

Investigation of the FLASH DAQ Timing System

Konstantin Kharitonov

Ural Federal University, Ekaterinburg, Russia

Supervisors: Stefan Düsterer, Erland Müller

DESY, Hamburg, Germany

Thursday 7th September, 2017

Abstract

This paper describes the process of testing mechanisms how the FLASH DAQ (data acquisition system) assigns unique numbers (pulse IDs) to soft x-ray pulses and how the system relates timestamp information acquired from different subsystems to the soft x-ray pulses.

Contents

1. Introduction	3
2. Measurement scheme	4
3. Idea description	5
4. Single Camera test.....	6
5. Multiple camera test	8
6. Multiple ADC test	9
7. Several ways of getting data.....	9
8. Data timestamps comparison	11
9. Multiple camera timestamp test	11
10. Conclusion.....	11
References	13
Appendix A	13

1. Introduction

At FLASH, the Free Electron Laser facility at DESY in Hamburg, the emission of photon pulses has special pulse structure. It consists of so-called pulse trains which are delivered with 10 Hz repetition rate. Each pulse train contains between 1 to 800 pulses separated by 1 to 20 μs and total length of the pulse train does not exceed 800 μs . Each pulse train is identified by an event number which also can be called bunch ID, or pulse ID, or burst ID. The ID is set 20 ms before the pulse arrives at the experiment and stays for 80 ms after the pulse. To be able to compare data measured by different detectors over time, each piece of data acquired by the measuring system also has a timestamp providing microsecond information [1]. Accuracy of the burst ID is very important at FLASH and at all free electron lasers in general. Since every laser pulse is result of the SAUSE (Self-amplified spontaneous emission) process, it is unique. So experimental conditions vary from pulse to pulse and it is important to know what pulse train corresponds to the data.

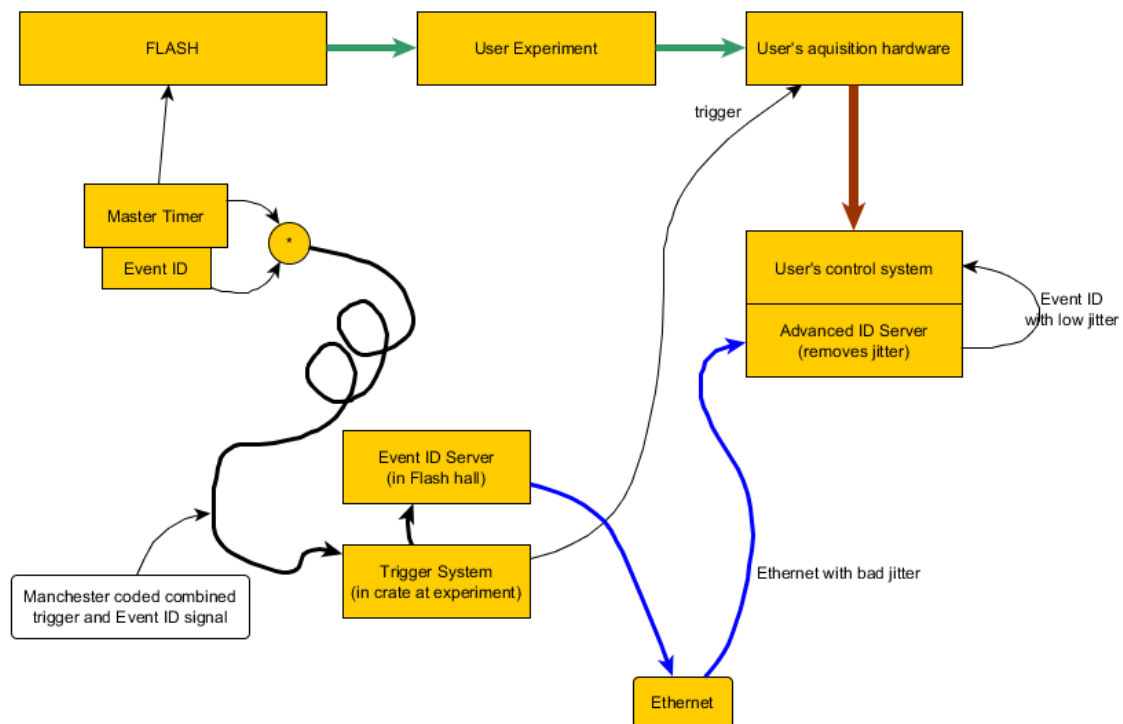


Figure 1 Basic principle of Event ID distribution [1]

The control and measurement system in the FLASH experimental halls consists of several MTCA (Micro Telecommunications Computing Architecture) stations and PCs, some of them are being used to connect analog devices using ADC and some to connect cameras. Besides, there are some additional DAQ (Data Acquisition) servers. Due to lack of synchronization with primary time servers there may be differences in local time between the machines. Together with latencies of the network regards transmission of the timing information this could lead to the change in the data's burst IDs and timestamps.

This report describes the methods which were used for reading information from the FLASH DAQ system, finding burst ID shifts and its (future) correction.

2. Measurement scheme

During the tests, the following measurement scheme was used: An alignment laser irradiated a photodiode (DET36A, Thorlabs) which was connected to two channels of a MHz ADC (SIS8300-L2 [2]) and a GHz ADC (ADQ412 [3]). In addition a camera (ACA1300-30gm [4]) imaged the photodiode [5]. The camera was connected to a camera PC. This setup allowed recording the alignment laser by several independent detectors. The fast shutter was used to create short laser flashes, by opening and closing the laser beam's path. Matlab was used for data acquisition and processing.

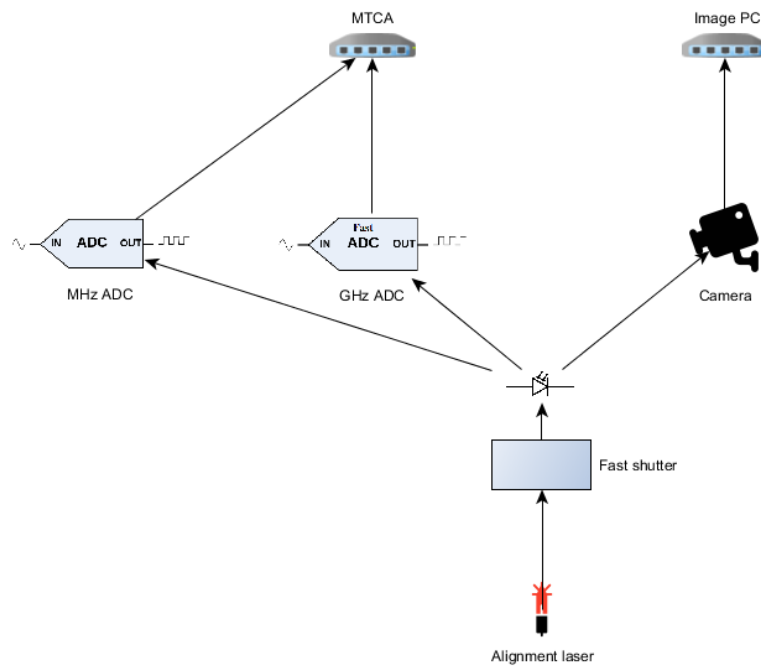


Figure 2 Measurement Scheme

3. Idea description

To be able to determine, if there is some burst-IDs shifting between two measurement devices and how much it is, I used two sequences of data which were taken in one experiment – one from each device. After that, I compared the data to determine, whether the burst-IDs of relevant data match. For more convenient processing I used data which could be easily “binarized”: 0 corresponds to closed shutter and 1 corresponds to open shutter. With a dataset like this, you can easily determine the presence of a shift and its magnitude by using a cross-correlation function [6], which is useful for determining the time delay between two signals. After calculating the cross-correlation between the two signals, the maximum of the cross-correlation function indicates the point in time where the signals are best aligned. Thus, for unshifted data correlation maximum should correspond to shift value 0 etc. Therefore, by using such a method for always two of the measurement devices you can automatically determine the magnitude of shifts and correct them. Otherwise you can choose one device as burst ID standard and correct all shifts according to this device.

The only disadvantage of the using correlation function to determine the shift value is that correlation function determines only shifts of data as one whole, whereas in our case, different data frames may have different shift value. Thereby only the most frequent magnitude of the shift can be found, corresponds to the maximum value of the cross-correlation.

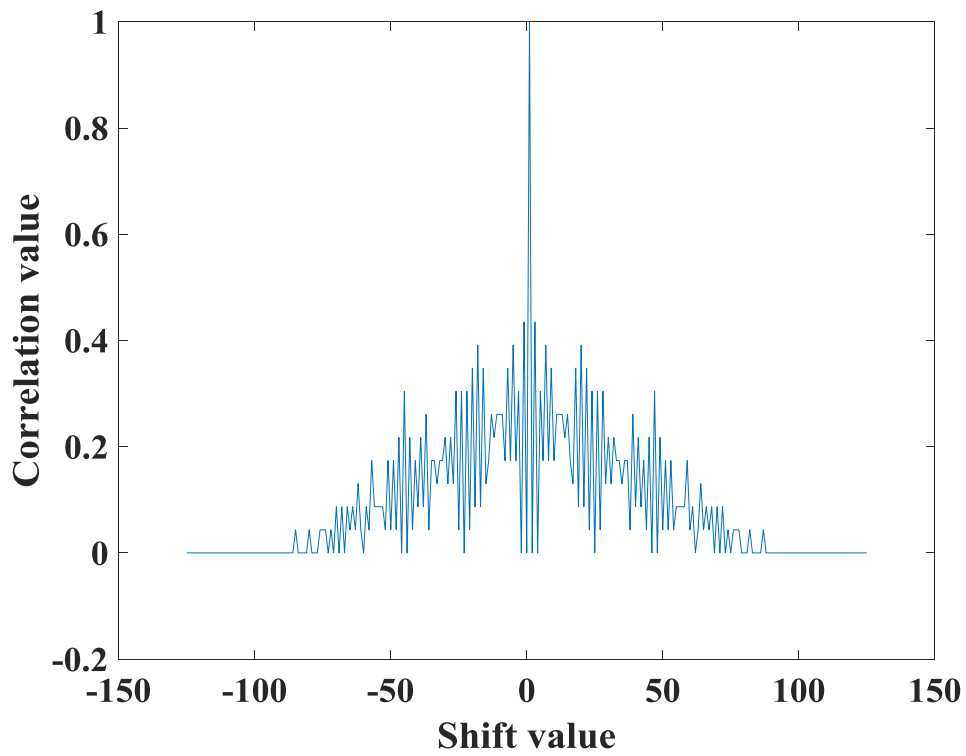


Figure 3: General view of the correlation function. Normalization means that correlation value of the two identical signals with zero delay equal to one.

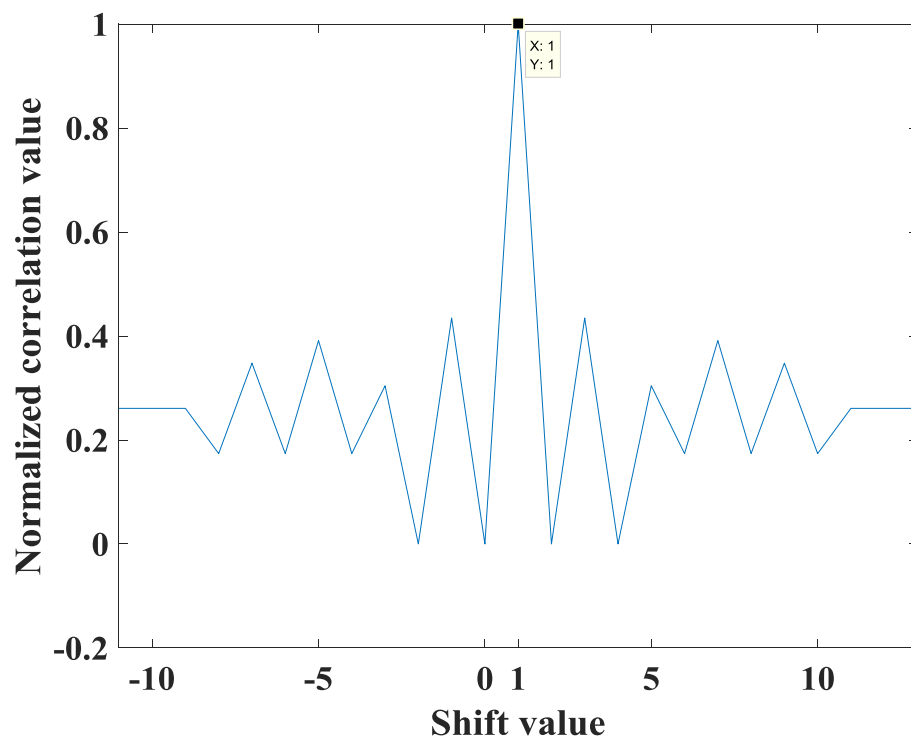


Figure 4: Cross correlation of data set where all data have shift equal to one.

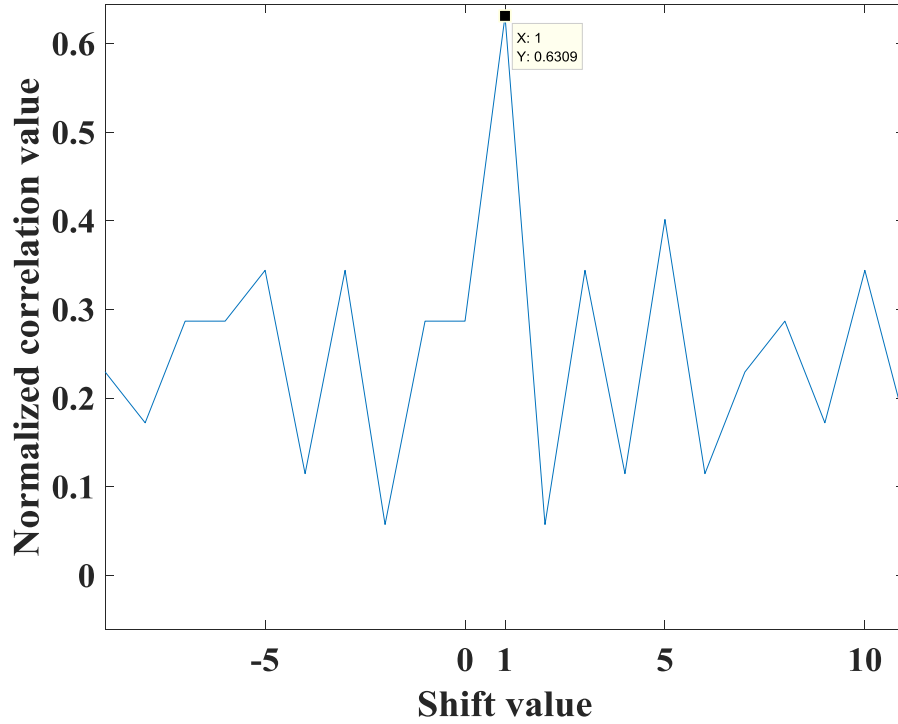


Figure 5: Cross correlation of data set where sixty percent of data have shift equal to one and thirty percent have shift equal to zero.

4. Single Camera test

The first test was performed using a camera and a photodiode, which was connected to a slow ADC and a “feedback” signal from the fast shutter together with the alignment laser. Thus, controlling the shutter, I could create a sequence of light pulses, which were registered by the photodiode and the camera. After that, all data was “binarized”, (mean value of image pixels was used in case of camera) and the magnitude of the shift was calculated. There are several camera parameters that one can change, which can affect the magnitude of the shift: resolution and image depth, which affects the image file size, and sender delay setting, which affects the time between camera taking an image and the data request from the DAQ. Datasets with different settings were analyzed, results are shown in the table.

Sender delay value ms \image size from maximum	1	1\2	1\4	1\8
20	93%	100%	No	No
40	No	No	No	No
60	No	No	No	No
80	No	No	No	No

Sender delay value ms \image size from maximum	1	1\2	1\4	1\8
20	100%	70%	No	No
40	100%	No	No	No
60	53%	No	No	No
80	No	No	No	No

Table 1 : Percent of shifts in data sequence depending on resolution image depth and sender delay left- 8bit image depth right- 16 bit image depth

The percent value of shifts shows proportion of data shifted to the most frequent shift value. It was calculated as ratio of the number of shifted data in the data sequence after the correction to the number of shifted data before the correction. Therefore, 100% of shifts mean that all data in sequence was shifted with the same magnitude, thus all shifts can be fully corrected. The value between 100% and zero means, that a random fraction of the recorded images was shifted; therefore you cannot correct all shifts automatically, but only those, which have the maximum shift magnitude (all images are shifted by the same amount). Also, combinations of settings which ensure no shifts were defined. In these measurements datasets from HDF5 files were used

5. Multiple camera test

Tests with several cameras using different image depth and resolution settings with optimal sender delay settings were performed. During the test up to four cameras were connected with image sized from maximum to one eighth of the maximum. Tests showed that in case of sixteen bit image depth only one camera can transmit data with full resolution settings without data loss. And four cameras can work without losses only with resolutions lower than one fourth from maximum. (Tests were performed using 1 GB network interface card (NIC), after changing the NIC to 10 GB results should significantly improve).

6. Multiple ADC test

Tests with multiple ADC channels in different MTCA crates were performed to find possible ID shifts between different ADC channels. Two channels of SIS8300 and two channels of ADQ412 were tested. A permanent shift between ADQ412 channels was detected. Each channel was located in a different MTCA crate. In addition the software version was different in the two MTCAs used. Repeated test after a software update of one crate showed that this shift was fixed completely.

7. Several ways of getting data

In the next series of measurements datasets were taken in several ways from several sources in DAQ and DOOCS (Distributed Object Oriented Control System) to determine if some points of the system induce changes of the timestamps. Data that were obtained from HDF5 files, through `daq_read` and `daq_read_svr` Mex functions, and through online reading from DOOCS using `jdoocs`

Data in .HDF5 files can be downloaded from the server through ftp client. Data received straight from .RAW files storage contained full set of timestamps.

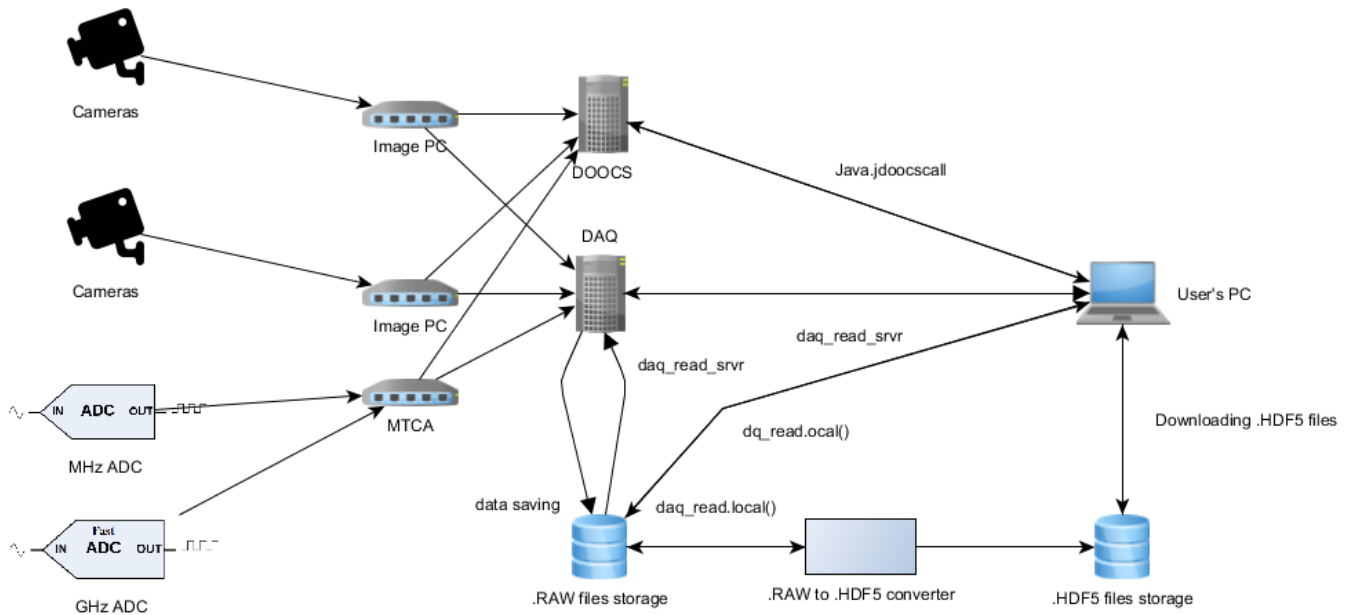


Figure 6 Data flow scheme

daq_read and daq_read_srvr are Matlab MEX functions which can be used to get data from the DAQ using Matlab. [7] The difference is that daq_read_srvr requests data from DAQ server while daq_read gets the data straight from .RAW file storage.

Tests showed that the data from the DAQ server have timestamps which are only filled in the second level with meaningful data while the microsecond timestamps contained only random two digits numbers. The data received straight from the .RAW files storage contained timestamps providing microsecond information.

Java.jdoocscall functions allow getting online data from DOOCS [8,9] to Matlab. The only drawback of this method is the low data collection rate, even when reading only two channels, the maximum data sampling rate is lower than 20 Hz, which however is enough since the camera takes shots with 10 Hz frequency. The data obtained by using Java.jdoocscall contains a full set of timestamps for the camera, but only a timestamp with seconds resolution for the ADC.

8. Data timestamps comparison

Data obtained from all sources was processed in Matlab. Firstly all datasets were binarized for more convenient correlation analysis, then the presence of shifts was tested, after confirmation of the absence of shifts the timestamps of the data with equal burst ID were checked, results are shown in the table below.

	MHz ADC		GHz ADC		Camera	
	Usec	Sec	Usec	Sec	Usec	Sec
.HDF5	identical	identical	identical	identical	identical	identical
daq_read	identical	identical	identical	identical	identical	identical
aq_read_srvr	Random	identical	Random	identical	Random	identical
Java.jdooocscall	can't read	identical	can't read	identical	identical	identical

Table 2: Results of timestamps testing

9. Multiple camera timestamp test

Timestamps of two cameras which were connected to different camera servers were checked through DOOCS. They also were identical to within microseconds

This confirms the theory that all timestamps in the measurement system are copied from the external burst ID timestamp.

10. Conclusion

In this project several tests of the DAQ system were performed.

A single camera and photodiode test allowed to find the settings that provide data collection from cameras without shifts of the burst ID. Tests have shown that the camera sender delay below 80ms with resolution greater than one fourth can cause shifts because the sender requests data from the camera before they are “ready”. A sender delay of 80 ms allows you to prevent shifts with any values of resolution and image depth.

As second part, limits of the resolution and the number of cameras that can be reliably were tested. It was found out that the system could collect data from cameras without data loss with one camera with full resolution, but only with one eighth resolution with three cameras because of data size increasing. In addition, all cameras must have the same image depth to work correctly. Updating the network card to a faster one removed the data size limitations.

Then, burst ID shifts between ADC channels were tested. To do that, an identical signal was connected to several channels of fast and slow ADCs. The test showed that there is no burst ID shift between slow ADC channels at any load, but fast ADC can show a burst ID shift between different MTCAs running with different software versions. This bug was fixed after the test.

Several ways to read data from DAQ and DOOCS were tested. For camera timestamps to within microseconds were received only from `daq_read`, DOOCS, and .HDF5 files. The timestamp field in data from `daq_read_svr` was blank. For ADC timestamps to within microseconds were received from every source except DOOCS and `daq_read_svr`, because ADC's microsecond timestamp from `daq_read_svr` looked like random set of two digits number. All timestamps were identical to within microseconds, what could lead us to conclusion that they all are inferred by the burst ID and not represent the local time of data acquisition, because different servers are definitely not synchronized to within microseconds, but every microseconds timestamp is identical to others from corresponding burst ID.

In future some deep inspection of the data's timestamps inside network should be performed to check timestamps at every point to understand better which one is correct and where are possible sources of shifts - and why. Also another kind of network architecture may be developed to either prevent timestamps from changing or increase time synchronization accuracy between different machines. Or the whole timing system could be based not on timestamps but on the clock signal which can be distributed across whole system. Such architecture would require some additional clock modules which can be connected to usual PC interfaces directly.

References

- [1] Description of bunch IDs system at FLASH
<http://hasfweb.desy.de/bin/view/Setup/BunchId>
- [2] SIS8300 datasheet <http://hasfweb.desy.de/pub/Setup/MtcaAdc/sis830012-m-x009-1-v101.pdf>
- [3] ADQ412 datasheet http://hasfweb.desy.de/pub/Setup/MtcaAdc/10-0494_C_ADQ412_datasheet.pdf
- [4] cA1300-30gm datasheet
<https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1300-30gm/>
- [5] DET36A datasheet <https://www.thorlabs.com/drawings/24e0f142ce925526-3232AF8F-F283-7060-28174CC98E5CCF2A/DET36A-Manual.pdf>
- [6] Cross correlation function <https://en.wikipedia.org/wiki/Cross-correlation>
- [7] DAQ Matlab manual
<http://www.desy.de/~wwwuser/daqmatlabaccessexamples.html>
- [8] JDOOCS documentation http://ttfinfo.desy.de/doocs/doocs_libs/jdoocs/doc/
- [9] DOOCS Matlab manual
<https://ttfinfo.desy.de/DOOCSWiki/Wiki.jsp?page=MATLABClientInterface>

Acknowledgements

Summer Student program at DESY was a unique experience for me. I would like to thank my supervisors Stefan Düsterer and Erland Müller for guiding me and answering my question. I would also like to thank the Summer Students Program organizers for allowing me to join this program. I also thank all people who were working around me at FLASH for their help or sometimes just talking.

Appendix A

Table of Contents

.Hd5 file usage:	1
DaqRead usage	2
Java.jdoocscall Usage	3

.Hd5 file usage:

```
%you can use built-in matlab function for .HD5 files
% List of .HDF5 names can be found here
%https://stash.desy.de/projects/CS/repos/pah/browse/data/
channel2HdfName.dat

addr='D:\Flash data/
FLASH1_USER2_stream_2_run18131_file1_20170728T154450.1.h5';
shutter=h5read(addr, '/Beamlines/BL/Fast shutter/shutter');% Shutter
%data reading
sh=mean(shutter,1);% you would have 2 dimensional array of ADC
spectrums
length=size(sh,2);% for shutter for simplicity you could use
%mean value of every spectrum
for l=1:1:length%l corresponds to opened shutter, 0 to closed
    if(sh(l)>0.5)
        sh(l)=1;
    else
        sh(l)=0;
    end
end

channel1=h5read(addr, '/Experiment/BL2/SIS8300 100MHz ADC/CH1/TD');
ch1=mean(channel1,1);%ADC data reading
m=min(ch1);
M=max(ch1);
ch1pr=ch1;
for l=1:1:length%
    if(ch1(l)>((M+m)/2))
        ch1(l)=1;
    else
        ch1(l)=0;
    end
end

cams=h5read(addr, '/Experiment/BL3/Camera 6/image.sec');%camera
timestamp
camus=h5read(addr, '/Experiment/BL3/Camera 6/image.usec');%and burst ID
camid=h5read(addr, '/Experiment/BL3/Camera 6/image.event');% data
reading
tmusec=h5read(addr, '/Timing/Bunch train info/index 1.usec');%Bunch
train
%timestamp data reading
```

```

camera=h5read(addr,'/Experiment/BL3/Camera 6/image');%camera images
reading
cammean=mean((mean(camera,1)),2);%Mean value of image pixels was used
%to determine presence of a laser beam and, consequently,
%the opening of a shutter
for l=1:1:length
    cam(l)=cammean(:, :, l);
end
length=size(cam,2);
campr=cam;
nonzero=find(campr>1);
mCam=min(campr(nonzero));
Mcam=max(campr(nonzero));

for l=1:1:length
    if(campr(l)>((Mcam+mCam)/2)*0.95)%Threshold coefficient
        cam(l)=1;%should be selected depending on the background
    else%brightness for effective binarization
        cam(l)=0;
    end
end

end

figure
plot(abs(cam-ch1));% This plot shows differences between camera a
title('Cam -ch1 before correction') %nd shutter signals
shifts_before=size((find(abs(cam-ch1))),2);

[acor lag]=xcorr(cam,ch1);% Using cross-correlation to
[~,I] = max(abs(acor));%find main shift magnitude value
lagDiff = lag(I);

camal = cam(lagDiff+1:end);%Move camera array to correct shifts

figure
stem(camal);
hold on
stem(ch1(1:end-lagDiff), 'marker', 'x');
hold off;
title('Cam and ch1 after ')
shifts_after=size((find(abs(camal-ch1(1:end-lagDiff))))),2);
figure
plot(abs(camal-ch1(1:end-lagDiff)))
title(1-(abs(shifts_after-shifts_before))/(shifts_before+1))
%Check how many shifts were corrected successfully

```

DaqRead usage

you need to know run number to get the data. Time does not need to be specified exactle,it works correctly even with random time, but do not exploit it. Adresses may be found here

```

%https://stash.desy.de/projects/CS/repos/pah/browse/data/
channel2HdfName.dat
%or in JDDD app. Last time DaqRead().local() ... contained seconds and

```

```

%microseconds timestamps, while DaqRead() contained only seconds and
some
%random stuff in microseconds timestamps

[shuttern, ids]=
DaqRead().flluser2().local().flluser2().run(18125 ).structs('2017-08-23
14:52',1,'FLASH.FEL/ADC.SIS.FL1FS/BL.SHUTTER');
[shuttern, ids]= DaqRead().flluser1().run(18397).structs('2017-08-23
14:56',1,'FLASH.FEL/ADC.SIS.FL1FS/BL.SHUTTER');

```

Java.jdoocscall Usage

information about java methods can be found here

```

%http://ttfinfo.desy.de/doocs/doocs_libs/jdoocs/doc/
%some methods which require pointers and arrays do not working with
matlab
%you can ask a custom method wrappers from Erland for them
%in this example custom wrapper was particularly used
%Java.jdoocscall allow you only to get live data

```

```

javaaddpath('\\win.desy.de\home\kharitok\My Documents\MATLAB')
import de.desy.fsfl.jdoocs.*

```

```

import ttf.doocs.*;
import ttf.doocs.clnt.*; % import of java methods to matlab

```

Set DOOCS channel address

```

%addresses can be found in JDDD or here
%*https://stash.desy.de/projects/CS/repos/pah/browse/data/
channel2HdfName.dat*

```

```

% Initialize DOOCS objects for equipment addresses
fCH = ttf.doocs.clnt.EqAdr();
fCH.adr('FLASH.FEL/ADC.ADQ.BL1/EXP1.CH03/CH00.TD');
SCH = ttf.doocs.clnt.EqAdr();
SCH.adr('FLASH.FEL/ADC.SIS.BL2/EXP2.CH01/CH00.TD');
SHU = ttf.doocs.clnt.EqAdr();
SHU.adr('FLASH.FEL/ADC.SIS.FL1FS/BL.SHUTTER/CH00.TD');
CAM = ttf.doocs.clnt.EqAdr();
CAM.adr('FLASH.FEL/FBL2.CAM/CAM6/IMAGE_EXT');

```

```

% Initialize DOOCS objects for synchronous call and data
eq = ttf.doocs.clnt.EqCall();
ed = ttf.doocs.clnt.EqData();

```

```

l=1;% you could run infinite loop for online data acquisition
%then just stop run using Control-C and save everything you need
while (l>0)

```

```

%Making calls
sh = eq.get(SHA, ed);
im = eq.get(CAMA, ed)a

```

```

fa = eq.get(fCH3A, ed);
sa = eq.get(SCH0A, ed);

%FAST ADC

fCH0(1)=mean(sh.get_spectrum.d_spect_array );
fCH0time(1)=sh.get_time;
fCH0timesp(1)=sh.get_spectrum.tm.value;

%SLOW ADC

SCH0(1)=mean(sh.get_spectrum.d_spect_array );
SCH0time(1)=sh.get_time;
SCH0timesp(1)=sh.get_spectrum.tm.value;

%SHUTTER
SHmean(1)=mean(sh.get_spectrum.d_spect_array );
SHtime(1)=sh.get_time;
SHstimesp(1)=sh.get_spectrum.tm.value;

%CAMERA
CAMstamp(1)=EqDataMatlab(im).pulseIdStamp();% custom wrapper method
CAMsec(1)=EqDataMatlab(im).secondsStamp();% EqDataMatlab
CAMmicro(1)=EqDataMatlab(im).microSeconsStamp();%used here
%ask Erland for it
image(1,:)=im.get_image_bytes();%image
width(1)=im.get_image_header().aoi_width;%width
heigth(1)=im.get_image_header().aoi_height;%heigth
imdate(1)=im.get_time;%jdoocs standart time mehod
% always returns 0 for images

end

```

Published with MATLAB® R2016a