EIGER- short manual

September 7, 2017

Contents

1 Usage

1.1 Mandatory setup - Hardware

An EIGER single module (500 kpixels) needs:

- A chilled (water+alcohol) at approximately 21 °C, which needs to dissipate 85 W. For the 9M, a special cooling liquid is required: 2/3 deionized water and 1/3 ESA Type 48.
- 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 1.1.1.
- 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.
- 2×10 Gb/s transceivers to optionally, receive data at high rate.

The equipment scales linearly with the number of modules. Figure 1 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.) 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 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**

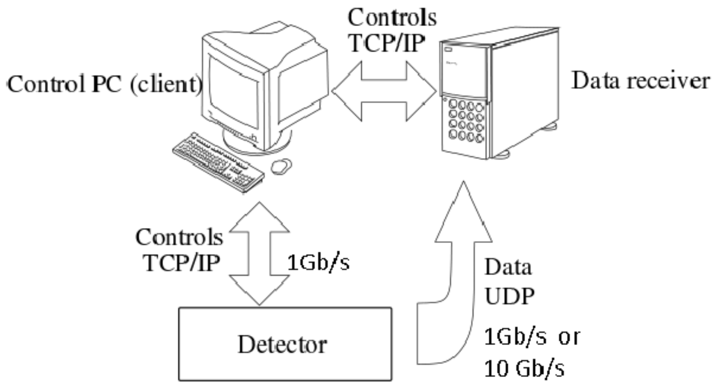


Figure 1: Communications protocol between the Client PC, the receiver PC and the detector.

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.**

1.1.1 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 temparature. 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

1.2 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/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. Note that in older version of the software `--mode 1` was used only for the "bottom" half module. Now, the receiver for the bottom is open without arguments anymore, but still in the configuration file one needs to write `n:flippeddatax 1`, where `n` 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.

1.3 Mandatory setup - Client

In the case of cSAXS, the detector software is installed on the x12sa-ed-1 machine: /sls/X12SA/data/x12saop/EigerPackage/slsDetectorsPackage

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
```

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
```

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` 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 referrered 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 section5 for how to set the detector.
- `flags continuous/storeinram`. Allows to take frame continuously or storing them on memory. Normally `continuous` should be used. Enabling the `stroreinram` mode allows you to obtain the maximum frame rate, but at the expenses to have to receive the data all at the end of the acquisition. Refer to readout timing specifications in section 5 for how to set the detector.

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 #[veryhighgain/highgain/lowgain/verylowgain] also poss
sls_detector_put 0-threshold energy_in_eV
```

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`.

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.

For EIGER, at the moment 5 settings are possible: `standard`, `lowgain`, `verylowgain`, `veryhighgain` and `highgain`. According to the setting chosen, one can reach different requirements (low noise or high rate). Refer to the settings requirements for your detector.
Notice that the option `settings standard/highgain/lowgain/veryhighgain/verylowgain` actually loads the trimbit files so it is time consuming. Only setting the `threshold` does not load trimbit files.

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

At cSAXS, the `settingsdir` and `caldir` are in /sls/X12SA/data/x12saop/EigerPackage/calibrations/

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:

- `sls_detector_put 0-receiver start`
- `sls_detector_put 0-status start`

You can poll the detector status using:

```
sls_detector_get 0-status
```

When the detector is `idle`, then you need to stop the receiver doing:

- `sls_detector_put 0-receiver stop`

You can then reset to zero the number of frames caught, if you desire:

- `sls_detector_put 0-resetframescaught 0`

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.

GbE	dynamic range	continuos maximum frame rate(Hz)
1	16	**256**
1	32	**128**
10	16	**2560**
10	32	**1280**
10	8	**5120**
10	4	**10240**

Table 1: Frame rate limits for the CONTINUOS streming out of images.

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).

5 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 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 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.

In table 3 is a list of all the readout times in the different configurations:

dr	clkdivider	mode	readout t(μs)	max frame rate (kHz)	max exptime (μs)	min period (μs)	max images
4	0	parallel	3.4	22	40	44	50k
4	0	nonparallel	44	21	3	49	50k
4	1	parallel	6	10.5	85	92	100k
4	1	nonparallel	88.7	10.5	3	93	100k
4	2	parallel	11.2	5.4	185	197	infinite
4	2	nonparallel	176.5	5.4	3	180	infinite
8	0	parallel	3.4	11.1	85	89	24k
8	0	nonparallel	85.7	11.1	3	91	24k
8	1	parallel	6.1	5.7	174	181	52k
8	1	nonparallel	170.5	5.7	3	175	52k
8	2	parallel	11.2	2.9	330	342	infinite
8	2	nonparallel	340.3	2.9	3	344	infinite
16	0	parallel	3.4	6	164	168	12k
16	0	nonparallel	126	3.4	164	295	23k
16	1	parallel	6.1	2.9	339	346	28k
16	1	nonparallel	255	1.7	339	592	infinite
16	2	parallel	11	1.5	66	78	infinite
16	2	nonparallel	504	0.85	7	512	infinite
32	2	parallel	11	2			
32	2	nonparallel	504	< 2			

Table 3: Readout settings. The min exptime possible is 3−5 μs.

As if you run too fast, the detector could become noisier, 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 `highgain` settings could not be used at 6 kHz. **We recommend to use the detector in 32 bit mode with `clkdivider 2`, `flags parallel`. We recommend to use the detector in 16 bit mode with `clkdivider 1`, `flags parallel`.** In general, choose first the desired dead time: this will tell you if you want to run in parallel or non parallel mode. Then, choose the maximum frame rate you want to aim, not exceeding what you aim for not to increase the noise.

6 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. 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, where `exptime` and `period` have to be set.
- `trigger` 1 frame taken for 1 trigger. You `frames` needs to be 1 always, `cycles` can be changed and defines how many triggers are considered. 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. 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.

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.

7 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:

$$n_d = n_i \cdot exp(-n_i \cdot \tau), \tag{1}$$

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 1 is not invertible. One needs to notice that the paralizable 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.

8 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.

9 1Gb/s, 10Gb/s links

9.1 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)
- the 10Gb/s physical link is signaled by the second most external LED next to the 1Gb/s one (should be green)

9.2 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
  ```

9.3 Setting up 10Gb correctly: experience so far

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

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)

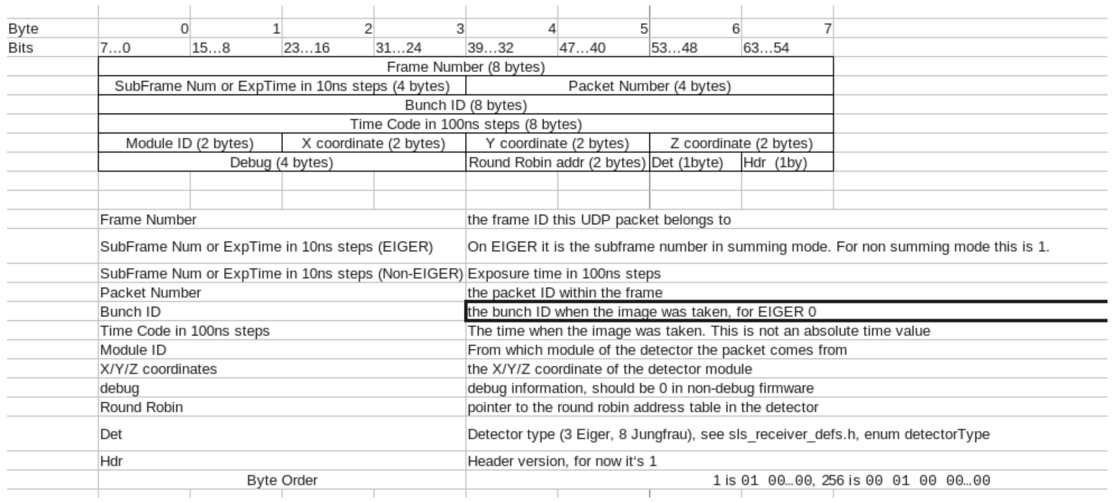
```
./sls_detector_put flowcontrol_10g 1
```

Set the transmission delays as explained in the manual.

10 Offline processing and monitoring

10.1 Data out of the detector: UDP packets

The current UDP header format is described in figure 2.



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 2: UDP header out of EIGER

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

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

10.2 Data out of the slsReceiver

For a module, the geometry of the ports are as in table 4: 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.

10.3 “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 “3.0” of the slsDetectorSoftware looks like:

```
Version            : 1.0
Dynamic Range      : 16
Ten Giga           : 1
Image Size         : 262144 bytes
x                  : 512 pixels
y                  : 256 pixels
Total Frames       : 1
```

```
Exptime (ns)       : 1000000000
SubExptime (ns)    : 2621440
Period (ns)        : 1000000000
Timestamp          : Thu Aug 17 10:55:19 2017


#Frame Header
Frame Number       : 8 bytes
SubFrame Number    : 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
```

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.

10.4 Offline image reconstruction

The offline image reconstruction is in `slsDetectorsPackage/slsImageReconstruction`.

The detector writes a raw file per receiver. An offline image reconstruction executable has been written to collate the possible files together and produce cbf files. The executable uses the CBFlib-0.9.5 library (downloaded from the web as it download some architecture dependent packages at installation).
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]` and `[start det]` param are optional. Defaults are “1”, the detector long side is on the x coordinate and start to reconstruct from module 0. The executables:

```
bcfMaker1.5M [file_name_with_dir]
bcfMaker9M [file_name_with_dir]
```

contain the hardcoded geometry for the 1.5M (3 modules horizontal on the long side) and for the 9M at cSAXS: 6(short side)×3 (long side) modules.
Missing packets in a frame and border pixels (×2 and ×4 are given with value −1 at the present time.

It is important to know, that the pixels at the edge between 2 chips count more as double size. We can virtually introduced 1 virtual pixel per double larger pixel, so to have an even number of counts everywhere. Virtual pixels (not filled) between module gaps are also inserted.

```
   GapPixelsBetweenChips_x = 2;
   GapPixelsBetweenChips_y = 2;
   GapPixelsBetweenModules_x = 8;
   GapPixelsBetweenModules_y = 36;
```

10.5 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 &
```

B Loading firmware bitfiles

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 one 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_p 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
sls_detector_put vthreshold 4000
sls_detector_put vtr 4000
```

E Running the (9M at cSAXS. For now)

- login as `x12saop@xbl-daq-27`
- `setup_eiger` #loads environmental variables and brings you to the right directory to execute commands
- slsReceiverScript3 1991 36 # from one shell.. opens 36 receivers
- p config ../../eiger_9m_10gb_xbl-daq-27_withbottom.config