1,N_channels), maxshape=(None,N_channels), dtype=float,
    dsequence = grp_frames.create_dataset("sequence_number", (1,1,), maxshape=(None,1,), dtype=np.uir

    mycnt=0
    # loop all files
    for i in range(len(files)):

        Filepath = Folder + files[i]

        Data = fromfile(Filepath, dtype=uint16, count=-1)

        nPackets = len(Data)/Frame_halflength
        nFrames = nPackets/2
        #print("The number of packets received:", nPackets)
        #print(Data[:Frame_halflength])
        #print(Data[Frame_halflength:2*Frame_halflength])

        # Index of packets
        Index_packets = Data[0::Frame_halflength]
        #print(Index_packets)

        # Index for odd and even number of indexes
        Index_packets_odd = where(mod(Index_packets,2)==1)[0]
        Index_packets_even = where(mod(Index_packets,2)==0)[0]
        #print(Index_packets_odd, Index_packets_even)

        # Pre-define data arrays
        data_frame_1st_half = zeros(Frame_halflength-Header_odd)
        data_frame_2nd_half = zeros(Frame_halflength-Header_even)


        # Loop the packets
        for j in range(len(Index_packets)):

            if mod(Index_packets[j],2) == 1:
                if Index_packets[j+1]-Index_packets[j] == 1:
                    data_frame_1st_half = Data[j*Frame_halflength+Header_odd:(j+1)*Frame_halflength]
                    data_frame_2nd_half = Data[(j+1)*Frame_halflength+Header_even:(j+2)*Frame_halfler

                    data_merge = concatenate((data_frame_1st_half, data_frame_2nd_half))
```

```
                        # Get rid of the gain bit of data
                        data_merge[where(data_merge<2**14)[0]] = data_merge[where(data_merge<2**14)[0]]
                        data_merge[concatenate((where(data_merge>=2**14)[0], where(data_merge<2**15+2**1
                        data_merge[where(data_merge>=2**15+2**14)[0]] = data_merge[where(data_merge>=2**1

                        data_corrected = (data_merge - pedestal)/gain

                        # mask out the bad data
                        data_corrected[where(mask==0)[0]] = 0

                        # Create dateset and merge the two packets
                        #data_corrected_frame = grp_frames.create_dataset(str(int(floor(j/2)+1)), (N_char
                        #data_corrected_frame[:] = data_corrected
                        #del data_corrected_frame

                        # New method to write in data
                        mycnt+=1
                        if mycnt > 1:
                            dset.resize((mycnt,N_channels))
                            dsequence.resize((mycnt,1))
                        dset[mycnt-1,:] = data_corrected
                        dsequence[mycnt-1,:] = mycnt

    # Close the hdf5 file
    f5.close()
    del f5
    del dset
    del dsequence


# Merge several files into one
# Merge binary files: all files must include same number of frames with same size
def file_merger_const(Path_open, File_base_in, index_i, N_files, Path_save):
    # Open the first file and detect its length
    #File_open = Path_open+"/"+File_base_in+str(index_i)+".bin"
    File_open = Path_open+"/"+File_base_in+str(index_i)+".raw"
    data_length = len(fromfile(File_open, dtype=uint16, count=-1))
    data_out = zeros(data_length*N_files, dtype="uint16")
    for j in range(N_files):
        #File_open = Path_open+"/"+File_base_in+str(index_i+j)+".bin"
        File_open = Path_open+"/"+File_base_in+str(index_i+j)+".raw"
        data_file = fromfile(File_open, dtype=uint16, count=-1)
        data_out[j*data_length:(j+1)*data_length] = data_file
        del data_file
        print str(j+1), "th file out of", str(N_files), " files;"
    #File_save = Path_save+"/"+File_base_in+"merge.bin"
    File_save = Path_save+"/"+File_base_in+"merge.raw"
    fd = open(File_save, "w")
    data_out.astype(uint16).tofile(fd)
    fd.close()
########################################################################
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search