EIGER- short manual

September 17, 2018

Contents

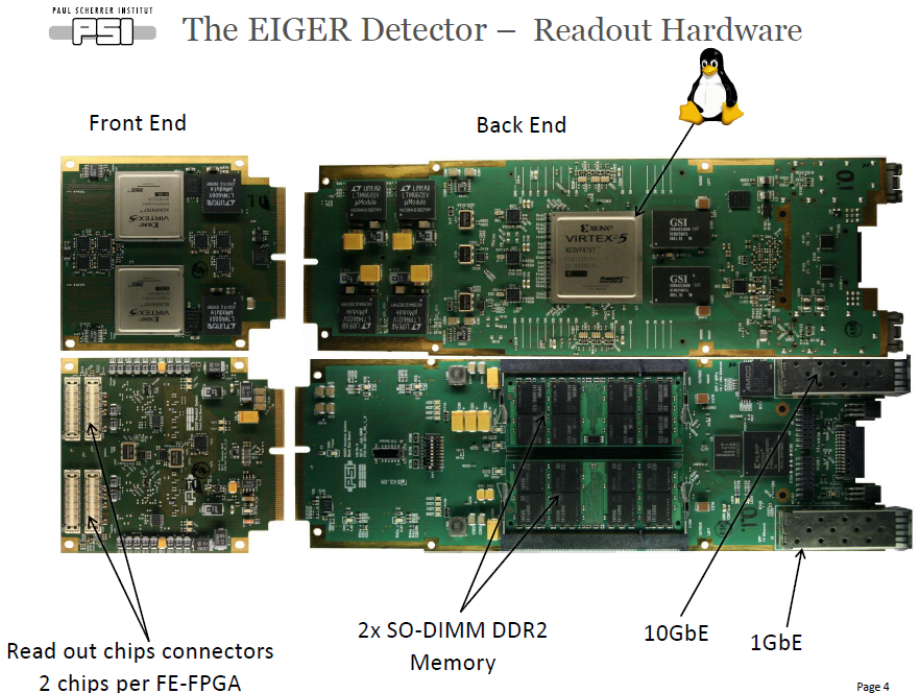

Figure 1: Picture with most relevant components of the EIGER readout system. The readout system starts with the Front End Boards (FEB) which performs data descrambling (also converts the packets from 12 → 16 bits) and rate correction. The BackEndBoard (BEB) has 2x2GB DDR2 memories and can perform data buffering (storing images on board) and data summation (16 bit → 32 bits). The controls to the detector are passed through the 1Gb, while in most installations, the data are sent out through the 10GB ethernet connection.

1 Usage

1.1 Short description

Figure 1 show the readout board basic components on an Eiger half module. An half module can read up to 4 readout chips.

1.2 Mandatory setup - Hardware

An EIGER single module (500 kpixels) needs:

- A chilled (water+alcohol) at 21 °C for a single module (500k pixels), which needs to dissipate 85 W (every module, i.e. for two half boards). For the 9M, 1.5M, a special cooling liquid is required: 2/3 deionized water and 1/3

ESA Type 48. This is important as the high temperature generated by the boards accelerate the corrosion due to Cu/Al reaction and the blockage of the small channels where the liquid flows, in particular near the face of the detector and if it is a parallel flow and not a single loop. The 9M and 1.5M run at 19 °C.

- A power supply (12 V, 8 A). For the 9 M, a special cpu is give to remotely switch on and off the detector: see section 1.2.1.

- 2×1 Gb/s Ethernet connectors to control the detector and, optionally, receive data at low rate. A DHCP server that gives IPs to the 1 Gb/s connectors of the detector is needed. Note that flow control has to be enabled on the switch you are using, if you plan to read the data out from there. If not, you need to implement delays in the sending out of the data.

- 2×10 Gb/s transceivers to optionally, receive data at high rate. The 10Gb/s transceiver need to match the wavelength (long/short range) of the fibers chosen by the beamline infrastructure.

The equipment scales linearly with the number of modules. Figure 2 shows the relationship between the **Client** (which sits on a beamline control PC), the **Receiver** (which can run in multiple instances on one or more PCs which receive data from the detector. The receiver(s) does not necessary have to be running on the same PC as the client.) The username under which the receiver runs is the owner of the data files, if using our implementation. It is important that the receiver is closely connected to the detector (they have to be on the same network). Note that if you implement the 1Gb/s readout only: client, receiver and detector have to be all three in the same network. If you implement the 10Gb/s readout, then client, the 1 GbE of the detector and the receiver have to stay on the 1GbE. But the receiver data receiving device and the 10GbE detector can be on their private network, minimizing the missing packets.

The Client talks to control over 1 Gb Ethernet connection using TCP/IP to the detector and to the receiver. The detector sends data in UDP packets to the receiver. This data sending can be done over 1 Gb/s or 10 Gb/s.

- **Switch on the detector only after having started the chiller: the 500k single module and the 1.5M at cSAXS/OMNY have a hardware temperature sensor, which will power off the boards if the temperature is too high. Note that the detector will be power on again as soon as the temperature has been lowered. The 9M will not boot up without the correct waterflow and temperature has it has an integrated flowmeter.**

- **Switch on the detector only after having connected all the cables and network. EIGER is unable to get IP address after it has been switched on without a proper network set up. In that case switch off and on the detector again.**

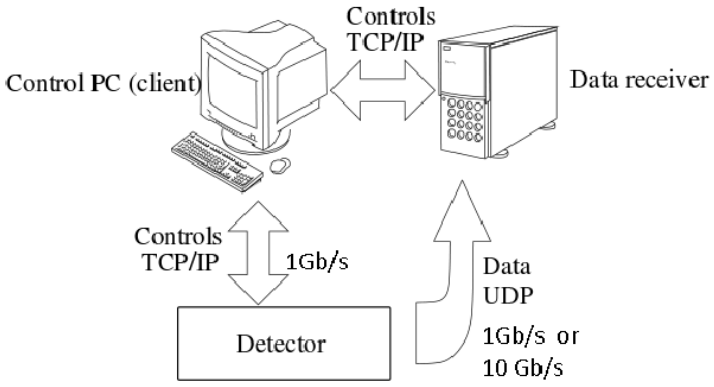

Figure 2: Communications protocol between the Client PC, the receiver PC and the detector.

1.2.1 9M power supply interface: bchip100

So the bchip100, which is a blackfin cpu, is located on the top side of the 9M and needs to be connected over 1Gb, to the same or a different network as the detector 1 GbE.

```
telnet bchip100
cd 9m/
```

The directory contains some executables that are needed to make your detector to work:

```
./on #to switch modules on
./off #to switch modules off
./hvget #gets the current HV value
./waterflow #returns the current waterflow returned by the flowmeter
./temp #returns the water temperature returned by the flowmeter
```

A watchdog is running on bchip100 to check for the flow and temperature. If outside of parameters (flow$<$ 80 dl/min, temperature $\neq$21$\pm$2), the detector will be switched off. Here is an explanation of the LED color scheme of the bchip100:

- NO LED Main Power off or Blackfin not ready, yet.
- RED Too high temperature or too less water flow Detector is shut down and locked. Detector will be unlocked (YELLOW) automatically when conditions are good again.

- YELLOW Detector is off and unlocked. Ready to be turned on.
- GREEN Detector is on

You can also Check temperatures and water flow in a browser (from the same subnet where the 9M is: http://bchip100/status.cgi

1.3 Mandatory setup - Receiver

The receiver is a process run on a PC closely connected to the detector. Open one receiver for every half module board (remember, a module has two receivers!!!) . Go to `slsDetectorsPackage/build/bin/`, **slsReceiver** should be started on the machine expected to receive the data from the detector.

- `./slsReceiver --rx_tcpport xxxx`
- `./slsReceiver --rx_tcpport yyyy`

where xxxx, yyyy are the tcp port numbers. Use 1955 and 1956 for example. The receiver for the bottom is open without arguments but still in the configuration file one needs to write `n:flippeddatax 1`, where `2n+1` indicated the half module number, 1 if it is a module.
Open as many receiver as half module boards. A single module has two half module boards.

From the software version 3.0.1, one can decide weather start a zmq callback from the receiver to the client (for example to visualize data in the slsDetectorGui or another gui). If the zmq steam is not required (cased of the command line for example, one can switch off the streaming with `./sls_detector_put rx_datastream 0`, enable it with `./sls_detector_put rx_datastream 1`. In the case of initializing the stream to use the slsDetectorGui, nothing needs to be taken care of by the user. If instead you want to stream the streaming on different channels, the zmq port of the client can be set stealing from the slsDetectorGui stream having `./sls_detector_put zmqport 300y`. Note that if this is done globally (not for every half module n independently, then the client automatically takes into account that for every half module, there are 2 zmq stream. The receiver stream `./sls_detector_put rx_zmqport 300y` has to match such that the GUI can work. If one desires to set the zmqport manually, he offset has to be taken into account: `./sls_detector_put 0:rx_zmqport 300y`, `./sls_detector_put 1:rx_zmqport 300y+2` and so on..

`slsMultiReceiver` uses two or more receivers in one single terminal: `./slsMultiReceiver startTCPPort numReceivers withCallback`, where startTCPPort assumes the other ports are consecutively increased.

The command `r_framesperfile` sets the number of frames written in the same file. By default now it is 10000. It can be changes. It needs to be lowered particularly if one wants to parallelize the following conversion of the files.

1.4 Mandatory setup - Client

The command line interface consists in these main functions:

sls_detector_acquire to acquire data from the detector

sls_detector_put to set detector parameters

sls_detector_get to retrieve detector parameters

First, your detector should always be configured for each PC that you might want to use for controlling the detector. All the examples given here show the command `0-`, which could be omitted for the EIGER system 0. In the case more EIGER systems are controlled at once, the call of `1-`,.. becomes compulsory.

To make sure the shared memory is cleaned, before starting, one should do:

```
sls_detector_get 0-free
```

To do that:

```
sls_detector_put 0-config mydetector.config
```

In the config file, if client, receiver and detector are using **1GbE** the following lines are mandatory (see slsDetectorsPackage/examples/eiger_1Gb.config):

```
detsizechan 1024 512 #detector geometry, long side of the module first
hostname beb059+beb058+ #1Gb detector hostname for controls
0:rx_tcpport 1991 #tcpport for the first halfmodule
0:rx_udpport 50011 #udp port first quadrant, first halfmodule
0:rx_udpport2 50012 #udp port second quadrant, first halfmodule
1:rx_tcpport 1992 #tcpport for the second halfmodule
1:rx_udpport 50013 #udp port first quadrant, second halfmodule
1:rx_udpport2 50014 #udp port second quadrant, second halfmodule
rx_hostname x12sa-vcons #1Gb receiver pc hostname
outdir /sls/X12SA/data/x12saop/Data10/Eiger0.5M
threaded 1
```

In the config file, if client, receiver and detector commands are on 1Gb, but detector data to receiver are sent using **10GbE** the following lines are mandatory (see slsDetectorsPackage/examples/eiger_10Gb.config):

```
detsizechan 1024 512 #detector geometry, long side of the module first
hostname beb059+beb058+ #1Gb detector hostname for controls
0:rx_tcpport 1991 #tcpport for the first halfmodule
0:rx_udpport 50011 #udp port first quadrant, first halfmodule
0:rx_udpport2 50012 #udp port second quadrant, first halfmodule
0:rx_udpip 10.0.30.210 #udp IP of the receiver over 10Gb
0:detectorip 10.0.30.100  #first half module 10 Gb IP
1:rx_tcpport 1992 #tcpport for the second halfmodule
1:rx_udpport 50013 #udp port first quadrant, second halfmodule
```

```
1:rx_udpport2 50014 #udp port second quadrant, second halfmodule
1:rx_udpip 10.0.40.210  #udp IP of the receiver over 10Gb,
                        can be the same or different from 0:rx_udpip
1:detectorip 10.0.40.101 #second half module 10 Gb IP
rx_hostname x12sa-vcons #1Gb receiver pc hostname
outdir /sls/X12SA/data/x12saop/Data10/Eiger0.5M
threaded 1
```

In the case you are developing your own receiver, then you need to remove the 1Gb receiver hostname `rx_hostname` and substitute it with the mac address of the device:

```
configuremac 0
rx_udpmac xx:xx:...
```

One can configure all the detector settings in a parameter file `setup.det`, which is loaded by doing:

```
sls_detector_put 0-parameters setup.det
```

In the case of EIGER, the proper bias voltage of the sensor has to be setup, i.e. the `setup.det` file needs to contain the line `vhighvoltage 150`. Other detector functionality, which are rarely changed can be setup here. Other important settings that are configured in the `setup.det` file are:

- `tengiga 0/1`, which sets whether the detector is enabled to send data through the 1 or the 10 Gb Ethernet.
- `flags parallel/nonparallel`, which sets whether the detector is set in parallel acquisition and readout or in sequential mode. This changes the readout time of the chip and affects the frame rate capability (faster is `parallel`, with higher noise but needed when the frame rate is > 2 kHz.
- `dr 32/16/8/4` sets the detector in autosumming mode (32 bit counter or not autosumming, 12 bit out of the chip). This is strictly connected to what is required for the readout clock of chip. See next point.
- `clkdivider 0/1/2`. Changes the readout clock: 200, 100, 50 MHz (also referred to as full, half, quarter speed). Note that autosumming mode (`dr 32` only works at clkdivider 2=quarter speed). By selecting Refer to readout timing specifications in section6 for how to set the detector.
- `flags continuous/storeinram`. Allows to take frame continuously or storing them on memory. Users should use the `continuous` flags. Enabling the `stroreinram` flag makes the data to be sent out all at the end of the acquisition. Refer to readout timing specifications in section 6 for how to set the detector. Examples will be given in section **??**.

One should notice that, by default, by choosing the option `dr 32`, then the software automatically sets the detector to `clkdivider 2`. By choosing the option `dr 16`, the software automatically sets the detector to `clkdivider 1`. One needs to choose `clkdivider 0` after setting the `dr 16` option to have the fastest frame rate. We would recommend expert users (beamline people) to write their parameters file for the users.

2 API versioning

The eigerDetectorServer running on the boards has a versioning API scheme that will make it crash if used with a wrong firmware. You can also check your versioning by hand with the code:

```
sls_detector_get softwareversion
```

gets the server (slsDetectorSoftware) version (answer is something like: `softwareversion 111920160722`.

```
sls_detector_get thisversion
```

returns the client version. The answer can be `thisversion 111220160718`.

```
/sls_detector_get detectorversion
```

returns the firmware version . The answer can be `detectorversion 11`. Killing and starting the server on the boards allows you to check the firmware version you have and also if your board is a top/bottom/master/slave.

3 Setting up the threshold

```
sls_detector_put 0-trimen N xxxx yyyy zzzz
sls_detector_put 0-settings standard
sls_detector_put 0-threshold energy_in_eV standard
```

The first line requires to specify how many (`N`) and at which energies in eV {ttxxxx, `yyyy`, `zzzz` and so on) trimmed files were generated (to allow for an interpolation). This line should normally be included into the `mydetector.config` file and should be set for you by one of the detector group. NORMALLY, in this new calibration scheme, only `settings standard` will be provided to you, unless specific cases to be discussed. The threshold at 6000 eV , for example would be set as:`sls_detector_put 0-threshold 6000 standard`.

For EIGER, at the moment normally only `standard` settings are possible. `lowgain`, `verylowgain`, `veryhighgain` and `highgain` are theoretically possible, but we never calibrate like this. They could be implemented later if needed.

Notice that setting the threshold actually loads the trimbit files (and interpolate them between the closest calibration energies) so it is time consuming.

The threshold is expressed in (eV) as the proper threshold setting, i.e. normally is set to 50% of the beam energy.

We have added a special command, `thresholdnotb`, which allows to scan the threshold energy without reloading the trimbits at every stage. One can either keep the trimbits at a specific value (es.32 if the range of energies to scan is large) or use the trimbits from a specific energy (like a central energy).

```
sls_detector_put 0-thresholdnotb energy_in_eV
```

4 Standard acquisition

After you setup the setting and the threshold, you need to specify the exposure time, the number of real time frames and eventually how many real time frames should be acquired:

```
sls_detector_put 0-exptime 1[time_is_s]
sls_detector_put 0-frames 10
sls_detector_put 0-period 0[time_is_s]
```

In this acquisition 10 consecutive 1 s frames will be acquired. Note that `period` defines the sum of the acquisition time and the desired dead time before the next frame. If `period` is set to 0, then the next frame will start as soon as the detector is ready to take another acquisition.

You need to setup where the files will be written to

```
sls_detector_put 0-outdir /scratch
sls_detector_put 0-fname run
sls_detector_put 0-index 0
```

this way your files will all be named /scratch/run_dj_i.raw where j is relative to each specific half module, i in the `index` starts from 0 when starting the detector the first time and is automatically incremented. The next acquisition `index` will be 1. One can reset the `index` to what wished.

To acquire simply type:

```
sls_detector_acquire 0-
```

Note that acquiring is blocking. You can poll the status of the detector with:

```
sls_detector_get status
```

If the detector is still acquiring, the answer will return `running`. If the detector has finished and ready for the next acquisition, then it will return `idle`. You can also ask for the status of the receiver, to know when it has returned and finished getting the data with:

```
sls_detector_get receiver
```

There is a more complex way of performing an acquisition, that is useful for debugging and in case one wants a non blocking behavior:

You can then reset to zero the number of frames caught, then start the receiver and the detector:

1. `sls_detector_put 0-resetframescaught 0`
2. `sls_detector_put 0-receiver start`
3. `sls_detector_put 0-status start`

You can poll the detector status using:

```
sls_detector_get 0-status
```

When the detector is `idle`, then the acquisition is done but the receiver could still be receiving data. If you want, you can check if the receiver is finished receiving as many frames as you were expecting (this is optional but required for many many frames acquisition or when using some delays to send data at very high frame rate.

4. `sls_detector_get framescaught`

Then you can stop the receiver as well now:

5. `sls_detector_put 0-receiver stop`

The detector will not accept other commands while acquiring. If an acquisition wishes to be properly aborted, then:

- `sls_detector_put 0-status stop`

this same command can be used after a non proper abortion of the acquisition to reset to normal status the detector.

5 Gap pixels inside a module

A module is composed of 2×4 chips. Each chip is of dimension 256×256 pixels. There is no dead area in a module, as a single sensor covers the 8 chips. The physical pixels at the border of the chips in the sensor are double in size, to allow not to loose photons in the gaps between the chip alignment. They count double what the other normal pixels would count. In the corner between chips, the pixels are 4-times the normal size. See figure 3 to check the geometry.

It is possible to interpolated the value on the larger pixels by splitting the events (or properly interpolating) introducing a virtual pixel for every double pixel, or 3 virtual pixels for every corner. In this way the counts of a single large pixel can be shared among the correct amount of pixels of the normal dimension.

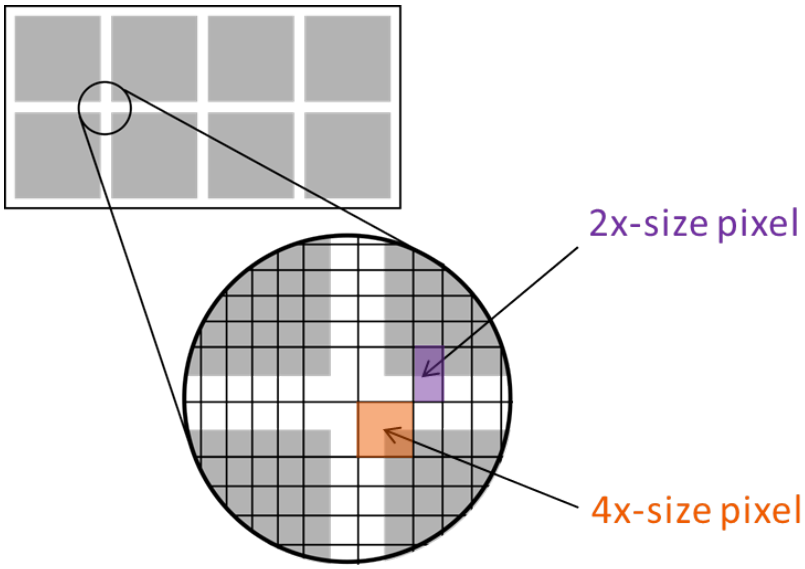

Figure 3: Geometry of gap pixels between a module.

The gap pixels can be added for the slsDetectorGui, from the datacall back or stealing the zmq port from the GUI (see later). The detector size can be added in the configuration file as first thing.

Putting the long side of the module first always as a convection for the code, WITHOUT GAP PIXELS an EIGER module is:

```
detsizechan 1024 512
```

and the client needs to be set `sls_detector_put gappixels 0`, which is the default behavior.
If you want to have GAP PIXELS included:

```
detsizechan 1030 514
```

and the client needs to be set `sls_detector_put gappixels 1`.

The size of the gap pixels between modules to insert is

```
GapPixelsBetweenModules_x = 8
GapPixelsBetweenModules_y = 36
```

where the `GapPixelsBetweenModules_x` are the one on the short side of the module, while `GapPixelsBetweenModules_y` are the ones on the long side of the module (where the wirebonds take physical space).

GbE	dynamic range	continuos maximum frame rate(Hz)	minimum period (μs)
1	16	**256**	3901
1	32	**128**	7820
10	16	**2560**	391
10	32	**1280**	782
10	8	**5120**	196
10	4	**10240**	98

Table 1: Frame rate limits for the CONTINUOS streaming out of images, i.e. the data rate out is just below 1Gb/s or 10Gb/s.

dynamic range	images
4	30000
8	15000
16	7600

Table 2: Amount of images that can be stored on board. As while we store them, we start to send them out, the effective number of images could be larger than this, but it will depend on the network setup (how fats you stream out images).

6 Readout timing- maximum frame rate

IMPORTANT: to have faster readout and smaller dead time, one can configure `clkdivider`, i.e. the speed at which the data are read, i.e. 200/100/50 MHz for `clkdivider 0/1/2` and the dead time between frames through `flags parallel`, i.e. acquire and read at the same time or acquire and then read out. The configuration of this timing variables allows to achieve different frame rates. NOTE THAT IN EIGER, WHATEVER YOU DO, THE FRAME RATE LIMITATIONS COME FROM THE NETWORK BOTTLENECK AS THE HARDWARE GOES FASTER THAN THE DATA OUT.

In the case of REAL CONTINUOUS readout, i.e. continuous acquire and readout from the boards (independent on how the chip is set), the continuous frame rates are listed in table 1. Note that in the `continuous` flag mode, some buffering is still done on the memories, so a higher frame rate than the proper real continuous one can be achieved. Still, this extra buffering is possible till the memories are not saturated. The number of images that can be stored on the DDR2 on board memories are listed in table 2.

The maximum frame rate achievable with 10 GbE, `dr 16`, `flags continuous`, `flags parallel`,`clkdivider 0`, **6.1 kHz**. This is currently limited by the connection between the Front End Board and the Backend board. We expect the 32 bit mode limit, internally, to be **2 kHz** (`clkdivider 2`). In dynamic range `dr 8` the frame rate is **11 kHz** and for`dr 4` is **22 kHz**. For 4 and 8 bit mode the frame rate are directly limited by the speed of the detector chip and not by the readout boards.

dr	clkdivider	expected chip readout t(μs)	measured chip readout t(μs)
4	0	41	40
4	1	82	84
4	2	123	172
8	0	82	82
8	1	164	167
8	2	328	336
12	0	123	122
12	1	246	251
12	2	491	500

Table 3: Readout time required from the chip to readout the pixels. The numbers are obtained using equation 5.

6.1 Minimum time between frames and Maximum frame rate

We need to leave enough time between an exposure and the following. This time is a combination of the time required by the chip, by the readout boards and eventually extra time to reduce some appearance of cross talk noise between the digital and analog parts of the chip. **It is essential to set the `period` of the detector, defined as the `exptime` plus an extra time, that needs to be at least the chip/board readout time. If this is set wrong (it is $<$ `exptime` plus chip/board readout time), then the detector takes the minimum time it can, but you are in a not controlled frame rate situation.**

The expected time difference between frames given by the pure chip readout time is in Table 3.

The `period` is s is defined as:

$$\text{period} = \text{exptime} + \text{minimum time between frames} \tag{1}$$

where the 'minimum time between frames' and the minimum period will be discussed in Table 4.

From software version 4.0.0, there is a very useful function `sls_detector_get measuredperiod` which return the measured period AFTER the acquisition. This is important to check that the settings to obtain the targeted frame rate was correct.

If you run too fast, the detector could become noisier (see problem shooting), it is important to match the detector settings to your frame rate. This can be done having more parameters files and load the one suitable with your experiment. We experienced that with low energy settings could not reach 6 kHz and no noise.

In 16 bit mode, it could make sense, in case of noise and low threshold to

dr	clkdivider	flags	t between frames(μs)	max frame rate (kHz)	min period (μs)	max imgs (nominal/our network)
4	0	parallel	3.4	22	44	30k/50k
4	1	parallel	6	10.5	92	30k/100k
4	2	parallel	11.2	5.4	197	infinite
8	0	parallel	3.4	11.1	89	15k/24k
8	1	parallel	6.1	5.7	181	15k/52k
8	2	parallel	11.2	2.9	342	infinite
16	0	parallel	3.4	6.1	(126+38)* =164	8k/12k
16	0	nonparallel	126	5.6	(126+52)*= 179	8k/23k
16	1	parallel	6.1	3.9	257	8k/28k
16	1	nonparallel	255	3.3	303	infinite
16	2	parallel	11	1.9	526	infinite
16	2	nonparallel	504	1.8	555	infinite

Table 4: Readout settings. The min exptime possible is 5−10 μs. This is due to the time to pass the pixel enable signal in the whole chip. The time between frames has been measured with the oscilloscope and the maximum frames rate has been tested with an external gating from a pulse generator at known frequency. The minimum period is obtained as 1/max frame rate.

either reduce the frame rate:

$$\text{new period} = \text{exptime} + \text{minimum time between frames} + (10{-}20\ \mu\text{s}) \quad (2)$$

to let the signal settle or, if the frame rate is important, leave the `period` at the same value but reduce the `exptime`:

$$\text{new exptime} = \text{old exptime} - (10{-}20\ \mu\text{s}) \quad (3)$$

In general, choose first the desired dead time: this will tell you if you want to run in parallel or non parallel mode, although most likely it is parallel mode. Then, choose the maximum frame rate you want to aim, not exceeding what you aim for not to increase the noise. In 4 and 8 bit modes it makes no sense to run nonparallel as the exposure time is too small compared to the readout time.

6.1.1 4 and 8 bit mode

In `parallel` mode, the minimum time between frames is due to the time required to latch the values of the counter with capacitors. These values are determined in firmware and they can be estimated as:

$$\text{time between frames, parallel} = 4\mu s \cdot (clkdivider + 1) \quad (4)$$

This time is independent on the `dr`.

In `nonparallel` mode, it is easily possible to calculate the required asic readout time. Indeed a block of (8*256) pixels are readout, the bits pixel are the `dr` and the speed of readout is 5ns/bit *(`clkdivider`+1) :

$$\text{asics readout time} = 5ns/bit \cdot 2^{(clkdivider+1)} \cdot dr \cdot (8*256) + 4\mu s \cdot (clkdivider+1) \quad (5)$$

While we expose the next frame, we still need to readout the previous frame, so we need to guarantee that the period is large enough at least to readout the frame. So the maximum frame rate has to be 1/(asic readout time). The minimum period has to be equal to the asic readout time.

6.1.2 16 bit mode

A similar situation happens in 16 bit mode, where this is more complicated because of three things:

1. The chip actual `dr` is 12 bit
2. The chip is readout as 12-bit/pixel, but the FEB inflates the pixel values to 16-bits when it passes to the BEB. This means that effectively the FEB to BEB connection limits the data throughput in the same way as if the `dr` of the chip would really be 16 bits.
3. While in 4 and 8 bit mode it makes no sense to run in `nonparallel` mode as the exptime/dead time ratio would be not advantageous, in 16 bit mode, one can choose how to run more freely.

If we are in parallel mode, the dead time between frames, is also here described by equation 4. If we are in `nonparallel` mode, the dead time between frames is defined by 5 ONLY for `clkdivider` 1 and 2. So the maximum frame rate has to be 1/(chip readout time) in this case. Only for `clkdivider` 0 we hit some limitation in the bandwidth of The FEB → BEB connection. In this case, the maximum frame rate is lowered compared to what expected.

6.1.3 32 bit mode

The autosumming mode of Eiger is the intended for long exposure times (frame rate of order of 100Hz, PILATUS like). A single acquisition is broken down into many smaller 12-bit acquisitions, each of a `subexptime` of 2.621440 ms by default. Normally, this is a good default value to sustain an intensity of 10^6 photons/pixel/s with no saturation. To change the value of `subexptime` see section 8.

The time between 12-bit subframes are listed in table 5.

The exposure time brokend up rounding up to the full next complete subframe that can be started. The number of subframes composing

dr	clkdivider	flags	t difference between subframes(μs)	max internal subframe rate (kHz)	maximum frame rate (Hz)
32	2	parallel	12	2	170
32	2	nonparallel	504	< 2	160

Table 5: Timing for the 32bit case. The maximum frame rate has been computed assuming 2 subframes of default `subexptime` of 2.62144 ms.

a single 32bit acquisition can be calculated as:

$$\#\text{ subframes} = (int)(\frac{\text{exptime (s)}}{\text{subexptime (s)} + \text{difference between frames (s)}} + 0.5) \tag{6}$$

This also means that `exptime`<`subexptime` will be rounded to`subexptime`. If you want shorter acquisitions, either reduce the `subexptime` or switch two 16-bit mode (you can always sum offline if needed).

From release 4.0.0, an extra `flag overflow/nooverflow` is added, with `nooverflow` default:

- `nooverflow`: the internal 12-bit result is summed, even if there was saturation of the 12-bit counter (4095) in any of the subframes. Note that if there is saturation for every subframe, you might get as a result a value of the counter equal to (4095$\times$ number of subframes), which you need to correctly identify. On the other hand if the saturation occurred only one time, you will get something "close" to the real number.
- `overflow`: In this case, even if a pixel saturate only 1 time in a subframe, you will be returned as a value of the counter for that pixel: $2^{32} - 1$, i.e. 4294967295.

The UDP header will contain, after you receive the data, the effective number of subframe per image (see section 11.1) as "SubFrame Num or Exp Time", i.e. the number of subframes recorded (32 bit eiger). The effective time the detector has recorded data can be computed as:

$$\text{effective exptime} = (\text{subexptime}) \cdot (\#\text{ subframes}) \tag{7}$$

From release 4.0.0, a configurable extra time difference between subframes can be introduced for the parallel mode, so that some noise appearing in detectors at low threshold can be removed. This is obtained through the `subdeadtime`. You need to add values specific for your detector (ask the detector group). Typically, values between 15-40 μs should be used (for the 9M it is currently 200 μs due to a noisier module).

7 External triggering options

The detector can be setup such to receive external triggers. Connect a LEMO signal to the TRIGGER IN connector in the Power Distribution Board (see

Figure 4: **Trigger INPUT** (looking at a single module from the back, top) is the **rightmost, down**.

Fig.). The logic 0 for the board is passed by low level 0−0.7 V, the logic 1 is passed to the board with a signal between 1.2−5 V. Eiger is 50 Ω terminated. By default the positive polarity is used (negative should not be passed to the board).

```
sls_detector_put 0-timing [auto/trigger/burst_trigger/gating]
sls_detector_put 0-frames x
sls_detector_put 0-cycles y
sls_detector_acquire 0-
```

No timeout is expected between the start of the acquisition and the arrival of the first trigger.

Here are the implemented options so far:

- `auto` is the software controlled acquisition (does not use triggers), where `exptime` and `period` have to be set. Set number of cycles (i.e. triggers) to 1 using `cycles`. Set number of frames using `frames`.
- `trigger` 1 frame taken for 1 trigger. Your `frames` needs to be 1 always, `cycles` can be changed and defines how many triggers are considered. `exptime` needs to be set. In the GUI this is called trigger exposure series.
- `burst_trigger` gets only 1 trigger, but allows to take many frames. With `frames` one can change the number of frames. `cycles` needs to be 1. `exptime` and `period` have to be set. In the gui it is called trigger readout.
- `gating` allows to get a frame only when the trigger pulse is gating. Note that in this case the exp time and period only depend on the gating signal. `cycles` allows to select how many gates to consider. Set number of frames to 1 using `frames`.

Hardware-wise, the ENABLE OUT signal outputs when the chips are really acquiring. This means that the single subframes will be output in 32 bit mode. The TRIGGER OUT outputs the sum-up-signal at the moment (which is useless). This will be changed in the future to output the envelop of the enable signal.

We are planning to change some functionality, i.e. unify the `trigger` and `burst` trigger modes and make both `frames` and `cycles` configurable at the same time.

8 Autosumming and rate corrections

In the case of autosumming mode, i.e, `dr 32`, the acquisition time (`exptime` is broken in as many subframes as they fit into the acquisition time minus all the subframes readout times. By default the `subexptime` is set to 2.621440 ms. This implies that 12 bit counter of EIGER will saturate when the rate is above or equal to 1.57 MHz/pixel. The minimum value is of order of 10 ns (although as explained values smaller than 500 μs do not make sense). The maximum value is 5.2 s.

The subframe length can be changed by the user by doing:

```
sls_detector_put 0-subexptime [time_in_s]
```

One needs to realize that the readout time, for each subframe is 10.5 μs if the detector is in parallel mode. 500 μs if the detector is in non parallel mode. Note that in `dr 32`, as the single frame readout from the chip is 500 μs, no `subexptime`<500 μs can be set in `parallel` mode. To have smaller `subexptime`, you need the `nonparallel` mode, although this will have a larger deadtime than the acquisition time.

Rate corrections are possible online (and the came procedure can be used offline) by creating a look-up table between the theoretically incident counter value c_i and the detected counter value c_d. In the EIGER on board server, this look-up table is generated assuming that the detected rate n_d can be modeled as a function of the incident rate n_i according to the paralyzable counter model:

$$n_d = n_i \cdot exp(-n_i \cdot \tau), \tag{8}$$

where τ represents an effective parameter for the dead time and the loss in efficiency. The look-up table is necessary as we are interested to obtain $c_i(c_d)$ and equation 8 is not invertible. One needs to notice that the paralyzable counter model to create a look-up tables applies only if photons arrive with a continuous pattern (like at the SLS). If photons are structured in fewer but intenser bunches, deviations may arise. This is the case for some operation modes at the ESRF. For those cases we are studying how to correct, probably from a simulated correction tables if an analytical curve cannot be found. **In the new calibration scheme, τ is given as a function of the energy. It**

is loaded from the trimbit files and interpolation between two trimbit files are performed. One needs to make sure the appropriate τ value is written in the trimbit files, then need to load the appropriate `settings` and `vthreshold` before.

Online rate corrections can be activated for `dr=32`. They are particularly useful in the autosumming mode as every single subframe is corrected before summing it. To correct for rate, the subframe duration has to be known to the correction algorithm. Rate corrections for `dr=16` will be activated as well in the next firmware release. To activate the rate corrections, one should do:

```
sls_detector_put 0-ratecorr [tauval_in_ns]
```

To deactivate:

```
sls_detector_put 0-ratecorr 0
```

Now to activate the rate corrections with the value written in the trimbit file or interpolated from there, once would do:

```
sls_detector_put 0-ratecorr -1
```

Every time either the rate corrections are activated, τ is changed or the subframe length is changed, then a new correction table is evaluated. Note that computing the correction table is time consuming.

9 Dependent parameters and limits

Here is a list of dependent parameters:

1. **dr** changes **clkdivider**: **dr** 16 → **clkdivider** 1. You can change it to (0, 1, 2); the frame rate changes accordingly to table 4. Setting the **dr** to 32 changes **clkdivider** to 2. Only way **dr** 32 can work.

Here is a list of "ignored" parameters, meaning that if the parameters are not what the detector expects, it will ignore them, but there is no guarantee that you get what you think you are asking:

1. **period**. Assuming that you set the correct **exptime** according to the table 4, the **period** to be used by the detector has to be >= **exptime**+readout time (table 4). Otherwise the detector will take data at the minimum possible period, which is **exptime**+readout times. **period** is not changed by the detector after the acquisition.

Here is a list of limits that should be checked:

1. If **dr** is 32 and **clkdivider** is not 2, whatever the detector gets out is wrong (the boards cannot properly keep up)

2. If the variable **frames** is greater than what the memory can store (table 2) and the frame rate exceed the continuos streaming (table 1), limits on the maximum number of images need to be implemented if the period is lower than the one listed in table 1. Check table 4 to see the different cases.

3. Running at a speed that does not support the frame rate you are asking: see table 4 to check if the frame rate (**period**) you are asking is compatible with the **clkdivider** you are asking.

4. Running at a readout time that does not support the frame rate you are asking. Check table 4 to check if the frame rate (**period**) you are asking is compatible with the **flags** you are asking.

5. The minimum allowed value for **exptime** should be 10 μs.

6. By default the **subexptime** is set to 2.621440 ms. Values smaller than 500 μs do not make sense. The maximum value is 5.2 s. This limits should be checked.

Here is a list of parameters that should be reset:

1. **resetframescaught** should be reset to zero after every acquisition taken with `receiver start`,`status start`,`receiver stop`. If the acquisition is taken with `sls_detector_acquire`, there is no need to reset this.

2. After changing the `timing` mode of the detector, one should reset to '1' the unused value, in that specific timing mode, between **frames** and **cycles**. See section 7 for how to use the timing. At the present moment the detector will acquire more frames than planned if the variable not used between **frames** and **cycles** is not reset. In future releases, the unused variable will be ignored. Still resetting is a good practice.

10 1Gb/s, 10Gb/s links

10.1 Checking the 1Gb/s, 10Gb/s physical links

LEDs on the backpanel board at the back of each half module signal:

- the 1Gb/s physical link is signaled by the most external LED (should be green). For top half modules is at the extreme left. For bottom half modules is at the extreme right.

- the 10Gb/s physical link is signaled by the second most external LED next to the 1Gb/s one (should be green).

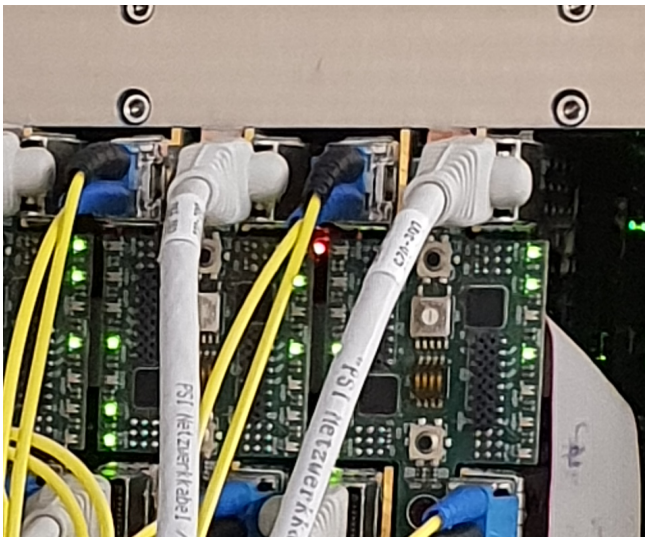

Figure 5: 1 and 10GB LEDs position.

10.2 Delays in sending for 1Gb/s, 10Gb/s, 10Gb flow control, receiver fifo

Extremely advanced options allow to:

- Activate the flow control for 10 Gb/s E (by default the 1 Gb/s E is always active and cannot be switched off:

  ```
  ./sls_detector_put flowcontrol_10g 1
  ```

- Delay the transmission of the left port. This delay option is useful in the case of many simultaneous receivers running, such that it reduces the throughput to the receivers all at the same time. To be used board by board (i.e `X:, Y:,etc..` with different units:

  ```
  ./sls_detector_put X:txndelay_left xxxx
  ```

- Transmission delay of the right port, same as above. The value here should be different from the left port to spread the transmission even more

  ```
  ./sls_detector_put X:txndelay_right yyyy
  ```

  As example:

  ```
  for X in $(seq 0 4); do ./sls_detector_put $X:txndelay_left $((X*100000)); done
  ```

```
./sls_detector_put $X:txndelay_right $((X*100000)); X=$((X+1)); done
```

- Set transmission delay of the entire frame. This is required as you want to finish sending the first frame to all receivers before starting sending the second frame to the receivers with shorter delay time. This value has to be greater than the maximum of the transmission delays of each port.

  ```
  ./sls_detector_put txndelay_frame zzzz
  ```

  In the example before, it would be: `zzzz`=4*100000+ 100000

- Readjust the size of the fifo of the receiver between listening and writing (useful when writing is limited)

  ```
  ./sls_detector_put rx_fifodepth xxxx
  ```

  `xxxx` is 100 images by default.

- Deactivate a half module (for ROI or debugging). Note that the MASTER module SHOULD NOT be deactivated:

  ```
  ./sls_detector_put X:activate 0
  ```

  where X is the half module you want to deactivate. The receiver at this point will return fake data (overflow) for this module. If you wish to eliminate the receiver overall for this module, then you need to run a configuration file where this module has been removed. To activate back a module, do:

  ```
  ./sls_detector_put X:activate 1
  ```

10.3 Setting up 10Gb correctly: experience so far

For configuring well the 10Gb card not to loose packets,

- MTU must be set up to 9000 (jumbo frames) on all the involved sides: detector, switch, server NIC
- you should set up static MAC address tables with separated VLANs

As root, also do:

```
ethtool -G xth1 rx 4096, ethtool -C xth1 rx-usecs 100
```

where `xth1` can be replaced with the correct 10Gb device. To minimise loosing packets, priorities are set better as root user, so have the receiver as root. To try to bypass being root, we trued something like this:

```
/etc/security/limits.conf  username  rtprio 99
```

Byte	0		1		2		3		4		5		6		7	
Bits	7…0		15…8		23…16		31…24		39…32		47…40		53…48		63…54	
	Frame Number (8 bytes)															
	SubFrame Num or ExpTime in 10ns steps (4 bytes)								Packet Number (4 bytes)							
	Bunch ID (8 bytes)															
	Time Code in 100ns steps (8 bytes)															
	Module ID (2 bytes)				X coordinate (2 bytes)				Y coordinate (2 bytes)				Z coordinate (2 bytes)			
	Debug (4 bytes)								Round Robin addr (2 bytes)				Det (1byte)		Hdr (1by)	

Frame Number	the frame ID this UDP packet belongs to
SubFrame Num or ExpTime in 10ns steps (EIGER)	On EIGER it is the subframe number in summing mode. For non summing mode this is 1.
SubFrame Num or ExpTime in 10ns steps (Non-EIGER)	Exposure time in 100ns steps
Packet Number	the packet ID within the frame
Bunch ID	the bunch ID when the image was taken, for EIGER 0
Time Code in 100ns steps	The time when the image was taken. This is not an absolute time value
Module ID	From which module of the detector the packet comes from
X/Y/Z coordinates	the X/Y/Z coordinate of the detector module
debug	debug information, should be 0 in non-debug firmware
Round Robin	pointer to the round robin address table in the detector
Det	Detector type (3 Eiger, 8 Jungfrau), see sls_receiver_defs.h, enum detectorType
Hdr	Header version, for now it's 1
Byte Order	1 is `01 00…00`, 256 is `00 01 00 00…00`

Figure 6: UDP header out of EIGER

but somehow it did not fully worked so we kept the trick of being root.

Very important is to activate the flow control in 10Gb (in 1Gb it is on by default and not configurable)

```
./sls_detector_put flowcontrol_10g 1
```

Set the transmission delays as explained in the manual.

It can help to increase the fifo size of the receiver to `rx_fifodepth` to 1000 images

```
./sls_detector_put rx_fifodepth 1000
```

One needs to keep into account that in 16 bit mode for 1 image we expect each slsReceiver to allocate 0.5MB. So for 1000 images, we expect 500MB memory for each receiver. This can be monitored in Linux with "top" or "free -m".

11 Offline processing and monitoring

11.1 Data out of the detector: UDP packets

The current UDP header format is described in figure 6.

11.2 Data out of the slsReceiver

For a module, the geometry of the ports are as in table 6: white the option `n:flippeddatax 1`, which flips in vertical the content of the module. By convection, we usually use `1:flippeddatax 1`, but one could flip the top instead.

0:rx_udpport 50011	0:rx_udpport2 50012
1:rx_udpport 50013	1:rx_udpport2 50014

Table 6: UDP port geometry for a single module, 4 UDP ports.

11.3 "raw" files

If you use the option of writing raw files, you will have a raw file for each UDP port (meaning most likely 2 chips), 4 files per module. In addition to the raw files, you will get also a "master" file, containing in ascii some detector general parameters and the explanation of how to interpret the data from the raw files.

The master file is named: `filename_master_0.raw` and for version "4.0.0" of the slsDetectorSoftware looks like:

```
Version                    : 2.0
Dynamic Range              : 32
Ten Giga                   : 1
Image Size                 : 524288 bytes
x                          : 512 pixels
y                          : 256 pixels
Max. Frames Per File       : 10000
Total Frames               : 1
Exptime (ns)               : 1000000000
SubExptime (ns)            : 2621440
SubPeriod(ns)              : 2621440
Period (ns)                : 1000000000
Timestamp                  : Mon Sep  3 09:07:05 2018


#Frame Header
Frame Number               : 8 bytes
SubFrame Number/ExpLength  : 4 bytes
Packet Number              : 4 bytes
Bunch ID                   : 8 bytes
Timestamp                  : 8 bytes
Module Id                  : 2 bytes
X Coordinate               : 2 bytes
Y Coordinate               : 2 bytes
Z Coordinate               : 2 bytes
Debug                      : 4 bytes
Round Robin Number         : 2 bytes
Detector Type              : 1 byte
Header Version             : 1 byte
Packets Caught Mask        : 64 bytes
```

Note that if one wants to reconstruct the real time the detector was acquiring

in 32 bit (autosumming mode), one would have to multiply the SubExptime (ns) for the SubFrame Number.

11.4 Offline image reconstruction

The offline image reconstruction`slsImageReconstruction` is not part of the package anymore. The code is still available doing
`git clone git@git.psi.ch:sls_detectors_software/sls_image_reconstruction.git slsImageReconstruction`.

Two possible conversions are possible: into **cbf** format or **hdf5** format. The detector writes 4 raw files per receiver. An offline image reconstruction executable has been written to collate the possible files together and produce output files. By default an interpolation between the values of the large pixels is performed. Gap pixels between modules are also inserted.

11.4.1 cbf

The cbf executable executable uses the CBFlib-0.9.5 library (downloaded from the web as it download some architecture dependent packages at installation).Edit the Makefile to correclty point at it.
At cSAXS, the CBFlib-0.9.5 has been compiled -such that the required packages are downloaded in /sls/X12SA/data/x12saop/EigerPackage/CBFlib-0.9.5.

To use it for a single module:

`cbfMaker [filename with dir]`

eg. `cbfMaker /scratch/run_63_d1_f000000000000_3.raw`

To use it for a 1.5 multi modules:

`cbfMaker [filename] [pixels x] [pixels y] ([singlemodulelongside_x] [start det])`

eg. `cbfMaker /scratch/run_63_d0_f000000000000_3.raw 3072 512 1 0`.
The `[singlemodulelongside_x] [option to interpolate gap pixels]` param are optional. Defaults are "1", the detector long side is on the x coordinate and start to reconstruct from module 0. The executables:

```
cbfMaker1.5M [file_name_with_dir]
cbfMakerOMNY [file_name_with_dir]
cbfMaker9M [file_name_with_dir]
```

contain the hardcoded geometry for the 1.5M (3 modules horizontal on the long side), the 1.5M OMNY geometry (3 modules next to each other on the long side) and for the 9M at cSAXS: 6(short side)$\times$3 (long side) modules.
Missing packets in a frame and border pixels ($\times$2 and $\times$4 are given with value -1 at the present time.

11.4.2 hdf5

In case of HDF5 output file, we rely on having the HDF5 1.10.1 library installed. Edit the Makefile to correclty point at it. Different compression methods are being tried so different external filters might be to be installed. This work is not finished yet.

11.5 Read temperatures/HV from boards

With an updated kernel on the linux boards (ask to the SLS detector group for specifications), it is possible to monitor the temperature on the boards:

```
temp_fpga   #gets the temperature of the fpga
temp_fpgaext   #gets the temperature close to the fpga
temp_10ge   #gets the temperature close to the 10GE
temp_dcdc   #gets the temperature close to the dc dc converter
temp_sodl   #gets the temperature close to the left so-dimm memory
temp_sodr   #gets the temperature close to the right so-dimm memory
temp_fpgafl   #gets the temperature of the left front end board fpga
temp_fpgafr   #gets the temperature of the right front end board fpga
```

You need to use the command specifying from which board you desire the temperature readings, for example:

```
./sls_detector_get 0:temp_fpga
./sls_detector_get 1:temp_fpga
```

In 500k–2M pixel systems there is a hardware temperature safety switch, which will cut power to the BEBs when reaching a too high temperature. For the 9M system, there is a temperature sensor read by the bchip100 PCU which will shutdown the detector when above a certain temperature.

The HV can also be set and read through the software:

```
./sls_detector_put vhighvoltage 150
./sls_detector_get vhighvoltage
```

Note that the get `vhighvoltage` would return the measured HV from the master module only. If getting the vhighvoltage for individual halfmodules, only the master will have a value different from -999.

A Kill the server, copy a new server, start the server

All the below operations are form a terminal and assume you login to the boards. Kill current server:

```
ssh root@bebxxx #password is root
killall eigerDetectorServer # kill server and stopserver
```

Copy a new version of the server (if necessary, otherwise skip it):

```
cd executables
scp user@pc:/path/eigerDetectorServerNewVersion .
chmod 777 eigerDetectorServerNewVersion
mv eigerDetectorServerNewVersion eigerDetectorServer
sync
```

Start the server again:

```
./eigerDetectorServer &
```

Note that the server appropriate for the software version used is located inside the package: `slsDetectorsPackage/serverBin/eigerDetectorServerxx.yy.`.

To copy the detector server on many boards, a script can be implemented on the lines of:

```
for i in beb111 beb070;
do ssh root@$i killall eigerDetectorServer;
scp eigerDetectorServer root@$i:~/executables/eigerDetectorServer ;
ssh root@$i sync; done
```

B Loading firmware bitfiles

As a new procedure, the first thing to do is to kill the server on the boards, copy the new one there without starting it. Note taht failure to do this step before may cause the linux on the baords to crash and not being able to ping it (this if the registers between the old and new firmware change).

This is teh procedure from a terminal;

```
for i in beb111 beb070;
do ssh root@$i killall eigerDetectorServer;
scp eigerDetectorServer root@$i:~/executables/eigerDetectorServer ;
ssh root@$i sync; done
```

A **bcp** executable (which needs **tftp** installed on the PC, is needed.

1. Manual way: you need to press something on the detector. To program bitfiles (firmware files), do a hard reset with a pin/thin stuff in the holes at the very back of the module. They are between the top 7 LED and the bottom 1 and opposite for the other side. Push hard till all LEDs are alternating green and red.

2. Software way (possible only if you have the correct programs copied on your board. If not, as the sls detector group).

   ```
   ssh root@bebxxx
   cd executables
   ./boot_recovery
   ```

In both case, after booting, only the central LED should be on green and red alternating.

From a terminal, do:

```
nc -p 3000 -u bebxxx 3000
```

where `xxx` is the board number. It is enough top monitor with `nc` only one board. Pres enter twice (till you see a prompt with the board hostname printed) and keep this terminal to monitor. It takes a bit of time to load the bitfiles, but the terminal tells you.
From another terminal you do:

```
./bcp feb_left.bit bebxxx:/febl
sleep 300; #or till the screen over netcat has told you Successful
./bcp feb_right.bit bebxxx:/febr
sleep 300; #or till the screen over netcat has told you Successful
./bcp download.bit bebxxx:/fw0
sleep 300; #or till the screen over netcat has told you Successful
```

If you need to program a new kernel (only needed when told to do so):

```
 ./bcp kernel_local bebxxx:/kernel
sleep 300; #or till the screen over netcat has told you Successful
```

do the same for the other boards. You can program in parallel many boards, but you cannot load two bitfiles on the same board till loading and copying one process has finished. So load all left febs together, then proceed to the right febs, then the bebs. Power off completely everything. Power it on.

C Pulsing the detector

There are two ways to pulse the detector:

- **Pulse digitally:** when you are interested to the output readout and do not care about the analog response from the pixels:

  ```
  sls_detector_put vthreshold 4000
  sls_detector_put vtr 4000
  sls_detector_put pulsechip N #to pulse N
  sls_detector_put pulsechip -1 #to get out of testing mode
  ```

  Note that the answer will be $2 \cdot \mathtt{N} + 2$ in this case.

- **Pulse analogically:** You want to really check the analogical part of the detector, not just the readout.

  ```
  sls_detector_put vcall 3600
  sls_detector_put vthreshold 1700
  ```

```
sls_detector_put vrf 3100
for i in $(seq 0 7) ;
do px=$((-255+i));
sls_detector_put  pulse 0 $px 0;
for j in $(seq 0 255) ; do
sls_detector_put pulsenmove N 0 1;
done;
done;
sls_detector_put resmat 0
sls_detector_acquire
```

You read `N` in every pixel if you are setup correctly.

D Load a noise pattern with shape

For debug purposes, we have created a noise pattern with a shape. If you reconstruct correctly your image, you should be able to read ".EIGER" in the same direction for both the top and bottom in normal human readable orientation. To load the special noise file look at `settingsdir/eiger/standard/eigernoise.sn0xx` in the package.

```
sls_detector_put trimbits ../settingsdir/eiger/standard/eigernoise
```

E Troubleshooting

E.1 Cannot successfully finish an acquisition

E.1.1 Only master module return from acquisition

When no packets are received AND detector states in 'running status'. Widest list of causes. Query the status of each half module till the maximum number `N`, `for i in $(seq 0 N); do sls_detector_get $i:status; done`, to check if there are half modules that are still running.

If only the master modules return but ALL the other half modules do not:

- FEB LED 1 and or 3 become red while trying to acquire an image: reconnect or change the DDR2 memories. Technically it is a FIFO problem to communicate the data to the rest of the chain.
- It can be that the master cable is not connected, check.
- It can be that the synchronization cable is not connected or the termination board at the synchronization does not work. Check.

E.1.2 A few modules do not return from acquisition

If only a few modules are still running but the others return, it is a real problem with a backend board or a synchronization bug. If you can, ssh into the board, kill and start the eigerDetectorServer again (see Section A for how to do this). Keep the terminal with the output from the eigerDetectorServer and repeat the acquisition.

- Check if the acquisition returned from the server or not. In case seek help from the SLSDetectorGroup.
- In the server you read something along the lines of "cannot read top right address". It is communication between the front and backend board. Or FEB FPGA is not programmed. Try to program again FPGA, and make sure you program FPGA bit files 70x, if you have 70x FPGAs, or 30x, if you have 30x FPGAs. If still fails, tell the SLSDetectorGroup as it could be a hardware permanent failure.

E.2 No packets (or very little) are received

In both cases running **wireshark** set to receive UDP packets on the ethernet interface of the receiver (filter the UDPport>=xxxx, where xxxx is written in the configuration file) can help you understanding if NO packets are seen or some packets are seen. You have to set the buffer size of the receiving device in wireshark to 100Mbyte minimum. If no packets are received, check that your receiving interface and detector UDPIPs are correct (if in 10Gb). Most of the time in this case it is a basic configuration problem. It can help looking at the receiver output, shown in an example here:

```
Missing Packets          : 224064
Complete Frames          : 3499
Last Frame Caught        : 3499
```

The `Last Frame Caught`, meaning the packet from the last frame that was sent out by the detector, can help in understanding the problem:

1. If some packets are received, but not all, it could be a network optimization problem. In this case, the `Last Frame Caught` will be a value close to the expected number of frames with missing frames distributed over the whole frame range. In this case:
   - For receiving data over 1Gb, the switch must have FLOW CONTROL enabled
   - If using 10GbE, check that the 10Gb link is active on the backpanel board. Then refer to Section 10.3 to see how to configure the 10Gb ports on the receiving machine correctly.

2. If the `Last Frame Caught` value is much lower than the expected frames and you are missing a bunch of frames from a point onwards, and you are using `receiver start, status start`: then it can be that you are stopping the receiver too early. In particular when you are using `delay` it might be that there is some time between when the detector is already done and in `idle` state but the receiver is still receiving data. Check with `./sls_detector_get framescaught` if the receiver is already done before doing `./sls_detector_put receiver stop`.

3. If the `Last Frame Caught` value is much lower than the expected frames and you are missing a bunch of frames from a point onwards and you are running at a higher frame rate than the continuous framerate (see table 1) with more images than the size of the memory (see table 2). It might be that you are running out of memory to store images. There is no protection for this. see point E.3

E.3 'Got Frame Number Zero from Firmware'

In this case, you have run out of memory size (see table 2 for the size) on the boards so you are trying to store on the DDR2 memories more images that they can contain and the network is not fast enough to send everything out from the 10GbE. So if you see:

```
Got Frame Number Zero from Firmware. Discarding Packet
```

it means that you run out of memory at the previous acquisition. The cure is taking 2 or 3 SINGLE images in a raw to clear out the memories.

E.4 The module seems dead, no lights on BEBs, no IP addresses

- Check the 2 fuses on the power distribution board. If one of the fuses is in shortcuircuit, then exchange it. Nominal values are 7 A and 5 A. Old modules with 5 A and 3 A could trip.

- The module is not properly cooled and the temperature safety switch has killed the power to the backend boards.

E.5 The module seems powered but no IP addresses

If the 1G LED (see Section 10.1) on the backpanel board is not green:

- Check that the 1Gb cable is plugged in.

- Check that there is a DCHP server assigning IP addresses to the board.

- The IP address is assigned only at booting up of the boards. Try to reboot in case the board booted before it could have an IP address.

- Check that you did not run out of IP addresses

Check that the board is not in recovery mode (i.e. the central LED on the back is stable green, see Fig 5). In this case reboot the board with the soft reset or power cycle it.

If the 1Gb LED on the backpanel board is green (see Section 10.1):

- Check that the IP address has been refreshed on the PC you are trying to communicate to the detector from. Run on the PC as root the following command to update the DNS cache: **nscd -i hosts**

E.6 Receiver cannot open socket

It is connected to the TCPport which the receiver uses:

- The port is already in use by the same receiver already opened somewhere or by another process: check with **ps -uxc** your processes
- In rare cases, it might be that the TCP port crashes. To find out which process uses the TCPPOrt do: **netstat -nlp — grep xxxx**, where xxxx is the tcpport number. To display open ports and established TCP connections, enter: **netstat -vatn**. Kill the process.

E.7 The client ignores the commands

Make sure that in the configuration file you do not have `lock 1` activated, as this will let only one username from one IP address talk to the detector. To deactivate it, you need to run `lock 0` from the client session where you locked it.

E.8 Zmq socket is blocked

It is connected to the TCPport which is used. In rare cases, it might be that the TCP port crashes. To find out which process uses the TCPPOrt do: **netstat -nlp — grep xxxx**, where xxxx is the tcpport number. To display open ports and established TCP connections, enter: **netstat -vatn**. Kill the process.

E.9 Client has shmget error

Note that occasionally if there is a shared memory of a different size (from an older software version), it will return also a line like this:

```
*** shmget error (server) ***-1
```

This needs to be cleaned with `ipcs -m` and then `ipcrm -M xxx`, where xxx are the keys with nattch 0. Alternative in the main slsDetectorFolder there is a script that can be used as `sh cleansharedmemory.sh`. Note that you need to run the script with the account of the client user, as the shared memory belongs to the client. It is good procedure to implement an automatic cleanup of the shared memory if the client user changes often.

E.10 Client has shared memory iusses

The shared memory from software version 4.0.0 creates shared memory segments in /dev/shm/. You can look at them and cancel them directly. Note that this is still user dependent. Environment variable SLSDETNAME can be set for using 2 different detectors from the same client pc. One needs a different multi detector id if the SLSDETNAME is different for both consoles.

E.11 Measure the HV

For every system:

- Software-wise measure it (now the software returns the measured value), with `sls_detector_get vhighvoltage`. The returned value is the HV (for proper Eiger setting is approximately 150 V) if it is correctly set. If two master modules are presents (multi systems), the average is returned (still to be tested). If one asks for the individual n half module bias voltage through `sls_detector_get n:vhighvoltage`, if the n module is a master, the actual voltage will be returned. If it is a slave, -999 will be returned.
- Hardware-wise (opening the detector) measure value of HV on C14 on the power distribution board. Check also that the small HV connector cable is really connected.

The 2M system at ESRF has a HV enable signal that needs to be shortcut in order to overwrite vacuum protections (when not in vacuum). The 1.5M for OMNY has a relay system that enables HV only when the vacuum is good. For both systems, it makes sense not to set the HV while running the configuration file but set it at a later stage when sure about the vacuum.

E.12 The image now has a vertical line

Check if the vertical line has a length of 256 pixels and a width of 8 columns. In this case it is a dataline being bad. It can be either a wirebond problem or a frontend board problem. try to read the FEB temperature (see Section **??**) and report the problem to the SLSDetector group. Most likely it will be a long term fix by checking the hardware.

E.13 The image now has more vertical lines

If you see strange lines in vertical occurring at period patterns, it is a memory problem. The pattern is 4 columns periodic in 16 bit mode, 8 columns periodic in 8 bit mode and 2 columns periodic in 32 bit mode. Try to switch on and off (sometimes it is a strange initialization problem).

E.14 ssh to the boards takes long

Depending on your network setup, to speed up the ssh to the boards from a pc with internal dhcp server running: **iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE; echo "1" ¿ /proc/sys/net/ipv4/ip_forward**, where eth1 has to be the 1Gb network device on the pc

E.15 Check firmware version installed on BEB

You can either ask in the client as described in section 2, or login to the boards directly. Follow some steps described in Section A.

```
ssh root@bebxxx #password is root
killall eigerDetectorServer # kill server and stopserver
cd executables/
./eigerDetectorServer&
```

Scroll up in the terminal till you find `Firmware Version:  xx`

E.16 Check if half-module is a master, a slave, a top or a bottom

Follow some steps described in Section A.

```
ssh root@bebxxx #password is root
killall eigerDetectorServer # kill server and stopserver
cd executables/
./eigerDetectorServer&
```

Scroll up in the terminal till you find:
********** TOP/BOTTOM **********
********** MASTER/SLAVE **********
********** NORMAL/SPECIAL **********

E.17 'Cannot connect to socket'

This error is typically due to the detector server not running. For why, see section E.18.

E.18 Detector server is not running

The detector server could not be running: either the detector was powered off, or it powered off itself due to too high temperature or, in the case of the 9M, if the waterflow sensor detected no flux and powered it off (the chiller stops occasionally as cSAXS).

If the powering and the temperature are OK, instead, it can be that the firmware version is incompatible to the server version and/or the client software version. So check the consistency of firmware/software/server versions.

E.19 'Acquire has already started' error message

If you see the client returning the following error message:
"Acquire has already started. If previous acquisition terminated unexpectedly, reset busy flag to restart.(sls_detector_put busy 0)"
You need to run the command:

```
./sls_detector_put busy 0
```

E.20 There is noise running the detector in 32-bit

If you are running the detector in 32-bit (autosumming), there might be some noise, particularly at lower thereshold energies. This is due to the fact that the analog part of the chips require some latency time to settle which is larger than the readout time. It is possible to run the detector only in `parallel` or `nonparallel` mode, respectively with readout times between frames of 12 μs and 504 μs. If you switch `flags` to non `nonparallel` mode you will give enough time for the signals to settle. From release 4.0.0, there is a configurable delay that can be set through the `subdeadtime` variable, such that you can remain with the `parallel` flag, but can obtain a configurable dead time between frames. Ask the SLS detector group for an appropriate dead time for your detector, but typically a dead time of 20-50 μs should be enough. Note that this `subdeadtime` need to include the 12 μs minimum readout time, so it has to be larger than 12 μs to do anything.

E.21 There is noise running the detector at high frame rate(4,8,16 bit)

If are running in `parallel` mode, in particular at low thereshold energies, you might encounter some noise. The reason is that the analog part of the chips require some latency time to settle which is larger than the readout time.

1. You can lower the frame rate and relax requirements on period: At low frame rate, you normally leave enough time between the end of the acquisition and the starting of the next, so you should not see this effect. In any case setting a `period`=`exptime`+readout time from Table 3 +extra 20μs cures the problem. The 20μs could also be 10 μs, they are very hardware dependent.

2. The frame rate requirement are stringent (as for time resolved measurements): the only option here is to reduce the `exptime` to let the extra 20 μs (or 10) μs. The `period` remains the same.

F Client checks - command line

Guide on returned strings:

1. `sls_detector_get free`

   Returns a list of shared memories cleaned (variable number depending on detector):

   ```
   Shared memory 273612805 deleted
   Shared memory 276922374 deleted
   Shared memory 270270468 deleted
   free freed
   ```

   Note that occasionally if there is a shared memory of a different size (from an older software version), it will return also a line like this:

   ```
   *** shmget error (server) ***-1
   ```

   This needs to be cleaned with `ipcs -m` and then `ipcrm -M xxx`, where xxx are the keys with nattch 0.

2. `sls_detector_get settings`
   `settings standard`

   `standard` is only if correct. `undefined` or anything else is wrong.

3. `sls_detector_get threshold`
   `threshold xxxx`

   Returns a string (xxxx) that can be interpreted as the threshold in eV. If it fails to set it, returns the last threshold it was set (which the detector still has). If settings are not defined or different trimbits are chosen, it will return "undefined".

4. `sls_detector_get fname`
   `fname string`

5. `sls_detector_get exptime`
   `exptime number`

   where `number` is a string to be interpreted as a float in (s).

6. `sls_detector_get period`
   `period number`

   where `nuymber` is a string to be interpreted as a float in (s).

7. `sls_detector_get frames`
   `frames number`

   where `number` is a string to be interpreted as an integer.

8. `sls_detector_get cycles`
   `cycles number`

   where `number` is a string to be interpreted as an integer.

9. `sls_detector_get status`
   `status string`

   where `string` can be `idle` or `running`.

10. `sls_detector_get index`
    `status number`

    where `number` is a string to be interpreted as an integer.

11. `sls_detector_get dr`
    `dr number`

    where `number` is a string that should be interpreted as an integer (4/8/16/32).

12. `sls_detector_get clkdivider`
    `clkdivider number`

    where `number` is a string that should be interpreted as an integer (0/1/2/3).

13. `sls_detector_get flags`
    `flags string1 string2`

    where `string1` is a string should be always `continous` and `string2` can be either `nonparallel` or `parallel`.

14. `sls_detector_get timing`
    `timing string`

    where `string` is a string which can be `auto/trigger/burst_trigger/gating`.

15. `sls_detector_get enablefwrite`
    `enablefwrite number`

    where `number` is a string which should be interpreted as an integer "0" or "1".

16. `sls_detector_get framescaught`
    `framescaught number`

    where `number` is a string which should be interpreted as an integer of the complete frames got by the receiver.

17. `sls_detector_get frameindex`
    `frameindex number`

    where `number` is a string which should be interpreted as an integer of the last frame number read from firmware. It comes from the receiver, though and reset after every acquisition series.

18. `sls_detector_get subexptime`
    `subexptime number`

    where `number` is a string that should be interpreted as a float in s. The default value is 0.002621440.

19. `sls_detector_get ratecorr`
    `ratecorr number`

    where `number` is a string that should be interpreted as a float in s. 0.000000 means correction off. Values above zero are the value of τ in ns.

20. `sls_detector_get vhighvoltage`
    `vhighvoltage number`

    where `number` is a string that should be interpreted as an int and for proper Eiger setting is approximately 150 V if it is correctly set. If two master modules are presents (multi systems), the average is returned (still to be tested). If one asks for the individual n half module bias voltage through `sls_detector_get n:vhighvoltage`, if the n module is a master, the actual voltage will be returned. If it is a slave, -999 will be returned.

21. `sls_detector_get busy`
    `busy number`

    where `number` is a string that should be interpreted as an int for 0/1 meaning no/yes. This command tells if the sharedmemory has in memory that an acquisition has been started or not. It should allows to use the non blocking acquire, regardless of any delay to the detector getting into 'running' mode.

G Complete data out rate tables

In table 7 is a list of all the readout times in the different configurations.

Table 7 shows the bandwidth of data transferring between the FEB and BEB and of the DDR2 memory access. the GTX lanes are only capable of 25.6 Gbit/s. This limits the 12/16 bit frame rate. The 2×DDR2 memories have a bandwidth or 2·25.6 Gb/s=51.2 Gb/s. Due to this memory access bandwidth, the 32 bit autosumming mode can only run in `clkdivider` 2.

dr	clkdivider	flags	readout t(μs)	max frame rate (kHz)	min period (μs)	max imgs (nominal/our network)
4	0	parallel	3.4	22	44	30k/50k
4	0	nonparallel	44	21	49	30k/50k
4	1	parallel	6	10.5	92	30k/100k
4	1	nonparallel	88.7	10.5	93	30k/100k
4	2	parallel	11.2	5.4	197	infinite
4	2	nonparallel	176.5	5.4	180	infinite
8	0	parallel	3.4	11.1	89	15k/24k
8	0	nonparallel	85.7	11.1	91	15k/24k
8	1	parallel	6.1	5.7	181	15k/52k
8	1	nonparallel	170.5	5.7	175	15k/52k
8	2	parallel	11.2	2.9	342	infinite
8	2	nonparallel	340.3	2.9	344	infinite
16	0	parallel	3.4	6.1	164	8k/12k
16	0	nonparallel	126	5.6	179	8k/23k
16	1	parallel	6.1	3.9	257	8k/28k
16	1	nonparallel	255	3.3	303	infinite
16	2	parallel	11	1.9	526	infinite
16	2	nonparallel	504	1.8	555	infinite
32	2	parallel	11	2		
32	2	nonparallel	504	< 2		

Table 7: Readout settings. The min exptime possible is $5-10$ μs. This is due to the time to pass the pixel enable signal in the whole chip.

	GTX [Gb/s] (FEB->BEB)	Frame Rate [kHz]	DDR2 Memory [Gb/s]	Net [Gb/s]
Max Rate	25.6		51.2	10
4Bit	24	22	51.2 > 24FEB+10NET	10
8Bit	24	11	51.2 > 24FEB+10NET	10
12/16Bit	32	5.9	51.2 > 25.6FEB+10NET	10
Summing (32Bit)	-	2	51.2 << 51.2FEB+51.2MEM +10NET	10

Figure 7: Transmission bandwidth for the FEB →BEB transfer (second column) and the DDR2 memories (fourth column).