Cluster Photon Finding for Integrating Charge Detectors

document v0.1, June 2018

Contents

Section 1

Introduction

MÖNCH is also a two-dimensional charge integrating detector detector for photon science applications.

As in the case of the JUNFGRAU detector, the information recorded by the detector is not the number of photons that hit the sensor, but an electric charge that is proportional to such number of photons over a certain period of time.

MÖNCH is a detector optimized for low flux applications where single photon resolution and low noise are the main requirements, whereas a high dynamic range is not of special interest.

As a result, two pieces of information are recorded for every pixel: the deposited charge and the employed gain stage that has been configured beforehand.

1.1 System overview

The MÖNCH chip comprises 400x400 pixels of 25x25 μm^2 each. The current prototype *MÖNCH_03* consists of a single chip for a total detection area of 1x1 cm^2. This prototype features up to three different gain stages that can be selected and remain fixed for the entire measurement.

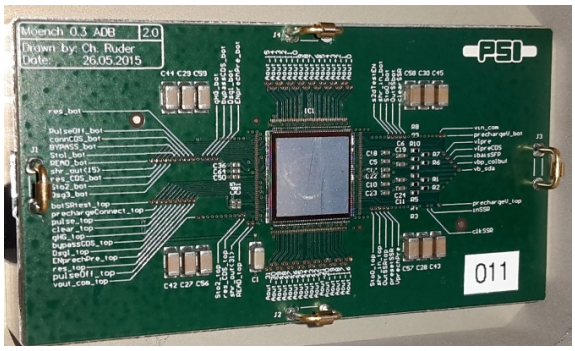

Figure 1.1: MÖNCH_03 chip

This prototype is connected to a JUNGFRAU-like Module Control Board (JMCB) that controls the MÖNCH chip during the acquisition and readout phase.

The detector is designed to support a maximum frame rate capability by design of 6kHz and will be connected to a MÖNCH Module Control Board (MMCB), whose main component will be an Altera Cyclone V Field Programmable Gate Array (FPGA).

This board will be the responsible of acquiring the digital output of the chip (employed gain bits) and the digital output of the 32 ADC channels that carry out the conversion of the analog output from the pixels. The whole stream of data will be descrambled to obtain a contiguous image that will be streamed out via ber links.

1.2 From charge to integrated energy

The process to convert the integrated charge to integrated energy is almost identical to the process carried out for JUNGFRAU detectors, being the only difference the fixed setting of the gain stage used for the measurement.

The data streamed out for every pixel consists of the digitized value of the deposited charge plus the coding of the employed gain stage together in a 16-bit word.

Bit	Symbol	Function
13-0	Charge	Digital value of the deposited charge.
15-14	Gain	Coding of the digital gain stage.

The conversion to integrated energy has to be carried out for every single pixel and with parameters that vary from pixel to pixel. These parameters are the actual gain for each gain stage and the offset due to noise and leakage, that also varies from gain stage to gain stage.

Nevertheless, since the gain stage has to be configured by the user beforehand, i.e., it does not change dynamically, it could be sufficient to load a single pair of gain-pedestal map with the values for a single gain stage.

In the general case for a MÖNCH detector, a minimum of 2 pixel maps need to be loaded onto the GPU: 1 gain map and 1 pedestal (offset) map for the gain level employed.

To make this application compatible with the current *photoncounter* application we could still load 6 pixel maps, even though only 2 of them (gain and pedestal) will be used for each measurementMÖNCH.

However, another option could be to parameterize the current application with the number of gain stages that will be used to load the corresponding number of gain maps, and to compute the initial pedestal maps out of the dark frames data-set.

Once this data has been loaded the conversion can be carried out by the same conversion algorithm that was developed.

1.3 Pedestal calculation and update

Pedestal values need to be computed when the pixel is not illuminated by any light source. However, as a difference to the gain maps, the pedestal map may have significant drifts due to the operation of the detector. Such drifts need to be taken into account, and therefore, pedestal changes need to be tracked periodically.

We differentiate between two operation modes in which this calculation can be carried out: a first operation mode in which the detector is completely in absence of light, and a second operation mode in which some of the pixels are not illuminated.

1.3.1 Pedestal reference in absence of light

In order to obtain a pedestal that can be used as a reference, the detector needs to be kept away from any light source. This situation can be forced by interrupting the experiment and closing the shutter of the detector.

Under these conditions, the pedestal reference is computed for all the pixels at the same time and for the different gain stages. The average of the last 1000 "dark" frames is used to compute the pedestal reference.

At this stage we also obtain the pedestal noise of each pixel $\sigma_{[x,y]}$, by calculating the standard deviation of the pedestal reference.

We will use the pedestal noise of the pixel $\sigma_{[x,y]}$ during the measurement to distinguish those pixels that were not hit by any photon from those who did detect some signal. This value of the pedestal noise is kept as a reference value and it is no longer modified until a new pedestal reference is taken.

1.3.2 Pedestal update algorithm

The calculation of the new pedestal is carried out only in those pixels that were not hit by any photon in the frame we are processing. We define these pixels as "dark" pixels and we detect them in practice when their ADC value is between the range $[P_{[x,y]} - c \cdot \sigma_{[x,y]}\ ,\ P_{[x,y]} + c \cdot \sigma_{[x,y]}]$

$$P_{[x,y]}[n-1] - c \cdot \sigma_{[x,y]} \leq ADC_{[x,y]}[n] \leq P_{[x,y]}[n-1] + c \cdot \sigma_{[x,y]} \tag{1.1}$$

Where $ADC_{[x,y]}[n]$ is the pixel value for the sample n, $P_{[x,y]}[n-1]$ is the computed pedestal for the previous sample $[n-1]$, $\sigma_{[x,y]}$ is the pedestal noise of each individual pixel that we have computed before and c is a parameter that is usually between 3 and 8, being 5 its default value.

Once the dark pixels have been detected, we can calculate the new pedestal value as the exponential moving average[1] of at least 1000 "dark" pixel samples.

$$M_{[x,y]}[n] = M_{[x,y]}[n-1] + ADC_{[x,y]}[n] - \frac{M_{[x,y]}[n-1]}{n} \tag{1.2}$$

$$P_{[x,y]}[n] = \frac{M_{[x,y]}[n]}{n} \tag{1.3}$$

Where $P_{[x,y]}[n]$ is the computed pedestal for the sample n, and being $P_{[x,y]}[0] = 0$ and $M_{[x,y]}[0] = 0$.

[1] `http://www.johndcook.com/blog/standard_deviation/`

Section 2

Cluster Photon Finding

In short, the Cluster Photon Finding (CPF) algorithm consists in identifying for each acquired frame those pixels whose signal is above a specific noise threshold $c \cdot \sigma$, and grouping them in clusters of $N \times N$ neighbouring pixels.

As a result, the output of this algorithm is an array of $N \times N$ clusters of pixels whose center is assigned to the pixel with the local maximum amplitude. Common cluster sizes the will be used are 2×2, 3×3, 5×5 and 11×11.

This algorithm is mainly useful in conditions where the the incoming photon flux is such that on average there is less than one photon detected per pixel cluster, condition that is known as *single-photon regime*.

In such conditions of low occupancy, it is possible to reduce the amount of data that needs to be stored significantly without loss of photon data.

In addition to the CPF algorithm, it is possible to exploit the charge sharing between neighbouring pixels to reconstruct the photon hit with better accuracy than the pixel pitch. This method is known as interpolation and could be executed after the extraction of photon clusters.

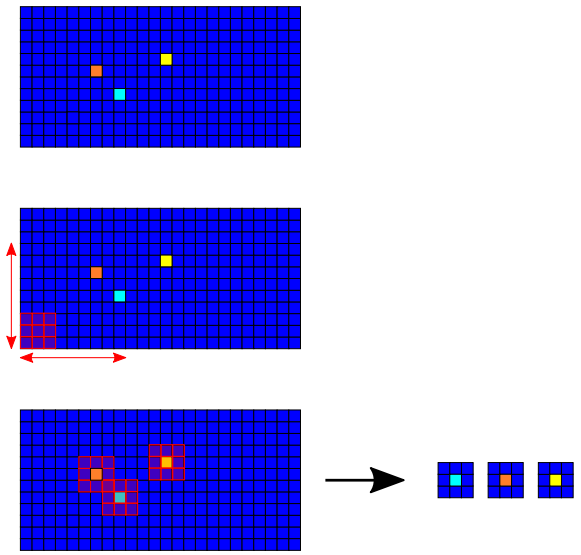

Figure 2.1: Sketch of the operation of a 3×3 Cluster Photon Finding algorithm

2.1 Application parameters

The following is a list of arguments we would need to pass to the application since they may change from measurement to measurement.

Argument	Default	Description
cpf	0	Cluster photon finding enable.
G	3	Number of gain stages of the detector.
E_{Beam}	1	Beam energy in keV.
c	5	Number of the standard deviations from the average.
N	3	Cluster size $N \times N$.

2.2 Data structure

The algorithm returns an array of clusters with the following structure

```
struct cluster
{
    int_64      frameNumber;

    int_16      coord_x;
    int_16      coord_y;

    int_32*     data;
};
```

2.2.1 Data structure description

int_32 frameNumber

The `frameNumber` member of the `cluster` structure is used to identify the frame where the cluster is detected.

int_16 coord_x

The `coord_x` member of the `cluster` structure is the x-coordinate of the cluster center, which in turns is the x-coordinate of the pixel with maximum amplitude.

int_16 coord_y

The `coord_y` member of the `cluster` structure is the y-coordinate of the cluster center, which in turns is the y-coordinate of the pixel with maximum amplitude.

int_32* data

The `data` member of the `cluster` structure is a pointer to the integrated energy of the pixels that belong to the cluster. For a 3×3 cluster it means the nine integrated energies of all the pixels in the cluster.

2.3 Pre-processing

To execute the CPF algorithm it is necessary to carry out first the conversion to integrated energy per pixel. As we have already mentioned, the process to obtain a "photon image" is identical to the process used for JUNGFRAU detectors, the only difference is the number of gain stages used by the detector, and the application of the new threshold to detect "dark" pixels.

Besides, the final conversion and rounding from integrated energy to number-of-photons is not necessary since the output of this algorithm consists of the integrated energy of the pixels inside the cluster.

2.4 CPF algorithm

The current CPF algorithm is implemented in C++ code as a member function `getClusters` of the class `singlePhotonDetector` [1].

```
int* singlePhotonDetector::getClusters(char* data)
```

The CPF algorithm iterates along every pixel in the detector and defines a pixel cluster of size $N \times N$ that includes the pixel under analysis its neighbouring pixels within the range of the detector.

Based on the integrated energy of each pixel within the cluster, it is possible to identify the hit of a photon within the boundaries of the cluster when one of the following condition is satisfied:

$$E_{cal}[x, y] > c \cdot \sigma_{[x,y]} \tag{2.1}$$

$$\sum_{l=i-\lfloor \frac{N}{2} \rfloor}^{i+\lceil \frac{N}{2} \rceil - 1} \sum_{m=j-\lfloor \frac{N}{2} \rfloor}^{j+\lceil \frac{N}{2} \rceil - 1} E_{cal}[l, m] > N \cdot c \cdot \sigma_{[x,y]} \tag{2.2}$$

That is, the integrated energy in the pixel under analysis is greater than the "dark" pixel threshold (2.1), or the sum of the integrated energy of all the pixels inside the cluster is greater than the "dark" pixel threshold of the pixel being analyzed times the size of the cluster (2.2).

Thus, for each pixel in the detector the CPF algorithm defines a pixel cluster with its neighbouring pixels and checks if the above condition is met:

- If none of the above conditions (2.1) and (2.2) are satisfied, meaning that no photon hit any pixel of the cluster, we can qualify the the pixel under analysis as "dark" pixel and use it to update the pedestal only when:

  $$-c \cdot \sigma_{[x,y]} \leq E_{cal}[x, y] \leq c \cdot \sigma_{[x,y]} \tag{2.3}$$

  Similar to the condition (1.1) that we use when the users do not want to run the CPF algorithm in the application.

- If one of the above conditions (2.1) or (2.2) is satisfied, meaning that at least a photon hit any of the pixels of the cluster, a second check has to be done:

  (a) If the pixel being analyzed is a local maximum, that is, it is the the pixel of the cluster that integrated the greatest energy, this pixel is taken as the center of the cluster and a new cluster is added to the cluster array

  (b) If the pixel being analyzed is not a local maximum, the analysis for this pixel is finished: the cluster is not added to the cluster array, and the pixel cannot be qualified as "dark" pixel.

Once this process has been carried out for all the pixels of the detector, all the photon data information needed is saved in the array of clusters and the analysis of the current frame finishes.

[1] `https://github.com/ComputationalRadiationPhysics/jungfrau-photoncounter/files/1521174/slsDetectorCalibration.zip`