Mythen v3.0 manual

August 24, 2018

Chapter 1

Installation and upgrades

The new MYTHEN software is intended to control the MCS mythen boards either by using a command line interface (text client) or by using with a graphical user interface (GUI).

Here you can find in brief the main things you need to know in order to start working with your detector.

1.1 The software package

The actual software for the Mythen II system (MCS1 to MCS24) runs on 32 bit Scientific Linux machines (SLC5 tested, gcc 4.1.2 but it should not be critical).

The complete software package is composed of several programs which can be instaleld (or locally compiled) depending on the needs:

- The **slsDetector shared and static libraries** which are necessary for all user interfaces and can be simply used for implementig custom detector drivers;
- The **command line interface (slsDetectorClient) sls_detector_put, sls_detector_get, sls_detector_acquire** which is provided to communicate with the detectors;
- A **virtual server mythenServer** which can be used to simulate the behavior of the detector for what concerns the communication in case the detector is not online or is in use.

1.2 Requirements

For installing the slsDetector shared and static libraries and the slsDetectorClient software, any Linux installation with a working gcc should be fine.

1.3 Compilation

If you simply want to install the software in the working directory you can:

- `make lib` compile slsDetector library
- `make slsDetectorClient` compile slsDetectorClient package
- `make all` compile slsDetector libraries, the slsDetectorClient package
- `make clean` remove object files and executables
- `make help` lists possible targets

To be able to run the slsDetectorClient commands, add their location to your path.

1.4 Building

To install the software you should first configure some enviroment variables by executing:

```
> source configure
```

(NOT `>./configure` otherwise the enviroment variables will not be available for the `make` command). This allows you to configure:

- **INSTALLROOT** Directory where you want to install the software. Defaults to /usr/local/
- **BINDIR** Directory where you want to install the binaries. Defaults to bin/
- **INCDIR** Directory where you want to pute the header files. Defaults to include/slsdetector/
- **LIBDIR** Directory where you want to install the libraries. Defaults to lib/
- **DOCDIR** Directory where you want to copy the documentation. Defaults to share/doc/

To build you can:

- `make install_lib` install detector library and include files"
- `make install_client` install slsDetectorClient
- `make install` install library, include files and mythenClient"
- `make install_libdoc` install library documentation
- `make install_clientdoc` install mythenClient documentation
- `make install_doc` install all documentation
- `make help` lists possible targets

1.5 Detector upgrade

The upgrade of the detector consists in both the upgrade of the communication software and of the firmware.

To upgrade the firmware you need either a working version of the Altera Quartus software or of the Quartus programmer, which can easly be downloade from
`https://www.altera.com/download/programming/quartus2/pq2-index.jsp`
Normally installation of the software and of the driver for the USB-Blaster (provided together with the MYTHEN detector) are simpler under Windows.
Under Windows, the first time that you connect the USB-Blasterto one of your USB ports, you will be asked to install new hardware. Set the path to search for the driver to: `C:\altera\80sp1\qprogrammer\drivers\usb-blasterp` (where `C:\altera\80sp1\qprogrammer\` is assumed to be ther path where your Quartus version is installed).

1. After starting the Quartus programmer, click on Hardware Setup and in the "Currently selected hardware" window select USB-Blaster.
2. In the Mode combo box select "Active Serial Programming".
3. Plug the end of your USB-Blaster WITH THE ADAPTER PROVIDED in the connector ASMI on the MCS board taking care that pin1 corresponds to the one indexed and with the rectangualr pad.
4. Click on add file and from select the programming file provided when the upgrade has been reccomended.
5. Check "Program/Configure" and "Verify".
6. Push the start button and wait until the programming process is finished (progress bar top left).
7. In case the programmer gives you error messages, check the polarity of your cable (pin1 corresponds) and that you have selected the correct programming connector.

To upgrade the software on the detector board transfer the provided software by ftp to the MCS:

```
ftp  mymcs.mydomain.com
username: root
password: pass
cd /mnt/flash/root
put mythenDetectorServer
quit
```

If the /mnt/flash/root directory does not exist, create it before the transfer by telnetting to the MCS.
After pressing reset on the board, the board should reboot.
If the program does not correctly start either check by using the http interface that it is started by the inittab (check that the file `/mnt/etc/inittab` ends with the line `myid2:3:once:/mnt/flash/root/mythenDetectorServer`).
Otherwise make the program executable by telnetting to the MCS and executing: `chmod a+xrw  /mnt/flash/root/mythenDetectorServer`
After pressing reset on the board, the board should reboot and the acqusition program correctly start.

1.6 The trimbits and calibration files

In order to be able to properly operate your detector you need a directory where the trimbit files (needed to set the detector settings and eventually equalize the individual channel thresholds) which in the following will be named *trimdir* and a directory where the calibration files (needed to convert the threshold energy in DAC units) are stored which in the following will be named *caldir*. *trimdir* and *caldir* can even be the same directory, and an example of it is given in the software package by the example directory `trimbits`.
Since these directories are customized by producing trimbit files and calibration for each detector, make sure not to overwrite yours every time you upgrade the software.

trimdir should contain three subdirectories `standard`, `fast` and `highgain` containing respectively the trimfiles `standard.trim`, `fast.trim` and `highgain.trim` which contain the correct voltage settings for the detector although all the individual channel thresholds set to 0. The original files contained in the package should be used, infact in case of error the detector would not recognize the correct settings.
The default trimbit files for each file will be stored in the directory according to the settings with the name `noise.snxxx` where `xxx` is the module serial number.

caldir should contain three subdirectories `standard`, `fast` and `highgain` containing respectively the trimfiles `standard.cal`, `fast.cal` and `highgain.cal` which contain an average calibration of the modules for the diffrent settings. However this can different from the correct one for each individual module even of several kev and therefore it is very important to perform an energy calibration on a module basis (see section **??**).
The default calibration files for each file will be stored in the directory according to the settings with the name `calibration.snxxx` where `xxx` is the module serial number.

Chapter 2

slsDetectorClient

2.1 Introduction

This program is intended to control the MYTHEN detectors via command line interface.

To get all the possibilities of usage simply type:

sls_detector_acquire to readout the detector at full speed

sls_detector_put to set detector parameters

sls_detector_get to retrieve detector parameters

There are different ways for communicationg with your detector(s).

multiDetector is represented by a group of controllers which operate symultaneously with the same parameters. You can define several multiDetector systems and int this case you address them using different indexes. In this case the syntax will be `sls\_detector\_cmd i-` where cmd can be acquire, put, get and i is the index of the multiDetector entity (if omitted defaults to 0 - standard usage). Normally it is handy to use the multiDetector structure also in case of single detectors. However in some cases one cannot avoid using the slsDetector structure for detailed configuration (e.g. meaning of external signals or other flags)

slsDetector is represented by a single controller. You can define several multiDetector systems and int this case you address them using different indexes. In this case the syntax will be `sls\_detector\_cmd i:` where cmd can be acquire, put, get and i is the index of the slsDetector entity, which cannot be omitted. When creating the multiDetector structure, the indexes are automatically assigned to the detectors contained in it. You can retrieve the indexes relative to the slsDetector using: `sls\_detector\_get hostname:pos, sls\_detector\_get id:pos` whic will return the hostname in position pos of your multiDetector structure (pos=0 in case of single detectors) and its index.

2.2 Acquisition

mythen_acquire [id[-/:]]
the detector is started and the data are acquired, postprocessed and written to file according to the configuration

2.3 Detector setup

mythen_put [id[:/-]]var arg
is used to configure the detector parameter var e.g. mythen_put 0:exptime 1 sets the exposure time to 1 s

help i get help

config fname reads the configuration file specified and sets the values

parameters fname sets the detector parameters specified in the file

setup rootname reads the files specfied (and that could be created by get setup) and resets the complete detector configuration including flatfield corrections, badchannels, trimbits etc.

hostname name this is mandatory!!!! sets hostname (or IP adress)

online b b can be 0 or 1 and sets the detector in offline/online state. Must be used to restore communication if some socket called failed because the detector was not connected.

status s either start or stop

caldir path Sets path of the calibration files

trimdir path Sets path of the trim files

outdir path directory to which the files will be written by default

fname name filename to which the files will be written by default (to which file and position indexes will eventually be attached)

index i start index of the files (automatically incremented by the acquisition functions)

nmod n Sets number of detector modules

extsig:i mode Sets usage of the external digital signal i. mode can be: off, gate_in_active_high, gate_in_active_low, trigger_in_rising_edge, trigger_in_falling_edge, ro_trigger_in_rising_edge, ro_trigger_in_falling_edge, gate_out_active_high, gate_out_active_low, trigger_out_rising_edge, trigger_out_falling_edge, ro_trigger_out_rising_edge, ro_trigger_out_falling_edge

timing Sets the timing mode of the detector. Can be auto, gating (works only if at least one of the signals is configured as gate_in), trigger (works only if at least one of the signals is configured as trigger_in), ro_trigger (works only if at least one of the signals is configured as ro_trigger_in), triggered_gating (works only if one ofthe signals is configured as gate_in and one as trigger_in).

settings sett Sets detector settings. Can be: standard fast highgain (depending on trheshold energy and maximum count rate: please refere to manual for limit values!);

threshold ev Sets detector threshold in eV. Should be half of the beam energy. It is precise only if the detector is calibrated

vthreshold dac Sets detector threshold in DAC units. A very rough calibration is dac=800-10*keV

exptime t Sets the exposure time per frame (in s)

period t Sets the frames period (in s)

delay t Sets the delay after trigger (in s)

gates n Sets the number of gates per frame

frames n Sets the number of frames per cycle (e.g. after each trigger)

cycles n Sets the number of cycles (e.g. number of triggers)

probes n Sets the number of probes to accumulate (max 3)

dr n Sets the dynamic range - can be (1,) 4, 8,16 or 24 bits

flags mode Sets the readout flags - can be none or storeinram

flatfield fname Sets the flatfield file name - none disable flat field corrections

ratecorr t Sets the rate corrections with dead time t ns (0 unsets, -1 uses default dead time for chosen settings

badchannels fname Sets the badchannels file name - none disable bad channels corrections

angconv fname Sets the angular conversion file name

globaloff o sets the fixed angular offset of your encoder - should be almost constant!

fineoff o sets a possible angular offset of your setup - should be small but can be senseful to modify

binsize s sets the binning size of the angular conversion (otherwise defaults from the angualr conversion constants)

angdir i sets the angular direction of the detector (i can be 1 or -1 - by default 1, channel 0 is smaller angle)

positions np (pos0 pos1...posnp) Sets the number of positions at which the detector is moved during the acquisition and their values

startscript script sets a script to be executed at the beginning of the measurements (e.g. open shutter). *none* unsets. Parameters will be parsed as `script nrun=i par=spar` where i is the run number and spar is the value of startscriptpar.

stopscript script sets a script to be executed at the end of the measurement (e.g. close shutter). *none* unsets. Parameters will be parsed as `script nrun=i par=spar` where i is the run number and spar is the value of stopscriptpar.

startscriptpar spar sets a parameter passed to the start script as string with the syntax par=spar. Its meaning must be interpreted inside the script!

stopscriptpar spar sets a parameter passed to the start script as string with the syntax par=spar. Its meaning must be interpreted inside the script!

scan0script script Sets a scan script to be executed at higher level. Script can be none (unset), threshold (change threshold DAC values for all modules), energy (change energy threshold DAC values using calibration for each module), trimbits (change trimbits for all channels) or any script (e.g changing temperature or moving sample) which will be called with the syntax `script nrun=i fn=fname var=val par=spar` where i is the file index, fname is the file name val is the current value of the scan variable and spar is the value of the scan parameter

scan1script script Sets a scan script to be executed at lower level. Script can be none (unset), threshold (change threshold DAC values for all modules), energy (change energy threshold DAC values using calibration for each module), trimbits (change trimbits for all channels) or any script (e.g changing temperature or moving sample) which will be called with the syntax `script nrun=i fn=fname var=val par=spar` where i is the file index, fname is the file name val is the current value of the scan variable and spar is the value of the scan parameter

scan0par spar sets the scan parameter to be passed to scan0script as a string with syntax par=spar. Its meaning has to be interpreted insode the script!

scan1par spar sets the scan parameter to be passed to scan1script as a string with syntax par=spar. Its meaning has to be interpreted insode the script!

scan0prec i sets the precision of the scan variable in order to properly generate the file names for scan0

scan1prec i sets the precision of the scan variable in order to properly generate the file names for scan1

scan0steps n (f0 f1..fn) sets the steps for the scan0script. n is the number of steps and the following values are the step values.

scan1steps n (f0 f1..fn) sets the steps for the scan1script. n is the number of steps and the following values are the step values.

scan0range mi ma st generates the steps for the scan0script in the range mi to ma with step st (is mi smaller than ma specify a negative step)

scan1range mi ma st generates the steps for the scan1script in the range mi to ma with step st (is mi smaller than ma specify a negative step)

scriptbefore script sets the script to be executed before each acquisition (before all positions) with the syntax `script nrun=i fn=fname par=spar sv0=svar0 sv1=svar1 p0=s` where i is the file index, fname is the file name, sva0, svar1 are the current values of the scan variables 0 and 1, spar0, spar1 are tthe scan parameter 0 and 1. *none* unsets.

scriptafter script sets the script to be executed after each acquisition (after all positions) with the syntax `script nrun=i fn=fname par=spar sv0=svar0 sv1=svar1 p0=spar0 p` where i is the file index, fname is the file name, sva0, svar1 are the current values of the scan variables 0 and 1, spar0, spar1 are tthe scan parameter 0 and 1. *none* unsets.

scriptbeforepar spar sets the parameter to be passed to the script before witht he syntax par=spar

scriptafterpar spar sets the parameter to be passed to the script after witht he syntax par=spar

headerbefore script sets the script to be executed before each acquisition (after moving the detector) with the syntax `script nrun=i fn=fname par=spar` where i is the run number, fname is the file name, spar is the header before parameter. The script is normally used to save a file header. *none* unsets.

headerafter script sets the script to be executed after each acquisition (after each position) with the syntax `script nrun=i fn=fname par=spar` where i is the run number, fname is the file name, spar is the header after parameter. The script is normally used to complete the file header. *none* unsets.

headerbeforepar spar sets the parameter to be passed to the header before script with the syntax par=spar

headerafterpar spar sets the parameter to be passed to the header after script with the syntax par=spar

2.4 Retrieving detector parameters (plus trimming and test modalities)

mythen_get [id[:/-]]var arg
is used to retrieve the detector parameter var e.g. mythen_get 0:exptime returns the exposure time in seconds

help This help

config fname writes the configuration file

parameters fname writes the main detector parameters for the measuremen tin the file

setup rootname writes the complete detector setup (including configuration, trimbits, flat field coefficients, badchannels etc.) is a set of files for which the extension is automatically generated

online return whether the detector is in online (1) or offline (0) state.

status gets the detector status - can be: running, error, transmitting, finished, waiting or idle

data gets all data from the detector (if any) processes them and writes them to file according to the preferences already setup

frame gets a single frame from the detector (if any) processes it and writes it to file according to the preferences already setup

hostname Gets the detector hostname (or IP address)

caldir Gets path of the calibration files

trimdir Gets path of the trim files

outdir directory to which the files will be written by default

fname filename to which the files will be written by default (to which file and position indexes will eventually be attached)

index start index of the files (automatically incremented by the acquisition functions)

nmod Gets number of detector modules

maxmod Gets maximum number of detector modules

extsig:i Gets usage of the external digital signal i. The return value can be: off, gate_in_active_high, gate_in_active_low, trigger_in_rising_edge, trigger_in_falling_edge, ro_trigger_in_rising_edge, ro_trigger_in_falling_edge, gate_out_active_high, gate_out_active_low, trigger_out_rising_edge, trigger_out_falling_edge, ro_trigger_out_rising_edge, ro_trigger_out_falling_edge

timing Sets the timing mode of the detector. Can be auto, gating (works only if at least one of the signals is configured as gate_in), trigger (works only if at least one of the signals is configured as trigger_in), ro_trigger (works only if at least one of the signals is configured as ro_trigger_in), triggered_gating (works only if one ofthe signals is configured as gate_in and one as trigger_in).

modulenumber Gets the module serial number

moduleversion Gets the module version

detectornumber Gets the detector number (MAC address)

detectorversion Gets the detector firmware version

softwareversion Gets the detector software version

digitest:i Makes a digital test of the detector module i. Returns 0 if it succeeds

bustest Makes a test of the detector bus. Returns 0 if it succeeds

settings Gets detector settings. Can be: standard fast highgain undefined

threshold Gets detector threshold in eV. It is precise only if the detector is calibrated

vthreshold Gets detector threshold in DAC units. A very rough calibration is dac=800-10*keV

exptime Gets the exposure time per frame (in s)

period Gets the frames period (in s)

delay Gets the delay after trigger (in s)

gates Gets the number of gates per frame

frames Gets the number of frames per cycle (e.g. after each trigger)

cycles Gets the number of cycles (e.g. number of triggers)

probes Gets the number of probes to accumulate (max 3)

timestamp Gets the internal time stamp of the nex frame acquired (i.e. during an acquisition, all timestamps of the frames are stored in a FIFO which can be read after the acquisition - returns -1 if the FIFO is empty)

dr Gets the dynamic range

trim:mode fname Trims the detector and writes the trimfile fname.snxxx. mode can be: noise beam improve fix offline - Check that the start conditions are OK!!!

flatfield fname returns whether the flat field corrections are enabled and if so writes the coefficients to the specified filename. If fname is none it is not written

ratecorr returns wether the rate corrections are enabled and what is the dead time used in ns

badchannels fname returns wether the bad channels corrections are enabled and if so writes the bad channels to the specified filename. If fname is none it is not written

angconv fname returns wether the angular conversion is enabled and if so writes the angular conversion coefficients to the specified filename. If fname is none, it is not written

globaloff returns the fixed angular offset of your encoder - should be almost constant!

fineoff returns a possible angualr offset of your setup - should be small but can be senseful to modify

binsize returns the binning size of the angular conversion

angdir gets the angular direction of the detector (can be 1 or -1 - by default 1, channel 0 is smaller angle)

positions returns the number of positions at which the detector is moved during the acquisition and their values

startscript script sets a script to be executed at the beginning of the measurements (e.g. open shutter). *none* unsets. Parameters will be parsed as `script nrun=i par=spar` where i is the run number and spar is the value of startscriptpar.

stopscript returns the script to be executed at the end of the measurement (e.g. close shutter). *none* unsets. Parameters will be parsed as `script nrun=i par=spar` where i is the run number and spar is the value of stopscriptpar.

startscriptpar returns the parameter passed to the start script as string with the syntax par=spar. Its meaning must be interpreted inside the script!

stopscriptpar returns the parameter passed to the start script as string with the syntax par=spar. Its meaning must be interpreted inside the script!

scan0script returns the scan script to be executed at higher level. Script can be none (unset), threshold (change threshold DAC values for all modules), energy (change energy threshold DAC values using calibration for each module), trimbits (change trimbits for all channels) or any script (e.g changing temperature or moving sample) which will be called with the syntax `script nrun=i fn=fname var=val par=spar` where i is the file index, fname is the file name val is the current value of the scan variable and spar is the value of the scan parameter

scan1script returns the scan script to be executed at lower level. Script can be none (unset), threshold (change threshold DAC values for all modules), energy (change energy threshold DAC values using calibration for each module), trimbits (change trimbits for all channels) or any script (e.g changing temperature or moving sample) which will be called with the syntax `script nrun=i fn=fname var=val par=spar` where i is the file index, fname is the file name val is the current value of the scan variable and spar is the value of the scan parameter

scan0par returns the scan parameter to be passed to scan0script as a string with syntax par=spar. Its meaning has to be interpreted insode the script!

scan1par returns the scan parameter to be passed to scan1script as a string with syntax par=spar. Its meaning has to be interpreted insode the script!

scan0prec returns the precision of the scan variable in order to properly generate the file names for scan0

scan1prec returns the precision of the scan variable in order to properly generate the file names for scan1

scan0steps returns the steps for the scan0script. n is the number of steps and the following values are the step values.

scan1steps returns the steps for the scan1script. n is the number of steps and the following values are the step values.

scan0range returns the steps for the scan0script. n is the number of steps and the following values are the step values.

scan1range returns the steps for the scan1script. n is the number of steps and the following values are the step values.

scriptbefore returns the script to be executed before each acquisition (before all positions) with the syntax `script nrun=i fn=fname par=spar sv0=svar0 sv1=svar1 p0=spar0` where i is the file index, fname is the file name, sva0, svar1 are the current values of the scan variables 0 and 1, spar0, spar1 are tthe scan parameter 0 and 1.

scriptafter returns the script to be executed after each acquisition (after all positions) with the syntax `script nrun=i fn=fname par=spar sv0=svar0 sv1=svar1 p0=spar0 p1=` where i is the file index, fname is the file name, sva0, svar1 are the current values of the scan variables 0 and 1, spar0, spar1 are tthe scan parameter 0 and 1.

scriptbeforepar returns the parameter to be passed to the script before witht he syntax par=spar

scriptafterpar returns the parameter to be passed to the script after witht he syntax par=spar

headerbefore returns the script to be executed before each acquisition (after moving the detector) with the syntax `script nrun=i fn=fname par=spar` where i is the run number, fname is the file name, spar is the header before parameter. The script is normally used to save a file header.

headerafter returns the script to be executed after each acquisition (after each position) with the syntax `script nrun=i fn=fname par=spar` where i is the run number, fname is the file name, spar is the header after parameter. The script is normally used to complete the file header.

headerbeforepar returns the parameter to be passed to the header before script with the syntax par=spar

headerafterpar returns the parameter to be passed to the header after script with the syntax par=spar

2.5 Tips

Mandatory setup

First of all you should setup the hostname and the detector size and dynamic range:

```
mythen_put hostname mcs1x00
mythen_get nmod
mythen_get dr
```

You should also tell the program where to find the default trimbits files and calibration files:

```
mythen_put trimdir /scratch/trimbits
mythen_get caldir /scratch/calibration
```

To chose the detector settings (e.g. standard):

```
mythen_put settings standard
```

In case `mythen_get settings` does not answer correctly, it most probably means that there is a problem in the architecture or setting of *trimdir* and *caldir* (see section 1.6).

Acquisition setup

You need to setup where the files will be written to

```
mythen_put outdir /scratch
mythen_put fname run
mythen_put index 0
```

this way your files will al be named /scracth/run_i.dat where is starts from 0 and is automatically incremented.

You will then need to setup the detector threshold and settings, the exposure time, the number of real time frames and eventually how many real time frames should be acquired:

```
mythen_put settings standard
mythen_put threshold 6000
mythen_put exptime 1.
mythen_put frames 10
```

In this case 10 consecutive 1s frames will be acquired. External gating and triggering or more advanced acquisition modes are not explained here.

Acquiring

There are two ways of acquiring data.
The first is fully automatic and freezes the terminal until the acquisition is finished:

```
mythen_acquire 0
```

This is particulary indicated for fast real time acquisitions.

If you want to acquire few long frames you can run:

```
mythen_put status start
```

and the poll the detector status using

```
mythen_get status
```

if the answer is either transmitting or finished, the data are ready to be downloaded from the detector. This can be done using either:

```
mythen_get frame
```

where a single data frame is downloaded or

```
mythen_get data
```

where all data present on the detector are downloaded. This is not indicated when many short real time frames should be acquired since the detector memory would be full before finishing the acquisition since the download time is so limited.

Data processing

Flat field and rate corrections can be applied direcly by simply selecting:

```
mythen_put flatield myflatfield.raw
mythen_put ratecorr -1
```

Concerning the angular conversion, it is very reccomended that the users edit the file usersFunctions.cpp contained in the folder slsDetectorSoftware/usersFunctions. In the file it is possible to modify the function used for calculating the angular conversion and the ones used for interfacing with the diffractometer equipment i.e. reading the encoder fo the detector position, the ionization chanmbers etc.

It is also possible to configure some scans/scripts to be executed during the acquisition. They will be normally called as system calls except for threshold, energy and trimbits scans.